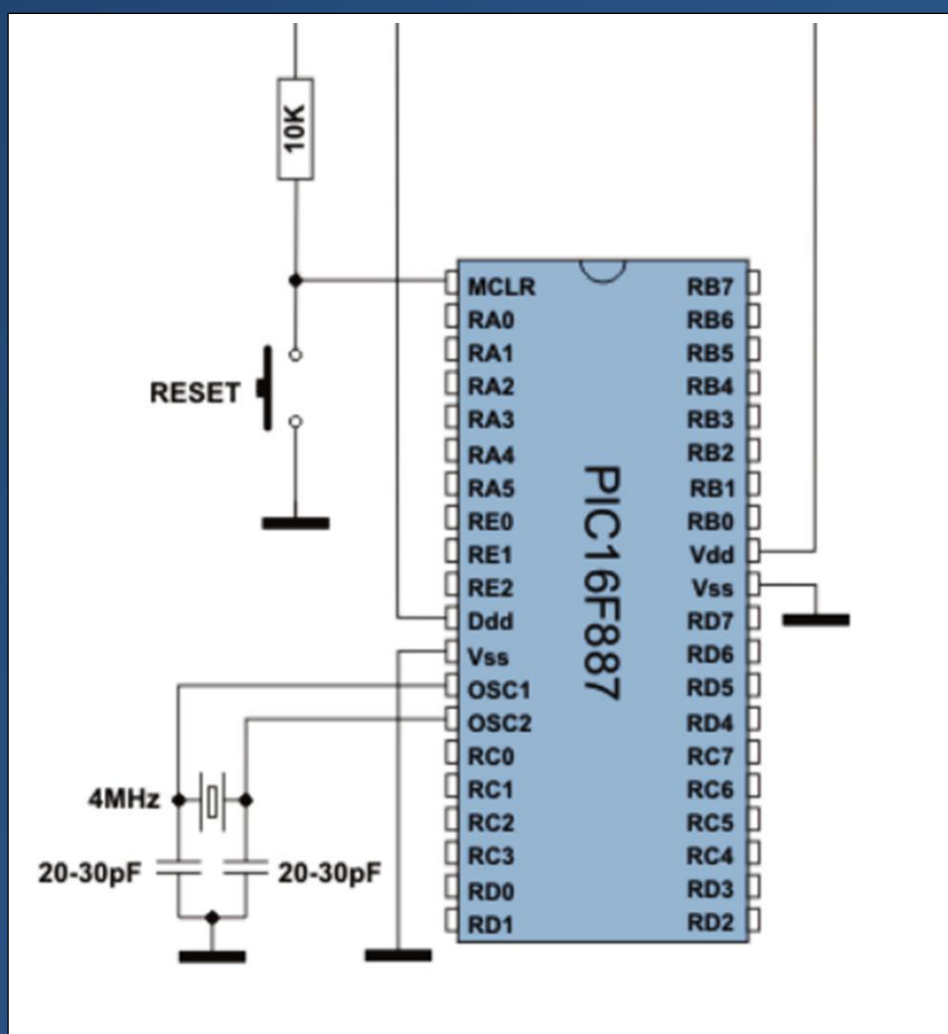
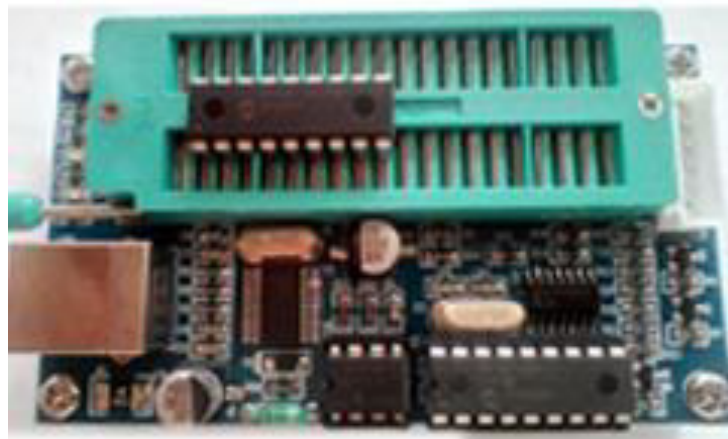


Microcontrolador PIC16F887

Características básicas y conexionado



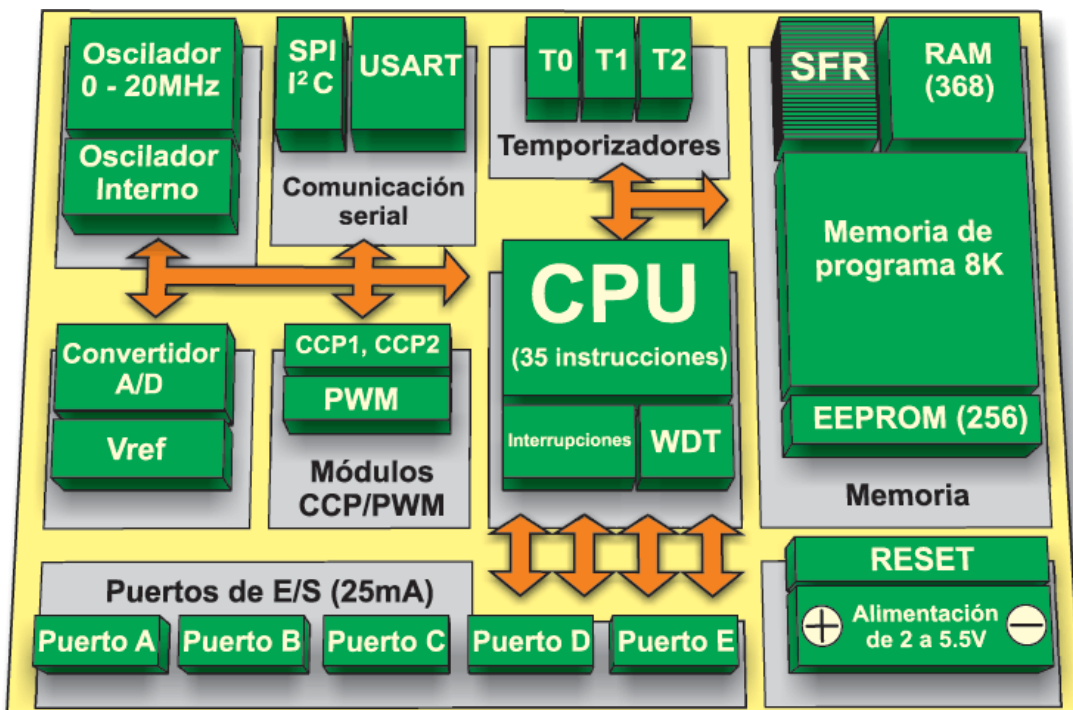
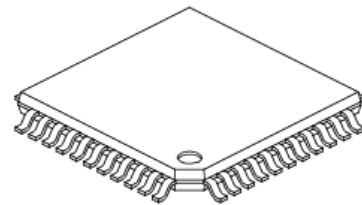
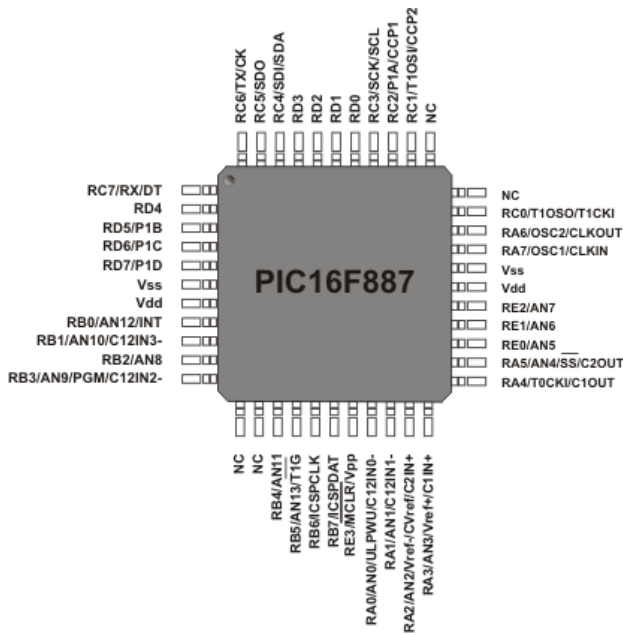
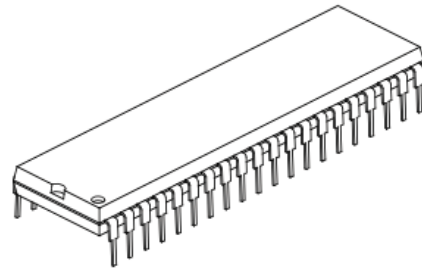
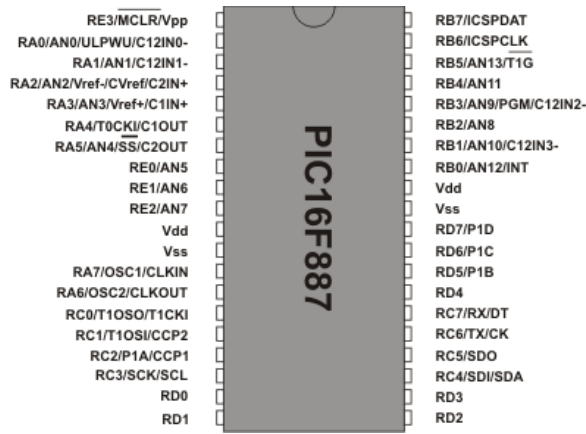
Características básicas del PIC 16F887



PIC16F887

CARACTERÍSTICAS BÁSICAS DEL PIC16F887

- **Arquitectura RISC**
 - El microcontrolador cuenta con solo 35 instrucciones diferentes
 - Todas las instrucciones son uni-ciclo excepto por las de ramificación
- **Frecuencia de operación 0-20 MHz**
- **Oscilador interno de alta precisión**
 - Calibrado de fábrica
 - Rango de frecuencia de 8MHz a 31KHz seleccionado por software
- **Voltaje de la fuente de alimentación de 2.0V a 5.5V**
 - Consumo: 220uA (2.0V, 4MHz), 11uA (2.0 V, 32 KHz) 50nA (en modo de espera)
- **Ahorro de energía en el *Modo de suspensión***
- **Brown-out Reset (BOR) con opción para controlar por software**
- **35 pines de entrada/salida**
 - alta corriente de fuente y de drenador para manejo de LED
 - resistencias *pull-up* programables individualmente por software
 - interrupción al cambiar el estado del pin
- **Memoria ROM de 8K con tecnología FLASH**
 - El chip se puede re-programar hasta 100.000 veces
- **Opción de *programación serial en el circuito***
 - El chip se puede programar incluso incorporado en el dispositivo destino.
- **256 bytes de memoria EEPROM**
 - Los datos se pueden grabar más de 1.000.000 veces
- **368 bytes de memoria RAM**
- **Convertidor A/D:**
 - 14 canales
 - resolución de 10 bits
- **3 temporizadores/contadores independientes**
- **Temporizador perro guardián**
- **Módulo comparador analógico con**
 - Dos comparadores analógicos
 - Referencia de voltaje fija (0.6V)
 - Referencia de voltaje programable en el chip
- **Módulo PWM incorporado**
- **Módulo USART mejorado**
 - Soporta las comunicaciones seriales RS-485, RS-232 y LIN2.0
 - Auto detección de baudios
- **Puerto Serie Síncrono Maestro (MSSP)**
 - Soporta los modos SPI e I2C



DESCRIPCIÓN DE PINES

La mayoría de los pines del microcontrolador PIC16F887 son multipropósito como se muestra en la figura anterior. Por ejemplo, la asignación RA3/AN3/Vref+/C1IN+ para el quinto pin del microcontrolador indica que éste dispone de las siguientes funciones:

- RA3 Tercera entrada/salida digital del puerto A
- AN3 Tercera entrada analógica
- Vref+ Referencia positiva de voltaje
- C1IN+ Entrada positiva del comparador C1

La funcionalidad de los pines presentados anteriormente es muy útil puesto que permite un mejor aprovechamiento de los recursos del microcontrolador sin afectar a su funcionamiento. Estas funciones de los pines no se pueden utilizar simultáneamente, sin embargo se pueden cambiar en cualquier instante durante el funcionamiento.

Las siguientes tablas se refieren al microcontrolador DIP de 40 pines.

Nombre	Número (DIP 40)	Función	Descripción
RE3/MCLR/Vpp	1	RE3	Entrada de propósito general en el puerto PORTE
		MCLR	Pin de reinicio. El nivel lógico bajo en este pin reinicia al microcontrolador
		Vpp	Voltaje de programación
RA0/AN0/ULPWU/C12IN0-	2	RA0	E/S de propósito general en el puerto PORTA
		AN0	Entrada del canal 0 del convertidor A/D
		ULPWU	Entrada de desactivar el modo de espera
		C12IN0-	Entrada negativa del comparador C1 o C2
RA1/AN1/C12IN1-	3	RA1	E/S de propósito general en el puerto A
		AN1	Canal 1 del convertidor A/D
		C12IN1-	Entrada negativa del comparador C1 o C2
RA2/AN2/Vref-/CVref/C2IN+	4	RA2	E/S de propósito general en el puerto PORTA
		AN2	Canal 2 del convertidor A/D
		Vref-	Entrada de referencia negativa de voltaje del convertidor A/D
		CVref	Salida de referencia de voltaje del comparador
		C2IN+	Entrada positiva del comparador C2
RA3/AN3/Vref+/C1IN+	5	RA3	E/S de propósito general en el puerto PORTA
		AN3	Canal 3 del convertidor A/D
		Vref+	Entrada de referencia positiva de voltaje del convertidor A/D
		C1IN+	Entrada positiva del comparador C1
RA4/T0CKI/C1OUT	6	RA4	E/S de propósito general en el puerto PORTA
		T0CKI	Entrada de reloj del temporizador T0
		C1OUT	Salida del comparador C1
RA5/AN4/SS/C2OUT	7	RA5	E/S de propósito general en el puerto PORTA
		AN4	Canal 4 del convertidor A/D
		SS	Entrada del módulo SPI (<i>Selección del esclavo</i>)
		C2OUT	Salida del comparador C2
RE0/AN5	8	RE0	E/S de propósito general en el puerto PORTE
		AN5	Canal 5 del convertidor A/D
RE1/AN6	9	RE1	E/S de propósito general en el puerto PORTE
		AN6	Canal 6 del convertidor A/D
RE2/AN7	10	RE2	E/S de propósito general en el puerto PORTE
		AN7	Canal 7 del convertidor A/D
Vdd	11	+	Suministro de voltaje positivo
Vss	12	-	Tierra (ground - GND)

Nombre	Número (DIP 40)	Función	Descripción
RA7/OSC1/CLKIN	13	RA7	E/S de propósito general en el puerto PORTA
		OSC1	Entrada del oscilador de cristal
		CLKIN	Entrada del reloj externo
RA6/OSC2/CLKOUT	14	OSC2	Salida del oscilador del cristal
		CLKO	Salida en la que se presenta la señal $F_{osc}/4$
		RA6	E/S de propósito general en el puerto PORTA
RC0/T1OSO/T1CKI	15	RC0	E/S de propósito general en el puerto PORTC
		T1OSO	Salida del oscilador del temporizador 1
		T1CKI	Entrada de reloj del temporizador 1
RC1/T1OSO/T1CKI	16	RC1	E/S de propósito general en el puerto PORTC
		T1OSI	Entrada del oscilador del temporizador 1
		CCP2	E/S de los módulos CCP1 y PWM1
RC2/P1A/CCP1	17	RC2	E/S de propósito general en el puerto PORTC
		P1A	Salida del módulo PWM
		CCP1	E/S de los módulos CCP1 y PWM1
RC3/SCK/SCL	18	RC3	E/S de propósito general en el puerto PORTC
		SCK	E/S de reloj del módulo MSSP en el modo SPI
		SCL	E/S de reloj del módulo MSSP en el modo I^2C
RD0	19	RD0	E/S de propósito general en el puerto PORTD
RD1	20	RD1	E/S de propósito general en el puerto PORTD
RD2	21	RD2	E/S de propósito general en el puerto PORTD
RD3	22	RD3	E/S de propósito general en el puerto PORTD
RC4/SDI/SDA	23	RC4	E/S de propósito general en el puerto PORTC
		SDI	Entrada <i>Data</i> del módulo MSSP en el modo SPI
		SDA	E/S <i>Data</i> del módulo MSSP en el modo I^2C
RC5/SDO	24	RC5	E/S de propósito general en el puerto PORTC
		SDO	Salida <i>Data</i> del módulo MSSP en el modo SPI
RC6/TX/CK	25	RC6	E/S de propósito general en el puerto PORTC
		TX	Salida asíncrona del módulo USART
		CK	Reloj síncrono del módulo USART
RC7/RX/DT	26	RC7	E/S de propósito general en el puerto PORTC
		RX	Entrada asíncrona del módulo USART
		DT	Datos del módulo USART en modo síncrono

Nombre	Número (DIP 40)	Función	Descripción
RD4	27	RD4	E/S de propósito general en el puerto PORTD
RD5/P1B	28	RD5	E/S de propósito general en el puerto PORTD
		P1B	Salida del módulo PWM
RD6/P1C	29	RD6	E/S de propósito general en el puerto PORTD
		P1C	Salida del módulo PWM
RD7/P1D	30	RD7	E/S de propósito general en el puerto PORTD
		P1D	Salida del módulo PWM
Vss	31	-	Tierra (GND)
Vdd	32	+	Suministro de voltaje positivo
RB0/AN12/INT	33	RB0	E/S de propósito general en el puerto PORTB
		AN12	Canal 12 del convertidor A/D
		INT	Interrupción externa
RB1/AN10/C12INT3-	34	RB1	E/S de propósito general en el puerto PORTB
		AN10	Canal 10 del convertidor A/D
		C12INT3-	Entrada negativa de los comparadores C1 o C2
RB2/AN8	35	RB2	E/S de propósito general en el puerto PORTB
		AN8	Canal 8 del convertidor A/D
RB3/AN9/PGM/C12IN2-	36	RB3	E/S de propósito general en el puerto PORTB
		AN9	Canal 9 del convertidor A/D
		PGM	Habilita la programación del chip
		C12IN2-	Entrada negativa de los comparadores C1 o C2
RB4/AN11	37	RB4	E/S de propósito general en el puerto PORTB
		AN11	Canal 11 del convertidor A/D
RB5/AN13/T1G	38	RB5	E/S de propósito general en el puerto PORTB
		AN13	Canal 13 del convertidor A/D
		T1G	Entrada externa del temporizador 11
RB6/ICSPCLK	39	RB6	E/S de propósito general en el puerto PORTB
		ICSPCLK	Entrada de reloj de programación serial
RB7/ICSPDAT	40	RB7	E/S de propósito general en el puerto PORTB
		ICSPDAT	Pin de E/S para introducir los datos durante la programación ICSP™

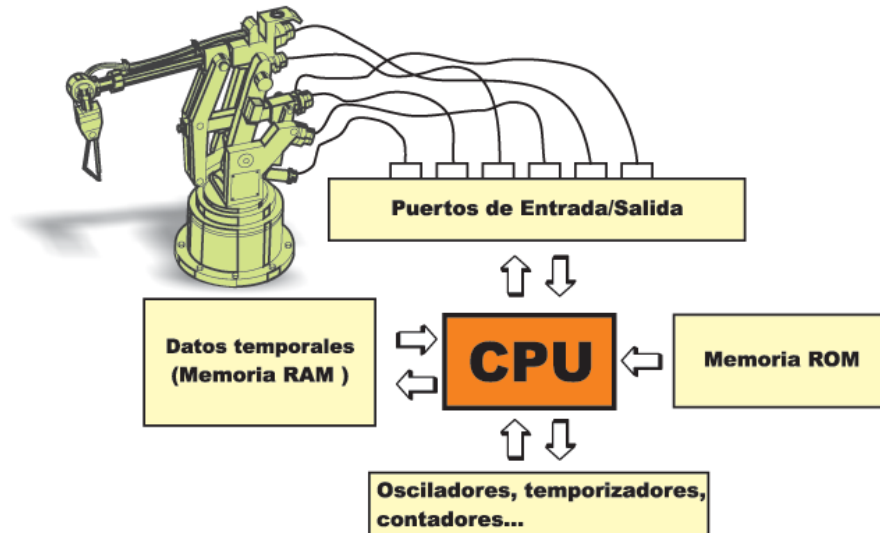
UNIDAD CENTRAL DE PROCESAMIENTO (CPU)

Con el propósito de explicar en forma clara y concisa, sin describir profundamente el funcionamiento de la CPU, vamos a hacer constar que la CPU está fabricada con la tecnología RISC ya que esto es un factor importante al decidir qué microcontrolador utilizar.

RISC es un acrónimo derivado del inglés *Reduced Instruction Set Computer*, lo que proporciona al PIC16F887 dos grandes ventajas:

- La CPU cuenta con sólo 35 instrucciones simples. Cabe decir que para poder programar otros microcontroladores en lenguaje ensamblador es necesario saber más de 200 instrucciones

- El tiempo de ejecución es igual para casi todas las instrucciones y tarda 4 ciclos de reloj. La frecuencia del oscilador se estabiliza por un cristal de cuarzo. Las instrucciones de salto y de ramificación tardan ocho ciclos de reloj en ejecutarse. Esto significa que si la velocidad de operación del microcontrolador es 20 MHz, el tiempo de ejecución de cada instrucción será 200nS, o sea, ¡el programa ejecutará 5 millones de instrucciones por segundo!



MEMORIA

El PIC16F887 tiene tres tipos de memoria: ROM, RAM y EEPROM. Como cada una tiene las funciones, características y organización específicas, vamos a presentarlas por separado.

MEMORIA ROM

La memoria ROM se utiliza para guardar permanente el programa que se está ejecutando. Es la razón por la que es frecuentemente llamada “memoria de programa”. El PIC16F887 tiene 8Kb de memoria ROM (en total 8192 localidades). Como la memoria ROM está fabricada con tecnología FLASH, su contenido se puede cambiar al proporcionarle un voltaje de programación especial (13V).

No obstante, no es necesario explicarlo en detalles puesto que se realiza automáticamente por un programa especial en la PC y un simple dispositivo electrónico denominado programador.

Escribir el programa en ensamblador,
(simulación del funcionamiento),
compilar a código máquina

Copiar el programa a memoria ROM



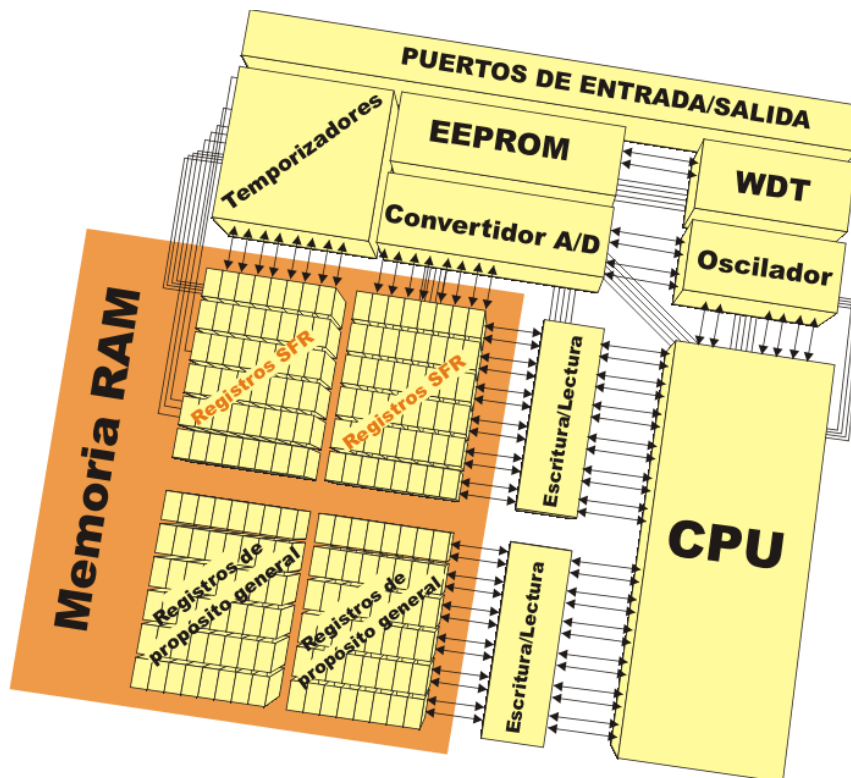
MEMORIA EEPROM

Similar a la memoria de programa, el contenido de memoria EEPROM está permanentemente guardado al apagar la fuente de alimentación. Sin embargo, a diferencia de la ROM, el contenido de la EEPROM se puede cambiar durante el funcionamiento del microcontrolador. Es la razón por la que esta memoria (256 localidades) es perfecta para guardar permanentemente algunos resultados creados y utilizados durante la ejecución del programa.

MEMORIA RAM

Es la tercera y la más compleja parte de la memoria del microcontrolador. En este caso consiste en dos partes: en registros de propósito general y en los registros de funciones especiales (SFR). Todos estos registros se dividen en cuatro bancos de memoria de los que vamos a hablar más tarde en este capítulo.

Aunque los dos grupos de registros se ponen a cero al apagar la fuente de alimentación, además están fabricados de la misma forma y se comportan de la manera similar, sus funciones no tienen muchas cosas en común.



REGISTROS DE PROPÓSITO GENERAL

Los registros de propósito general se utilizan para almacenar los datos temporales y los resultados creados durante el funcionamiento. Por ejemplo, si el programa realiza el conteo (de los productos en una cadena de montaje), es necesario tener un registro que represente lo que en la vida cotidiana llamamos "suma". Como el microcontrolador no es nada creativo, es necesario especificar la dirección de un registro de propósito general y asignarle esa función. Se debe crear un programa simple para incrementar el valor de este registro por 1, después de que cada producto haya pasado por el sensor.

Ahora el microcontrolador puede ejecutar el programa ya que sabe qué es y dónde está la suma que se va a incrementar. De manera similar, a cada variable de programa se le debe pre-asignar alguno de los registros de propósito general.

/ En esta secuencia, la variable en el registro sum se aumenta cada vez que se lleve un uno (1) lógico en el pin de entrada RB0. */*

```
...
if (PORTB.0 = 1) // Comprobar si el pin RB0 está a uno
sum++;          // Si está, el valor de la variable se aumenta por 1
...            // Si no está, el programa sale de la sentencia if
...
```

REGISTROS DE FUNCIONES ESPECIALES (SFR)

Los registros de funciones especiales son también parte de la memoria RAM. A diferencia de los registros de propósito general, su propósito es predeterminado durante el proceso de fabricación y no se pueden cambiar. Como los bits están conectados a los circuitos particulares en el chip (convertidor A/D, módulo de comunicación serial, etc), cualquier cambio de su contenido afecta directamente al funcionamiento del microcontrolador o de alguno de los circuitos.

Por ejemplo, el registro ADCON0 controla el funcionamiento del convertidor A/D. Al cambiar los bits se determina qué pin del puerto se configurará como la entrada del convertidor, el momento del inicio de la conversión así como la velocidad de la conversión.

Otra característica de estas localidades de memoria es que tienen nombres (tanto los registros como sus bits), lo que simplifica considerablemente el proceso de escribir un programa. Como el lenguaje de programación de alto nivel puede utilizar la lista de todos los registros con sus direcciones exactas, basta con especificar el nombre de registro para leer o cambiar su contenido.

// En esta secuencia, el contenido de los registros TRISC y PORTC será modificado

```
...
TRISC = 0x00    // un cero lógico (0) se escribe en el registro TRISC (todos
                // los pines del puerto PORTC se configuran como salidas)
PORTC = 0b01100011 // cambio de estado lógico de todos los pines del puerto PORTC
...
```

BANCOS DE LA MEMORIA RAM

La memoria RAM está dividida en cuatro bancos. Antes de acceder a un registro al escribir un programa (para leer o cambiar su contenido), es necesario seleccionar el banco que contiene ese registro. Más tarde vamos a tratar dos bits del registro STATUS utilizados para selección del banco. Para simplificar el funcionamiento, los SFR utilizados con más frecuencia tienen la misma dirección en todos los bancos, lo que permite accederlos con facilidad.

Dir.	Nombre	Dir.	Nombre	Dir.	Nombre	Dir.	Nombre
00h	INDF	80h	INDF	100h	INDF	180h	INDF
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h	WDTCON	185h	SRCON
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h	CM1CON0	187h	BAUDCTL
08h	PORTD	88h	TRISD	108h	CM2CON0	188h	ANSEL
09h	PORTE	89h	TRISE	109h	CM2CON1	189h	ANSELH
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDAT	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	No utilizado
0Fh	TMR1H	8Fh	OSCCON	10Fh	EEADRH	18Fh	No utilizado
10h	T1CON	90h	OSCTUNE	110h		190h	
11h	TMR2	91h	SSPCON2				
12h	T2CON	92h	PR2				
13h	SSPBUF	93h	SSPADD				
14h	SSPCON	94h	SSPSTAT				
15h	CCPR1L	95h	WPUB				
16h	CCPR1H	96h	IOCB				
17h	CCP1CON	97h	VRCON				
18h	RCSTA	98h	TXSTA				
19h	TXREG	99h	SPBRG				
1Ah	RCREG	9Ah	SPBRGH				
1Bh	CCPR2L	9Bh	PWM1CON		Registros de propósito general		Registros de propósito general
1Ch	CCPR2H	9Ch	ECCPAS				
1Dh	CCP2CON	9Dh	PSTRCON				
1Eh	ADRESH	9Eh	ADRESL		96 bytes		96 bytes
1Fh	ADCON0	9Fh	ADCON1				
20h		A0h					
	Registros de propósito general		Registros de propósito general				
7Fh	96 bytes	FFh	80 bytes	17Fh		1EFh	
Banco 0		Banco 1		Banco 2		Banco 3	

Trabajar con bancos puede ser difícil sólo si se escribe un programa en lenguaje ensamblador. Al utilizar el lenguaje de programación de alto nivel como es C y el compilador como es *mikroC PRO for PIC*, basta con escribir el nombre del registro. A partir de esa información, el compilador selecciona el banco necesario. Las instrucciones apropiadas para la selección del banco serán incorporadas en el código durante el proceso de la compilación. Hasta ahora usted ha utilizado sólo el lenguaje ensamblador y esta es la primera vez que utiliza el compilador C, verdad? Es una noticia maravillosa, no lo cree?

Registros SFR del banco 0

Dirección	Nombre	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00h	INDF	Registro indirecto							
01h	TMR0	Registro del temporizador 0							
02h	PCL	Byte menos significativo del contador de programa							
03h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C
04h	FSR	Puntero indirecto a dirección							
05h	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
09h	PORTE	-	-	-	-	RE3	RE2	RE1	RE0
0Ah	PCLATH	-	-	-	5 bits superiores del contador de programa				
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
0Ch	PIR1	-	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
0Dh	PIR2	OSFIF	C2IF	C1IF	EEIF	BCLIF	ULPWUIF	-	CCP2IF
0Eh	TMR1L	Byte menos significativo del temporizador 1 de 16 bits							
0Fh	TMR1H	Byte más significativo del temporizador 1 de 16 bits							
10h	T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
11h	TMR2	Registro del temporizador 2							
12h	T2CON	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
13h	SSPBUF	Registro del puerto serie síncrono (búfer de recepción/registro de transmisión)							
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
15h	CCPR1L	Byte menos significativo del Registro 1 para capturar/comparar la señal PWM							
16h	CCPR1H	Byte más significativo del Registro 1 para capturar/comparar la señal PWM							
17h	CCP1CON	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
19h	TXREG	Registro del módulo EUSART de transmisión de datos							
1Ah	RCREG	Registro del módulo EUSART de recepción de datos							
1Bh	CCPR2L	Byte menos significativo del Registro 2 para capturar/comparar la señal PWM							
1Ch	CCPR2H	Byte más significativo del registro 1 para capturar/comparar la señal PWM							
1Dh	CCP2CON	-	-	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0
1Eh	ADRESH	Registro de almacenar el byte superior del resultado de la conversión A/D							
1Fh	ADCON0	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON

Registros SFR del banco 1

Dirección	Nombre	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
80h	INDF	Registro indirecto							
81h	OPTION_REG	RBPV	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
82h	PCL	Byte menos significativo del contador de programa							
83h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C
84h	FSR	Puntero indirecto a dirección							
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
87h	TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
88h	TRISD	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0
89h	TRISE	-	-	-	-	TRISE3	TRISE2	TRISE1	TRISE0
8Ah	PCLATH	-	-	-	5 bits superiores del contador de programa				
8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
8Ch	PIE1	-	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
8Dh	PIE2	OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	-	CCP2IE
8Eh	PCON	-	-	ULPWUE	SBOREN	-	-	POR	BOR
8Fh	OSCCON	-	IRCF2	IRCF1	IRCF0	OSTS	HTS	LTS	SCS
90h	OSCTUNE	-	-	-	TUN4	TUN3	TUN2	TUN1	TUN0
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
92h	PR2	Registro del período del temporizador 2							
93h	SSPADD	Registro de la dirección del puerto serie síncrono (modo I ² C)							
93h	SSPMSK	MSK7	MSK6	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0
94h	SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF
95h	WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0
96h	IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0
97h	VRCON	VREN	VROE	VRR	VRSS	VR3	VR2	VR1	VR0
98h	TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
99h	SPBRG	BRG7	BRG6	BRG5	BRG4	BRG3	BRG2	BRG1	BRG0
9Ah	SPBRGH	BRG15	BRG14	BRG13	BRG12	BRG11	BRG10	BRG9	BRG8
9Bh	PWM1CON	PRSEN	PDC6	PDC5	PDC4	PDC3	PDC2	PDC1	PDC0
9Ch	ECCPAS	ECCPASE	ECCPAS2	ECCPAS1	ECCPAS0	PSSAC1	PSSAC0	PSSBD1	PSSBD0
9Dh	PSTRCON	-	-	-	STRSYNC	STRD	STRC	STRB	STRA
9Eh	ADRESL	Registro de almacenar el byte inferior del resultado de la conversión A/D							
9Fh	ADCON1	ADFM	-	VCFG1	VCFG0	-	-	-	-

Registros SFR del banco 2

Dirección	Nombre	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
100h	INDF	Registro indirecto							
101h	TMR0	Registro del temporizador T0							
102h	PCL	Byte menos significativo del contador de programa							
103h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C
104h	FSR	Puntero indirecto a dirección							
105h	WDTCON	-	-	-	WDTPS3	WDTPS2	WDTPS1	WDTPS0	SWDTEN
106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
107h	CM1CON0	C1ON	C1OUT	C1OE	C1POL	-	C1R	C1CH1	C1CH0
108h	CM2CON0	C2ON	C2OUT	C2OE	C2POL	-	C2R	C2CH1	C2CH0
109h	CM2CON1	MC1OUT	MC2OUT	C1RSEL	C2RSEL	-	-	T1GSS	C2SYNC
10Ah	PCLATH	-	-	-	5 bits superiores del contador de programa				
10Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
10Ch	EEDAT	EEDAT7	EEDAT6	EEDAT5	EEDAT4	EEDAT3	EEDAT2	EEDAT1	EEDAT0
10Dh	EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0
10Eh	EEDATH	-	-	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0
10Fh	EEADRH	-	-	-	EEADRH4	EEADRH3	EEADRH2	EEADRH1	EEADRH0

Registros SFR del banco 2

Dirección	Nombre	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
180h	INDF	Registro indirecto							
181h	OPTION_REG	RBPV	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
182h	PCL	Byte menos significativo del contador de programa							
183h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C
184h	FSR	Puntero indirecto a dirección							
185h	SRCON	SR1	SR0	C1SEN	C2REN	PULSS	PULSR	-	FVREN
186h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
187h	BAUDCTL	ABDOVF	RCIDL	-	SCKP	BRG16	-	WUE	ABDEN
188h	ANSEL	ANS7	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0
189h	ANSELH	-	-	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8
19Ah	PCLATH	-	-	-	5 bits superiores del contador de programa				
19Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
19Ch	EECON1	EEPGRD	-	-	-	WRERR	WREN	WR	RD
19Dh	EECON2	Registro de control 2 del módulo EEPROM (no es un registro real)							

PILA

Una parte de la RAM utilizada como pila consiste de ocho registros de 13 bits. Antes de que el microcontrolador se ponga a ejecutar una subrutina (instrucción CALL) o al ocurrir una interrupción, la dirección de la primera siguiente instrucción en ser ejecutada se coloca en la pila (se apila), o sea, en uno de los registros. Gracias a eso, después de ejecutarse una subrutina o una interrupción, el microcontrolador “sabe” dónde continuar con la ejecución de programa. Esta dirección se borra (se desapila) después de volver al programa, ya que no es necesario guardarla, disponiendo automáticamente esas localidades de la pila para un uso futuro.

Cabe tener en mente que el dato se apila circularmente. Esto significa que después de que se apile ocho veces, la novena vez se sobrescribe el valor que se almacenó al apilar el dato por primera vez. La décima vez que se apile, se sobrescribe el valor que se almacenó al apilar el dato por segunda vez etc. Datos sobrescritos de esta manera no se pueden recuperar. Además, el programador no puede acceder a estos registros para hacer escritura/lectura. No hay ningún bit de estado para indicar el estado de desbordamiento o subdesbordamiento de pila. Por esta razón hay que tener un especial cuidado al escribir un programa.

Vamos a hacerlo en mikroC...

/ Al entrar o al salir de la instrucción en ensamblador del programa, el compilador no va a guardar los datos en el banco de la RAM actualmente activo. Esto significa que en esta sección de programa la selección de banco depende de los registros SFR utilizados. Al volver a la sección de programa escrito en C, los bits de control RP0 y RP1 deben devolver el estado que tenían antes de la ejecución del código en lenguaje ensamblador. En este ejemplo, el problema se soluciona al utilizar la variable auxiliar saveBank que guarda el estado de estos dos bits*/*

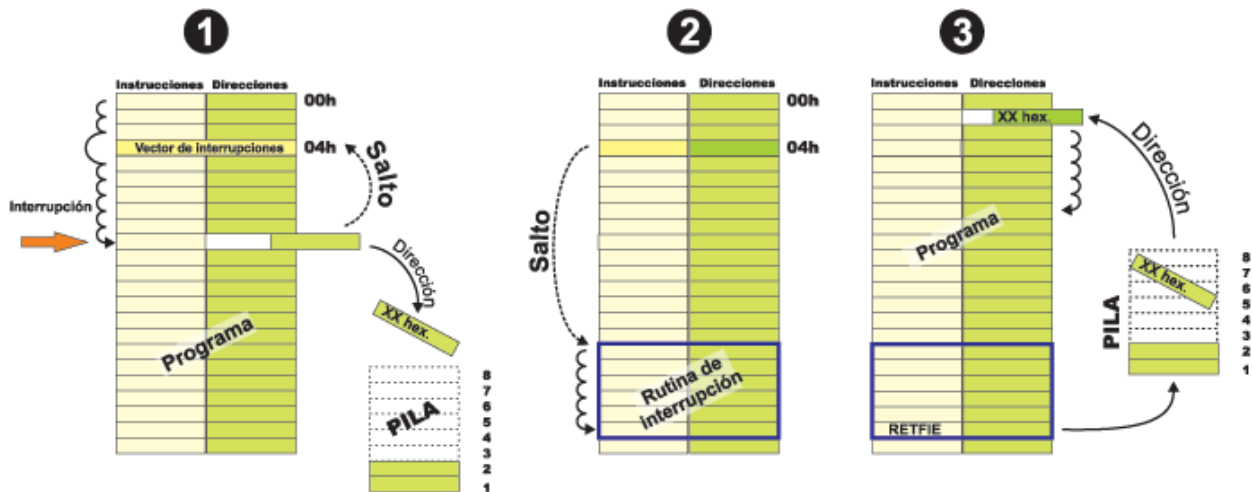
```
saveBank = STATUS & 0b01100000; // Guardar el estado de los bits RP0 y RP1
                // (bits 5 y 6 del registro STATUS)
asm {           // Inicio de la secuencia en ensamblador
...
                // Código ensamblador
...
}               // Final de la secuencia en ensamblador
STATUS &= 0b10011111; // Bits RP0 y RP1 devuelven su estado original
STATUS |= saveBank;
...
...
```

SISTEMA DE INTERRUPCIONES

Al aparecer una petición de interrupción lo primero que hace el microcontrolador es ejecutar la instrucción actual después de que se detiene el proceso de ejecución de programa. Como resultado, la dirección de memoria de programa actual se apila automáticamente y la dirección por defecto (predefinida por el fabricante) se escribe en el contador de programa. La localidad en la que el programa continúa con la ejecución se le denomina vector de interrupción. En el caso del microcontrolador PIC16F887 esta dirección es 0x0004h. Como se muestra en la siguiente figura la localidad que contiene el vector de interrupción se omite durante la ejecución de programa regular.

Una parte de programa que se ejecutará al hacer una petición de interrupción se le denomina rutina de interrupción. Su primera instrucción se encuentra en el vector de interrupción. Cuánto tiempo tardará en ejecutar esta subrutina y cómo será depende de la destreza del programador así como de la fuente de interrupción. Algunos microcontroladores tienen más de un vector de interrupción (cada petición de interrupción tiene su vector), pero en este caso sólo hay uno. En consecuencia, la primera parte de la rutina de interrupción consiste en detectar la fuente de interrupción.

Por fin, al reconocer la fuente de interrupción y al terminar de ejecutar la rutina de interrupción el microcontrolador alcanza la instrucción **RETFIE**, toma la dirección de la pila y continúa con la ejecución de programa desde donde se interrumpió.



mikroC reconoce una rutina de interrupción que se ejecutará como la función `void interrupt()`. El cuerpo de la función, o sea, rutina de interrupción, debe ser escrito por el usuario.

```
void interrupt() { // Interrupt routine
    cnt++; // Interrupt causes variable cnt to be incremented by 1
}
```

Cómo utilizar los registros SFR

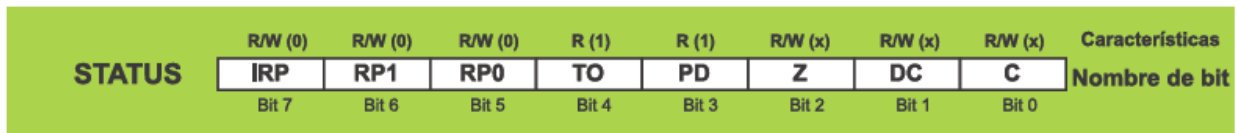
Supongamos que usted ha comprado ya un microcontrolador y que tiene una buena idea de cómo utilizarlo... La lista de los registros SFR así como de sus bits es muy larga. Cada uno controla algún proceso. En general, parece como una gran tabla de control con un gran número de instrumentos e interruptores. ¿Ahora está preocupado de cómo conseguir aprender acerca de todos ellos? Es poco probable, pero no se preocupe, ¡Usted no tiene que hacerlo! Los microcontroladores son tan potentes que se parecen a los supermercados: ofrecen tantas cosas a bajos precios y a usted solo le toca elegir las que necesita. Por eso, seleccione el campo en que está interesado y examine sólo lo que necesita. Cuando entienda completamente el funcionamiento de hardware, examine los registros SFR encargados de controlarlo (normalmente son unos pocos).

Como todos los dispositivos tienen un tipo de sistema de control el microcontrolador tiene sus "palancas" con las que usted debe estar familiarizado para ser capaz de utilizarlos correctamente. Por supuesto, estamos hablando de los registros SFR desde los que el proceso de programación se inicia y en los que el mismo termina.

PRINCIPALES REGISTROS SFR

El siguiente texto describe los principales registros SFR del microcontrolador PIC16F887. Los bits de cada registro controlan los circuitos diferentes dentro del chip, así que no es posible clasificarlos en grupos especiales. Por esta razón, se describen junto con los procesos que controlan.

Registro STATUS



Leyenda

R/W	Bit de lectura/escritura
R	Bit de solo lectura
(0)	Después del reinicio, el bit se pone a cero
(1)	Después del reinicio, el bit se pone a uno
(x)	Después del reinicio, el estado de bit es desconocido

El registro STATUS contiene: el estado aritmético de datos en el registro W, el estado RESET, los bits para seleccionar el banco para los datos de la memoria.

- **IRP** - Registro de selección de Banco (usado para direccionamiento indirecto)
 - **1** - Bancos 0 y 1 son activos (localidades de memoria 00h-FFh)
 - **0** - Bancos 2 y 3 son activos (localidades de memoria 100h-1FFh)
- **RP1,RP0** - Registro de selección de banco (usado para direccionamiento directo).

RP1 RP0 Banco activo

0	0	Banco 0
0	1	Banco 1
1	0	Banco 2
1	1	Banco 3

- **TO - Time-out bit (bit de salida del temporizador perro guardián)**
 - **1** - Después de encender el microcontrolador, después de ejecutarse la instrucción **CLRWDT** que reinicia al WDT (temporizador perro guardián) o después de ejecutarse la instrucción **SLEEP** que pone al microcontrolador en el modo de bajo consumo.
 - **0** - Después de acabarse el tiempo del WDT.
- **PD - Power-down bit (bit de apagado)**
 - **1** - Después de encender el microcontrolador, después de ejecutar la instrucción **CLRWDT** que reinicia al WDT.
 - **0** - Después de ejecutarse la instrucción **SLEEP** que pone al microcontrolador en el modo de bajo consumo.
- **Z - Zero bit (bit cero)**
 - **1** - El resultado de una operación lógica o aritmética es 0.
 - **0** - El resultado de una operación lógica o aritmética es distinto de 0.
- **DC - Digit carry/borrow bit (bit de acarreo/préstamo de dígito)** cambia al sumar o al restar si ocurre un "desbordamiento" o un "préstamo" en el resultado.
 - **1** - Hubo acarreo del cuarto bit de orden bajo (nibble bajo) en el resultado.
 - **0** - No hubo acarreo del cuarto bit de orden bajo (nibble bajo) en el resultado.

- **C - Carry/Borrow bit** (bit de acarreo/préstamo) cambia al sumar o al restar si ocurre un "desbordamiento" o un "préstamo" en el resultado, o sea si el resultado es mayor de 255 o menor de 0.
 - **1** - Ocurrió acarreo en el bit más significativo (MSB) del resultado.
 - **0** - No ocurrió acarreo en el bit más significativo (MSB) del resultado.

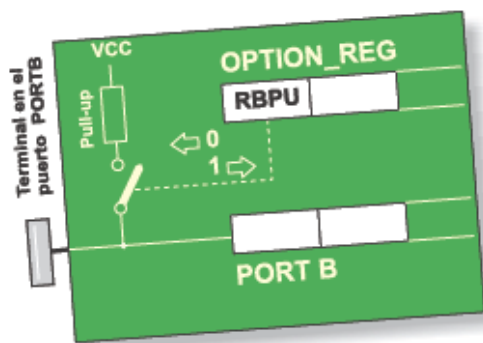
Registro OPTION_REG

OPTION_REG	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Características
	RBPUP	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

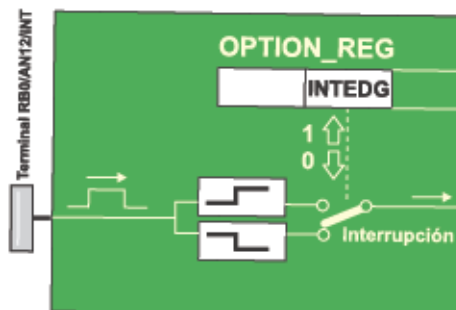
Leyenda

R/W	Bit de lectura/escritura
(1)	Después del reinicio, el bit se pone a uno

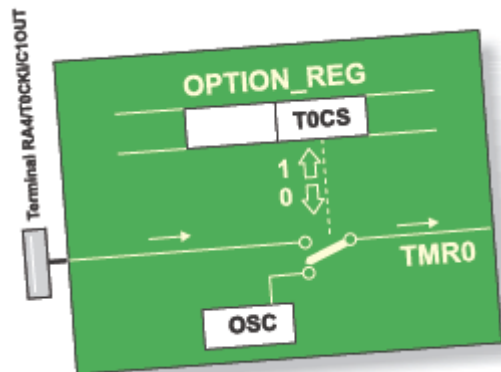
El registro OPTION_REG contiene varios bits de control para configurar el pre-escalador del Temporizador 0/WDT, el temporizador Timer0, la interrupción externa y las resistencias pull-up en el puerto PORTB.



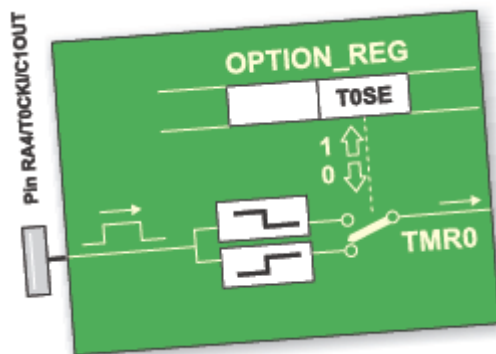
- **RBPUP - Port B Pull up Enable bit (resistencia Pull Up Puerto B)**
 - **1** - Desactivadas.
 - **0** - Activadas.



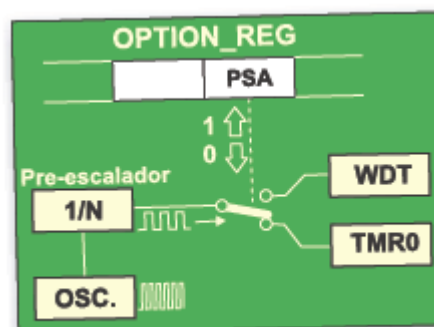
- **INTEDG - Interrupt Edge Select bit (bit selector de flanco activo de la interrupción externa)**
 - **1** - Interrupción por flanco ascendente en el RB0/INT.
 - **0** - Interrupción por flanco descendente en el RB0/INT.



- **T0CS - TMR0 Clock Source Select bit (bit selector de tipo de reloj para el Timer0)**
 - **1** - Pulsos introducidos a través del TOCKI (contador).
 - **0** - Pulsos de reloj internos Fosc/4 (temporizador).



- **T0SE - TMR0 Source Edge Select bit (bit selector de tipo de flanco en TOCKI)** selecciona el flanco (ascendente o descendente) contado por el temporizador Timer0 por el pin RA4/T0CKI.
 - **1** - Incrementa en flanco descendente en el pin TOCKI.
 - **0** - Incrementa en flanco ascendente en el pinTOCKI.



- **PSA - Prescaler Assignment bit** asigna el pre-escalador (hay sólo uno) al temporizador o al WDT.
 - **1** - Pre - escalador se le asigna al WDT.
 - **0** - Pre - escalador se le asigna al temporizador Timer0.

PS2, PS1, PS0 Prescaler Rate Select bits (bit selector del valor del divisor de frecuencia)

El valor del divisor de frecuencia se selecciona al combinar estos tres bits. Como se muestra en la siguiente tabla, el valor del divisor de frecuencia se le asigna al temporizador (Timer0) o al temporizador perro guardián (WDT).

PS2	PS1	PS0	TMR0	WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Para conseguir el valor del divisor de frecuencia 1:1 cuando el temporizador Timer0 cuenta pulsos, el preescalador debe ser asignado al WDT. En consecuencia, el temporizador Timer0 no utiliza el pre-escalador, sino que cuenta directamente los pulsos generados por el oscilador, lo que era el objetivo.

Vamos a hacerlo en mikroC...

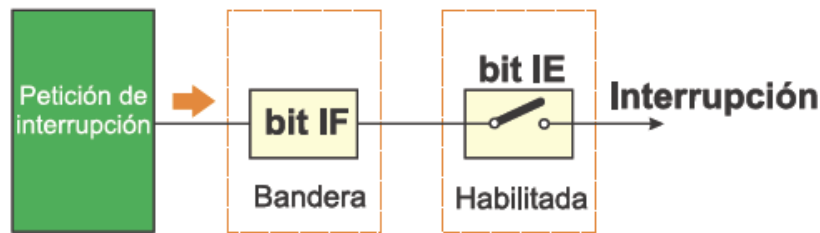
/ Si el comando CLRWDT no se ejecuta,
el WDT va a reiniciar al microcontrolador cada 32.768 uS (f=4 MHz) */*

```
void main() {
  OPTION_REG = 0b00001111; // Pre-escalador está asignado al WDT (1:128)
  asm CLRWDT; // Comando en ensamblador para reiniciar al WDT
  ...
  ...// El tiempo entre estos dos comandos CLRWDT no debe exceder 32.768 microsegundos (128x256)
  ...
  asm CLRWDT; // Comando en ensamblador para reiniciar al WDT
  ...
  ...// El tiempo entre estos dos comandos CLRWDT no debe exceder 32.768 microsegundos (128x256)
  ...
  asm CLRWDT; // Comando en ensamblador para reiniciar al WDT
}
```

REGISTROS DEL SISTEMA DE INTERRUPCIÓN

Al llegar la petición de interrupción, no significa que una interrupción ocurrirá automáticamente, puesto que debe ser habilitada por el usuario (por el programa) también. Por esta razón, hay bits especiales utilizados para habilitar o deshabilitar interrupciones. Es fácil de reconocerlos por las letras IE contenidas en sus nombres (Interrupt Enable - Interrupción habilitada). Además, cada interrupción se asocia con otro bit denominado bandera que indica que una petición de interrupción ha llegado sin verificar si está habilitada.

Asimismo, se reconocen con facilidad por las dos últimas letras contenidas en sus nombres - IF (Interrupt Flag - Bandera de interrupción).



Como hemos visto, toda la idea es muy simple y eficiente. Al llegar la petición de interrupción, primero el bit de bandera se pone a uno.

Si el bit IE apropiado está a cero (0), esta condición será ignorada completamente. De lo contrario, ocurre una interrupción. Si varias fuentes de interrupción están habilitadas, es necesario detectar la activa antes de que la rutina de interrupción se ponga a ejecutar. La detección de la fuente se realiza al comprobar los bits de las banderas.

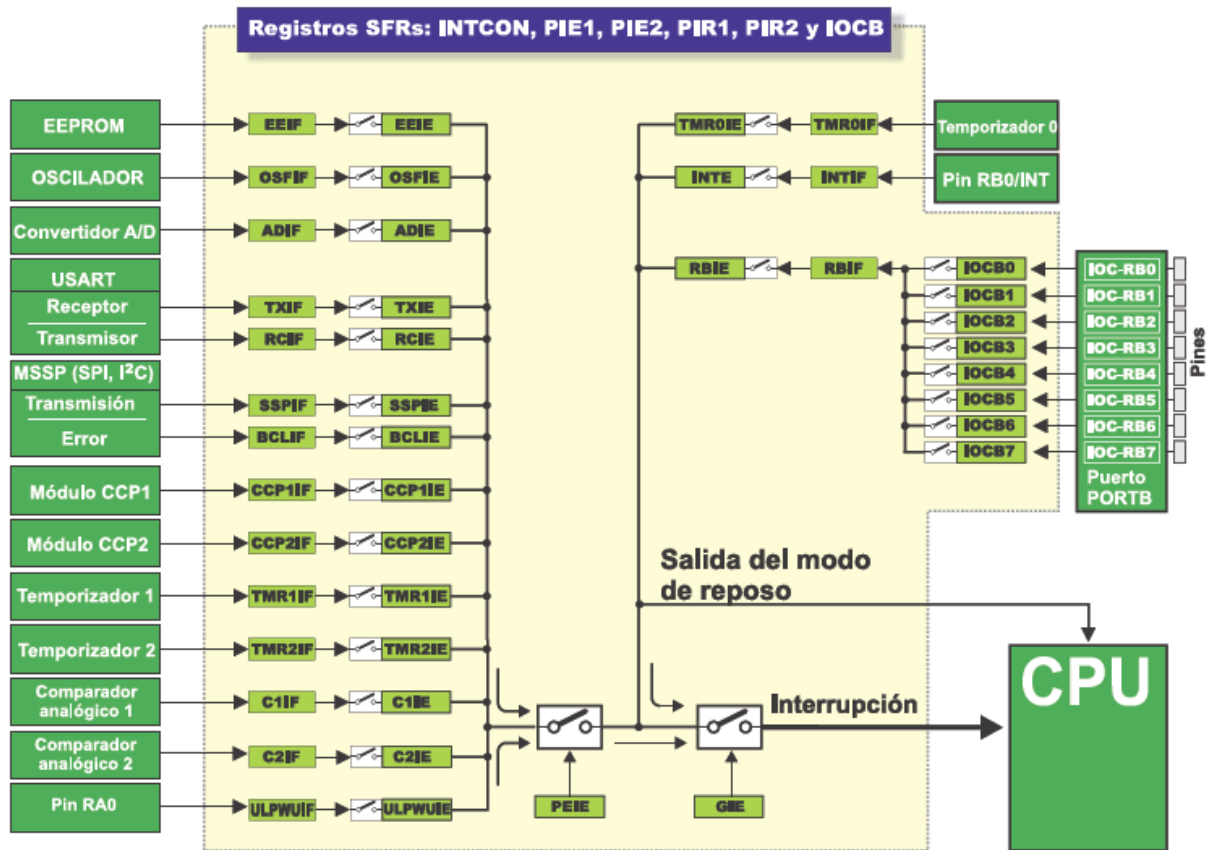
Cabe destacar que los bits de cada bandera no se ponen a cero automáticamente, sino por el software, mientras que la ejecución de la rutina de interrupción continúa ejecutándose. Si no hacemos caso a este detalle, ocurrirá otra interrupción inmediatamente después de volver al programa principal, aunque no hay más peticiones de ejecución. Simplemente, la bandera, así como el bit IE, se quedan en uno.

Todas las fuentes de interrupción típicas para el microcontrolador PIC16F887 se muestran en la siguiente página. Fíjese en lo siguiente:

El bit **GIE** habilita/deshabilita simultáneamente las interrupciones no enmascaradas.

El **PEIE bit** habilita/deshabilita las interrupciones no enmascaradas de periféricos. Esto no se refiere al temporizador Timer0 y a las fuentes de interrupción del puerto PORTB.

Para habilitar una interrupción causada por el cambio del estado lógico en el puerto PORTB, es necesario habilitarla para cada bit por separado. En este caso, los bits del registro **IOCB** se comportan como los bits IE de control.



Sistema de interrupción del microcontrolador PIC16F887

Registro INTCON

El registro INTCON contiene varios bits de habilitación y de bandera para el desbordamiento en el registro TMR0, e interrupciones por el cambio del estado en el puerto PORTB y las interrupciones externas en el pin INT.

INTCON	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)	Características
	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Legenda

R/W	Bit de lectura/escritura
(0)	Después del reinicio, el bit se pone a cero
(x)	Después del reinicio, el estado de bit es desconocido

- **GIE - Global Interrupt Enable bit** - (bit de habilitación de interrupciones globales) controla simultáneamente todas las fuentes de interrupciones posibles.
 - **1** - Habilita las interrupciones no enmascaradas.
 - **0** - Deshabilita las interrupciones no enmascaradas.
- **PEIE - Peripheral Interrupt Enable bit** (bit de habilitación de interrupciones periféricas) es similar al bit GIE, sin embargo controla interrupciones habilitadas por los periféricos. Eso significa que no influye en interrupciones causadas por el

temporizador Timer0 o por el cambio del estado en el puerto PORTB o por el cambio en el pin RB0/INT.

- **1** - Habilita las interrupciones periféricas no enmascaradas.
 - **0** - Deshabilita las interrupciones periféricas no enmascaradas.
- **T0IE - TMR0 Overflow Interrupt Enable bit** (bit de habilitación de interrupciones por el desbordamiento del temporizador Timer0) controla interrupciones causadas por el desbordamiento del Timer0.
 - **1** - Habilita interrupciones por Timer0.
 - **0** - Deshabilita interrupciones por Timer0.
 - **INTE - RB0/INT External Interrupt Enable bit** (bit de habilitación de la interrupción externa en RB0) controla interrupciones causadas por el cambio del estado lógico en el pin de entrada RB0/INT (interrupción externa).
 - **1** - Habilita interrupciones externas INT.
 - **0** - Deshabilita interrupciones externas INT.
 - **RBIE - RB Port Change Interrupt Enable bit** (bit de habilitación de interrupciones por cambios en el puerto PORTB). Cuando se configuran como entradas, los pines en el puerto PORTB pueden causar una interrupción al cambiar el estado lógico (no importa si se produce bajada o subida de tensión, lo que importa es que se produce un cambio). Este bit determina si una interrupción va a ocurrir.
 - **1** - Habilita interrupciones por cambio en el puerto PORTB.
 - **0** - Deshabilita interrupciones por cambio en el puerto PORTB.
 - **T0IF - TMR0 Overflow Interrupt Flag bit** (bit de bandera de interrupción por el desbordamiento del Timer0) detecta el desbordamiento en el registro del temporizador Timer0, o sea el contador se pone a cero.
 - **1** - En el registro del Timer0 ha ocurrido desbordamiento (esta bandera debe volverse a 0 por software).
 - **0** - En el registro del Timer0 no ha ocurrido desbordamiento.
 - **INTF - RB0/INT External Interrupt Flag bit** (bit de bandera de interrupción externa en INT) detecta el cambio en el estado lógico en el pin INT.
 - **1** - Ha ocurrido una interrupción externa por INT (esta bandera debe volverse a 0 por software)
 - **0** - No ha ocurrido una interrupción externa por INT.
 - **RBIF - RB Port Change Interrupt Flag bit** (bit de bandera de interrupción por cambio en el puerto RB) detecta cualquier cambio del estado lógico de alguno de los pines de entrada en el puerto PORTB.
 - **1** - Al menos uno de los pines de E/S de propósito general en el puerto PORTB ha cambiado de valor. Después de leer el puerto PORTB, el bit RBIF debe volverse a 0 por software).
 - **0** - Ninguno de los pines de E/S de propósito general en el puerto PORTB ha cambiado de valor.

Vamos a hacerlo en mikroC...

// El pin PORTB.4 se configura como una entrada sensible al cambio del estado lógico.

```
void initMain() {
ANSEL = ANSELH = 0; // Todos los pines de E/S se configuran como digitales
PORTB = 0; // Todos los pines del puerto PORTB se ponen a cero
TRISB = 0b00010000; // Todos los pines del puerto PORTB menos PORTB.4 se
// configuran como salidas

RBIE = 1; // Se habilitan las interrupciones por el cambio en el puerto PORTB
IOCB4 = 1; // Se habilita la interrupción por el cambio en el pin 4 en el
// puerto PORTB
GIE = 1; // Se habilita la interrupción global
... // Desde este punto, se produce una interrupción con cualquier cambio
... // del estado lógico en el pin PORTB.4
...
}
```

Registro PIE1

El registro PIE1 contiene los bits de habilitación de interrupciones periféricas.

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
PIE1	-	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
								Nombre de bit

Legenda

-	Bit no implementado
R/W	Bit de lectura/escritura
(0)	Después del reinicio, el bit se pone a cero

- **ADIE - A/D Converter Interrupt Enable bit (bit de habilitación de interrupciones del convertidor A/D).**
 - **1** - Habilita la interrupción ADC.
 - **0** - Deshabilita la interrupción ADC.
- **RCIE - EUSART Receive Interrupt Enable bit (bit de habilitación de interrupciones de recepción del EUSART).**
 - **1** - Habilita la interrupción de recepción del EUSART.
 - **0** - Deshabilita la interrupción de recepción del EUSART.
- **TXIE - EUSART Transmit Interrupt Enable bit (bit de habilitación de interrupciones de transmisión del EUSART).**
 - **1** - Habilita la interrupción de transmisión del EUSART.
 - **0** - Deshabilita la interrupción de transmisión del EUSART.
- **SSPIE - Master Synchronous Serial Port (MSSP) Interrupt Enable bit - (bit de habilitación de la interrupción del puerto serie síncrono maestro (MSSP) habilita generar una petición de interrupción después de cada transmisión de datos por el módulo de comunicación serie síncrona (modo SPI o I2C).**
 - **1** - Habilita la interrupción del MSSP.
 - **0** - Deshabilita la interrupción del MSSP.

- **CCP1IE - CCP1 Interrupt Enable bit** (bit de habilitación de la interrupción del módulo 1 de Comparación/Captura/PWM - CCP1) permite generar una petición de interrupción en el módulo CCP1 utilizado para procesamiento de la señal PWM.
 - **1** - Habilita la interrupción CCP1.
 - **0** - Deshabilita la interrupción CCP1.

- **TMR2IE - TMR2 to PR2 Match Interrupt Enable bit (bit de habilitación de la interrupción de igualdad entre TMR2 y PR2)**
 - **1** - Habilita la interrupción de igualdad entre TMR2 y PR2.
 - **0** - Deshabilita la interrupción de igualdad entre TMR2 y PR2.

- **TMR1IE - TMR1 Overflow Interrupt Enable bit** (bit de habilitación de la interrupción de desbordamiento del temporizador Timer1) habilita generar una petición de interrupción después de cada desbordamiento en el registro del temporizador Timer1, o sea el contador se pone a cero.
 - **1** - Habilita la interrupción de desbordamiento del temporizador Timer1.
 - **0** - Deshabilita la interrupción de desbordamiento del temporizador Timer1.

Vamos a hecerlo en mikroC...

/ Se produce una interrupción con cada desbordamiento en el registro del temporizador1 que consiste en TMR1H y TMR1L. En cada rutina de interrupciones, la variable cnt será incrementada por 1 */*

```

unsigned short cnt; // Definir la variable cnt
void interrupt() // Inicio de la rutina de interrupción
cnt++; // Al producirse una interrupción, la cnt se
// incrementa por 1
PIR1.TMR1IF = 0; // El bit TMR1IF se reinicia
TMR1H = 0x80; // A los registros del temporizador TMR1H y TMR1L se les
TMR1L = 0x00; // devuelven sus valores iniciales
} // Final de la rutina de interrupción

void main() {
ANSEL = ANSELH = 0; // Todos los pines de E/S se configuran como digitales
T1CON = 1; // Encender el temporizador Timer
PIR1.TMR1IF = 0; // El bit TMR1IF se pone a cero
TMR1H = 0x80; // Establecer los valores iniciales para el temporizador Timer1
TMR1L = 0x00;
PIE1.TMR1IE = 1; // Habilitar la interrupción al producirse un
// desbordamiento en el Timer1
cnt = 0; // Reiniciar la variable cnt
INTCON = 0xC0; // Habilitar la interrupción (los bits GIE y PEIE)
...

```

Registro PIE2

El registro PIE2 también contiene varios bits de habilitación de interrupciones.

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
PIE2	OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	-	CCP2IE
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
								Nombre de bit

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
(0)	Después del reinicio, el bit se pone a cero

- **OSFIE - Oscillator Fail Interrupt Enable bit (bit de habilitación de la interrupción de fallo en el oscilador)**
 - **1** - Habilita la interrupción de fallo en el oscilador.
 - **0** - Deshabilita la interrupción de fallo en el oscilador.
- **C2IE - Comparator C2 Interrupt Enable bit (bit de habilitación de la interrupción del comparador C2)**
 - **1** - Habilita la interrupción del comparador C2.
 - **0** - Deshabilita la interrupción del comparador C2.
- **C1IE - Comparator C1 Interrupt Enable bit (bit de habilitación de la interrupción del comparador C1)**
 - **1** - Habilita la interrupción del comparador C1
 - **0** - Deshabilita la interrupción del comparador C1.
- **EEIE - EEPROM Write Operation Interrupt Enable bit (bit de habilitación de la interrupción de escritura en la memoria EEPROM)**
 - **1** - Habilita la interrupción de escritura en la memoria EEPROM.
 - **0** - Deshabilita la interrupción de escritura en la memoria EEPROM.
- **BCLIE - Bus Collision Interrupt Enable bit (bit de habilitación de la interrupción de colisión de bus)**
 - **1** - Habilita la interrupción de colisión de bus.
 - **0** - Deshabilita la interrupción de colisión de bus.
- **ULPWUIE - Ultra Low-Power Wake-up Interrupt Enable bit (bit de habilitación de la interrupción para salir del modo de ultra bajo consumo - la reactivación)**
 - **1** - Habilita la interrupción para salir del modo de ultra bajo consumo.
 - **0** - Deshabilita la interrupción para salir del modo de ultra bajo consumo.
- **CCP2IE - CCP2 Interrupt Enable bit (bit de habilitación de la interrupción del módulo 2 de Comparación/Captura/PWM (CCP2))**
 - **1** - Habilita la interrupción del CCP2.
 - **0** - Deshabilita la interrupción del CCP2.

Vamos a hacerlo en mikroC...

/ El comparador C2 se configura para utilizar los pines RA0 y RA2 como entradas. Al producirse un cambio en la salida del comparador, el pin de salida PORTB.1 cambia el estado lógico en la rutina de interrupción.*/*

```
void interrupt() { // Inicio de la rutina de interrupción
    PORTB.F1 = ~PORTB.F1 ; // La interrupción invertirá el estado lógico del
                        // pin PORTB.1
    PIR2.C2IF = 0; // Bit de bandera de interrupción C2IF se pone a cero
} // Final de la rutina de interrupción

void main() {
    TRISB = 0; // Todos los pines del puerto PORTB se configuran
              // como salidas
    PORTB.F1 = 1; // El pin PORTB.1 se pone a uno
    ANSEL = 0b00000101;; // Los pines RA0/C12IN0- y RA2/C2IN+ son las
                        // entradas analógicas
    ANSELH = 0; // Todos los pines de E/S se configuran como digitales
    CM2CON0.C2CH0 = CM2CON0.C2CH1 = 0; // El pin RA0 se selecciona para ser una
                        // entrada invertida del C2
    PIE2.C2IE = 1; // Habilita la interrupción del comparador C2INT
    CON.GIE = 1; // Interrupción global está habilitada
    CM2CON0.C2ON = 1; // Comparador C2 está habilitado
    ...
    ...
}
```

Registro PIR1

El registro PIR1 contiene los bits de banderas de interrupción.

	R/W (0)	R (0)	R (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
PIR1	-	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
								Nombre de bit

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit se pone a cero

- **ADIF - A/D Converter Interrupt Flag bit (bit de bandera de la interrupción del convertidor A/D)**
 - **1** - Se ha completado una conversión A/D (el bit debe volverse a 0 por software)
 - **0** - No se ha completado una conversión A/D o no ha empezado
- **RCIF - EUSART Receive Interrupt Flag bit (bit de bandera de la interrupción de recepción del EUSART)**
 - **1** - El búfer de recepción del EUSART está lleno. El bit se pone a cero al leer el registro RCREG.
 - **0** - El búfer de recepción del EUSART no está lleno.

- **TXIF - EUSART Transmit Interrupt Flag bit (bit de la interrupción de transmisión del EUSART)**
 - **1** - El búfer de transmisión del EUSART está vacío. El bit se pone a cero al escribir un dato en el registro TXREG.
 - **0** - El búfer de transmisión del EUSART está lleno.

- **SSPIF - Master Synchronous Serial Port (MSSP) Interrupt Flag bit (bit de bandera de la interrupción de puerto serie síncrono maestro)**
 - **1** - Se ha cumplido la condición de ocurrir una interrupción del MSSP al transmitir/ recibir los datos. Difieren dependiendo del modo de operación del MSSP (SPI o I²C). El bit debe ponerse a cero por software antes de volver de la rutina de servicio de interrupciones)
 - **0** - No se ha cumplido ninguna condición de ocurrir una interrupción del MSSP.

- **CCP1IF - CCP1 Interrupt Flag bit (bit de bandera de la interrupción del módulo 1 de Comparación/Captura/PWM (CCP1)).**
 - **1** - Se ha cumplido la condición de la interrupción del CCP1 (CCP1 es una unidad para captar, comparar y generar una señal PWM). Dependiendo del modo de operación (modo captura o modo comparación), se produce una captura o la igualdad en la comparación. En ambos casos, el bit debe volverse a cero por software. El bit no se utiliza en el modo PWM.
 - **0** - No se ha cumplido la condición de la interrupción del CCP1.

- **TMR2IF - Timer2 to PR2 Interrupt Flag bit (bit de bandera de la interrupción de igualdad entre el temporizador Timer2 y el registro PR2)**
 - **1** - Se ha producido igualdad con el valor del TMR2 (registro de 8 bits del temporizador) y el valor del PR2. El bit debe ponerse a cero por software antes de volver de la rutina de servicio de interrupciones).
 - **0** - No se ha producido igualdad con el valor del TMR2 y el valor del PR2.

- **TMR1IF - Timer1 Overflow Interrupt Flag bit (bit de bandera de la interrupción de desbordamiento del temporizador Timer1)**
 - **1** - Se ha producido desbordamiento del Timer1. El bit debe ponerse a cero por software.
 - **0** - No se ha producido desbordamiento del Timer1.

Registro PIR2

El registro PIR2 contiene los bits de banderas da la interrupción.

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
PIR2	OSFIF	C2IF	C1IF	EEIF	BCLIF	ULPWUIF	-	CCP2IF
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Nombre de bit

Leyenda

- Bit no implementado
- R/W Bit de lectura/escritura
- (0) Después del reinicio, el bit se pone a cero

- **OSFIF - Oscillator Fail Interrupt Flag bit (bit de bandera de la interrupción de fallo en el oscilador)**
 - **1** - Se ha producido un fallo en el oscilador del sistema. La entrada de reloj ha sido conmutada al oscilador interno INTOSC. El bit debe ponerse a cero por software.
 - **0** - El oscilador del sistema funciona correctamente.
- **C2IF - Comparator C2 Interrupt Flag bit (bit de bandera de la interrupción del comparador C2)**
 - **1** - La salida del comparador analógico C2 ha sido cambiada (el bit C2OUT). El bit debe ponerse a cero por software.
 - **0** - La salida del comparador analógico C2 no ha sido cambiada.
- **C1IF - Comparator C1 Interrupt Flag bit (bit de bandera de la interrupción del comparador C1)**
 - **1** - La salida del comparador analógico C1 ha sido cambiada (el bit C1OUT). El bit debe ponerse a cero por software.
 - **0** - La salida del comparador analógico C1 no ha sido cambiada.
- **EEIF - EE Write Operation Interrupt Flag bit (bit de bandera de la interrupción de la operación de escritura en la memoria EEPROM)**
 - **1** - La operación de escritura en la memoria EEPROM se ha completado. El bit debe ponerse a cero por software.
 - **0** - La operación de escritura en la memoria EEPROM no se ha completado o todavía no se ha iniciado.
- **BCLIF - Bus Collision Interrupt Flag bit (bit de bandera de la interrupción de colisión de bus en el MSSP)**
 - **1** - Se ha producido una colisión de bus en el MSSP al ser configurado para el modo maestro I2C. El bit debe ponerse a cero por software.
 - **0** - No se ha producido colisión de bus en el MSSP.
- **ULPWUIF - Ultra Low-power Wake-up Interrupt Flag bit (bit de bandera de la interrupción para salir del modo de ultra bajo consumo - la reactivación)**
 - **1** - Se ha cumplido la condición de salir del modo de ultra bajo consumo. El bit debe ponerse a cero por software.
 - **0** - No se ha cumplido la condición de salir del modo de ultra bajo consumo.
- **CCP2IF - CCP2 Interrupt Flag bit (bit de la interrupción del módulo 2 de Comparación/Captura/PWM - CCP2)**
 - **1** - Se ha cumplido la condición de la interrupción del CCP2 (CCP2 es una unidad para captar, comparar y generar una señal PWM). Dependiendo del modo de operación (modo captura o modo comparación), se produce una captura o la igualdad en la comparación. En ambos casos, el bit debe volverse a cero por software. El bit no se utiliza en el modo PWM.
 - **0** - No se ha cumplido la condición de la interrupción del CCP2.

Vamos a hacerlo en mikroC...

// Secuencia de activación del módulo ULPWU

```
void main() {
  PORTA.F0 = 1;           // Pin PORTA.0 se pone a uno
  ANSEL = ANSELH = 0;    // Todos los pines de E/S se configuran como digitales
  TRISA = 0;             // Los pines del puerto PORTA se configuran como salidas
  Delay_ms(1);          // Cargar el capacitor
  PIR2.ULPWUIF = 0;     // Bandera ULPWUIF se pone a cero
  PCON.ULPWUE = 1;      // Habilitar el funcionamiento del módulo ULPWU
  TRISA.F0 = 1;         // PORTA.0 se configura como entrada
  PIE2.ULPWUIE = 1;    // Habilitar la interrupción por el módulo ULPWU
  INTCON.GIE = INTCON.PEIE = 1; // Habilitar todas las interrupciones
  asm SLEEP;           // Pasar al modo de bajo consumo
  ...
  ...
}
```

Registro PCON

El registro PCON contiene solamente dos bits de banderas utilizados para diferenciar entre un Power-on reset (POR), un Brown-out reset (BOR), un reinicio por el temporizador perro guardián (WDT) y un reinicio externo por el pin MCLR.



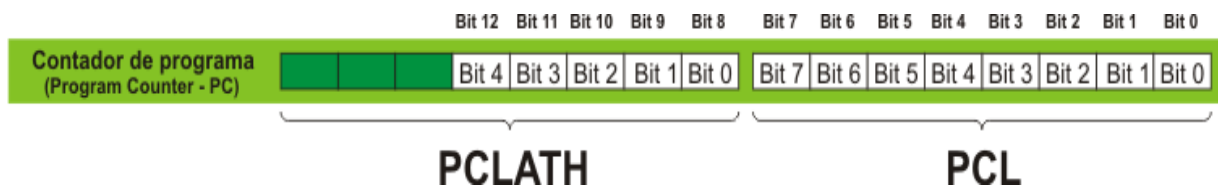
Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
(0)	Después del reinicio, el bit se pone a cero
(1)	Después del reinicio, el bit se pone a uno
(x)	Después del reinicio, el estado del bit es desconocido

- **ULPWUE - Ultra Low-Power Wake-up Enable bit (bit de habilitación para salir del modo de ultra bajo consumo - la reactivación)**
 - **1** - Se habilita salir del modo de ultra bajo consumo.
 - **0** - No se habilita salir del modo de ultra bajo consumo.
- **SBOREN - Software BOR Enable bit (bit de habilitación del BOR por software)**
 - **1** - Se habilita Brown-out reset.
 - **0** - Se deshabilita Brown-out reset.
- **POR - Power-on Reset Status bit (bit de estado Power - on reset)**
 - **1** - No se ha producido Power - on reset.
 - **0** - Se ha producido Power - on reset. El bit debe ponerse a uno por software después de que se haya producido un Power - on reset.
- **BOR - Brown-out Reset Status bit (bit de estado Brown - out reset)**
 - **1** - No se ha producido Brown - out reset.
 - **0** - Se ha producido Brown - out reset. El bit debe ponerse a uno por software después de que se haya producido Brown - out reset.

REGISTROS PCL Y PCLATH

La memoria de programa del PIC16F887 es de 8K y tiene 8192 localidades para el almacenamiento de programa. Por esta razón, el contador de programa debe de ser de 13 bits de anchura ($2^{13} = 8192$). Para habilitar el acceso a una localidad de memoria de programa durante el funcionamiento del microcontrolador, es necesario acceder a su dirección por medio de los registros SFR. Como todos los registros SFR son de 8 bits de anchura, este registro de direccionamiento es creado "artificialmente" al dividir los 13 bits en dos registros independientes, PCLATH y PCL. Si la ejecución de programa no afecta al contador de programa, el valor de este registro va incrementándose automática y constantemente: +1, +1, +1, +1... De esta manera, el programa se ejecuta como está escrito - instrucción a instrucción, seguido por un incremento de dirección constante.

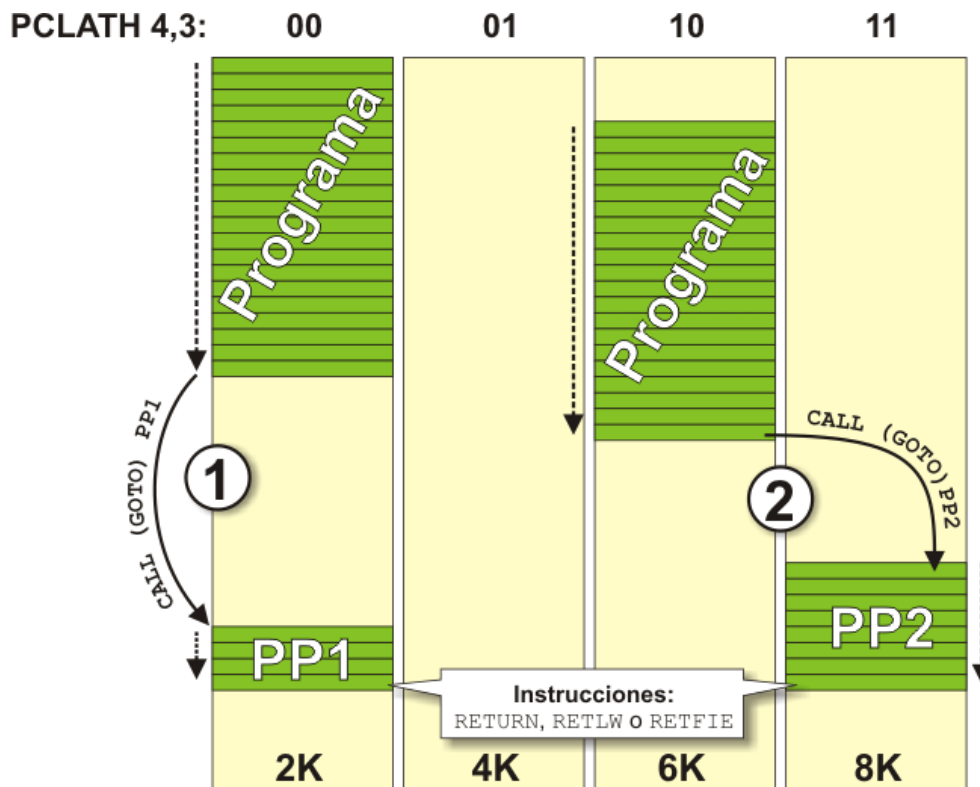


Si el contador de programa ha sido cambiado por software, debe tomar en cuenta lo siguiente para evitar problemas:

- Los ocho bits inferiores (el byte inferior) del registro PCL son de lectura/escritura, mientras que los cinco bits superiores del registro PCLATH son de sólo escritura.
- El registro PCLATH se borra con cada reinicio.
- En el lenguaje ensamblador, el valor del contador de programa está marcado con PCL y se refiere sólo a los ocho bits. Usted debe tener cuidado al utilizar la instrucción "ADDWF PCL". Esto es una instrucción de salto que especifica la localidad destino al añadir un número a la dirección actual. Se utiliza con frecuencia para saltar a la tabla de búsqueda o a la tabla de ramificación de programa y leerlas. Un problema surge si la dirección actual es de tal tamaño que al sumar se produce un cambio en un bit que pertenece al byte superior del registro PCLATH.

La ejecución de una instrucción sobre el registro PCL causa simultáneamente la sustitución de los bits del contador de programa por los contenidos en el registro PCLATH. De todos modos, el registro PCL puede acceder sólo a 8 bits inferiores del resultado de la instrucción, pues el siguiente salto será completamente incorrecto. La solución a este problema es poner estas instrucciones en las direcciones que terminan en xx00h. De esta manera se habilitan los saltos de programa hasta 255 localidades. Si se ejecutan los saltos más largos por medio de esta instrucción, el registro PCLATH debe ser incrementado por 1 cada vez que se produce desbordamiento en el registro PCL.

- Al llamar una subrutina o al ejecutarse un salto (instrucciones **CALL** y **GOTO**), el microcontrolador es capaz de proporcionar solamente direccionamiento de 11 bits. Similar a la RAM que está dividida en "bancos", la ROM está dividida en las cuatro "páginas" de 2K cada una. Las instrucciones dentro de estas páginas se ejecutan regularmente. Dicho de manera sencilla, como el procesador está proporcionado con una dirección de 11 bits del programa, es capaz de direccionar cualquier localidad dentro de 2KB. La siguiente figura muestra el salto a la dirección del subprograma PP1.



Sin embargo, si una subrutina o una dirección de salto no está en la misma página que la localidad de salto, se deben proporcionar dos bits superiores que faltan al escribir en el registro PCLATH. La siguiente figura muestra el salto a la dirección de la subrutina PP2.

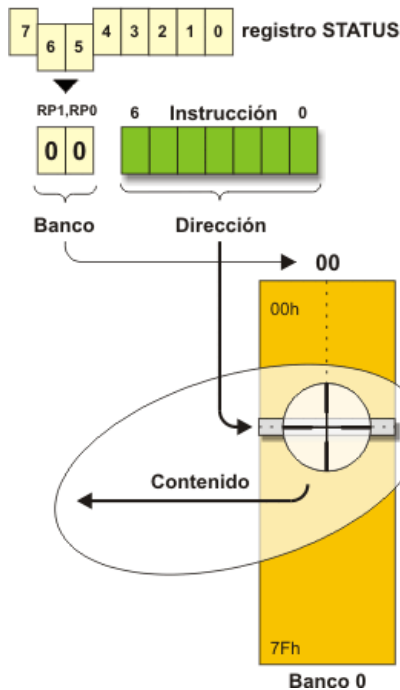
En ambos casos, cuando la subrutina llega a las instrucciones **RETURN**, **RETLW** o **RETFIE** (vuelta al programa principal), el microcontrolador continuará con la ejecución de programa desde donde se interrumpió, ya que la dirección de retorno se empuja y se guarda en la pila que consiste en registros de 13 bits, como hemos mencionado.

REGISTROS DE DIRECCIONAMIENTO INDIRECTO

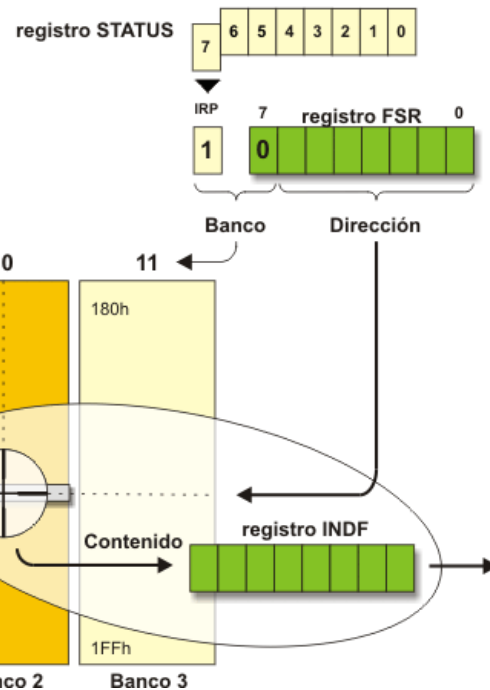
Además del direccionamiento directo, que es lógico y claro (basta con especificar la dirección de un registro para leer su contenido), este microcontrolador es capaz de realizar el direccionamiento indirecto por los registros INDF y FSR. A veces esto facilita el proceso de escribir un programa. El procedimiento entero está habilitado ya que el registro INDF no es real (no existe físicamente), sino que solamente especifica el registro cuya dirección está situada en el registro FSR. Por esta razón, escribir o leer los datos del registro INDF realmente significa escribir o leer del registro cuya dirección está situada en el registro FSR. En otras palabras, direcciones de registros se especifican en el registro FSR, y su contenido se almacena en el registro INDF. La diferencia entre el direccionamiento directo e indirecto se muestra en la siguiente figura:

Como hemos visto, el problema con "los bits de direccionamiento que faltan" se soluciona con un "préstamo" del otro registro. Esta vez, es el séptimo bit, denominado bit IRP del registro STATUS.

Direccionamiento directo



Direccionamiento indirecto



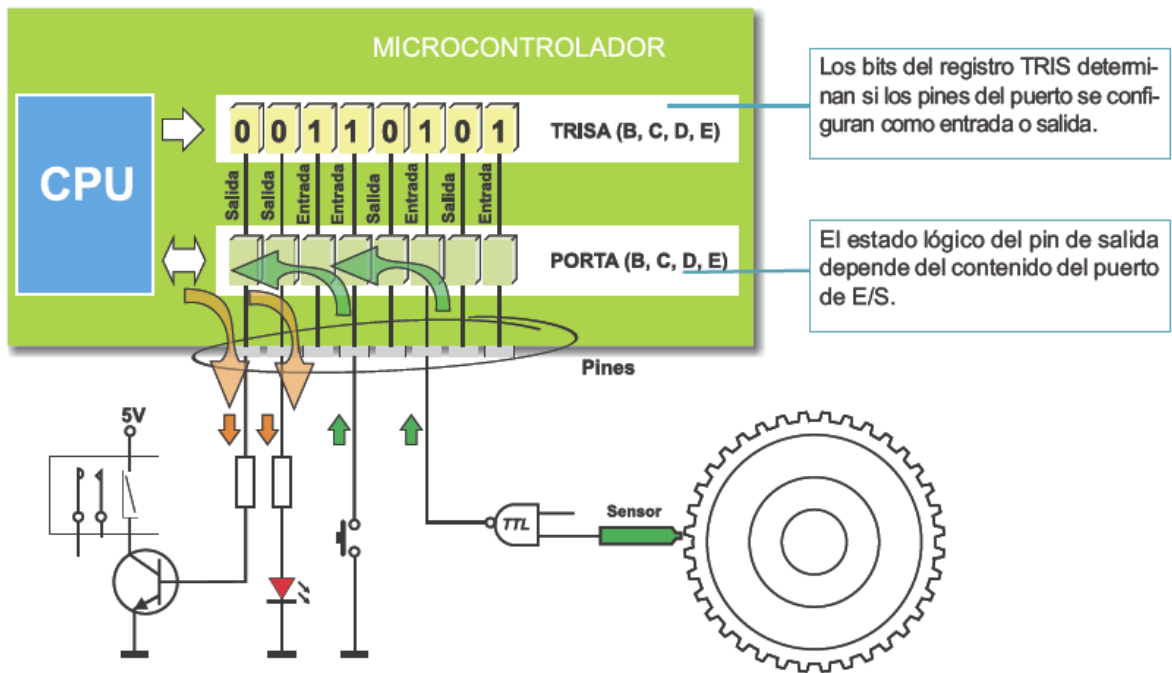
Una de las características más importantes del microcontrolador es el número de los pines de entrada/ salida, que permite conectarlo con los periféricos. El PIC16F887 tiene en total 35 pines de E/S de propósito general, lo que es suficiente para la mayoría de las aplicaciones.

PUERTOS DE ENTRADA/SALIDA

Con el propósito de sincronizar el funcionamiento de los puertos de E/S con la organización interna del microcontrolador de 8 bits, ellos se agrupan, de manera similar a los registros, en cinco puertos denotados con A, B, C, D y E. Todos ellos tienen las siguientes características en común:

- Por las razones prácticas, muchos pines de E/S son multifuncionales. Si un pin realiza una de estas funciones, puede ser utilizado como pin de E/S de propósito general.
- Cada puerto tiene su propio registro de control de flujo, o sea el registro TRIS correspondiente: TRISA, TRISB, TRISC etc. lo que determina el comportamiento de bits del puerto, pero no determina su contenido.

Al poner a cero un bit del registro TRIS (pin=0), el pin correspondiente del puerto se configurará como una salida. De manera similar, al poner a uno un bit del registro TRIS (bit=1), el pin correspondiente del puerto se configurará como una entrada. Esta regla es fácil de recordar: 0 = Entrada 1 = Salida.



Puerto PORTA y registro TRISA

El puerto PORTA es un puerto bidireccional, de 8 bits de anchura. Los bits de los registros TRISA y ANSEL controlan los pines del PORTA. Todos los pines del PORTA se comportan como entradas/salidas digitales. Cinco de ellos pueden ser entradas analógicas (denotadas por AN):

PORTA	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	Características
	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Nombre de bit

TRISA	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Características
	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Nombre de bit

Leyenda

R/W	Bit de lectura/escritura
(x)	Después del reinicio, el estado de bit es desconocido
(1)	Después del reinicio, el bit se pone a 1

RA0 = AN0 (determinado por el bit ANS0 del registro ANSEL)

RA1 = AN1 (determinado por el bit ANS1 del registro ANSEL)

RA2 = AN2 (determinado por el bit ANS2 del registro ANSEL)

RA3 = AN3 (determinado por el bit ANS3 del registro ANSEL)

RA5 = AN4 (determinado por el bit ANS4 del registro ANSEL)

Similar a que los bits del registro TRISA determinan cuáles pines serán configurados como entradas y cuáles serán configurados como salidas, los bits apropiados del registro ANSEL determinan si los pines serán configurados como entradas analógicas o entradas/salidas digitales.

Cada bit de este puerto tiene una función adicional relacionada a algunas unidades periféricas integradas, que vamos a describir en los siguientes capítulos. Este capítulo cubre sólo la función adicional del pin RA0, puesto que está relacionado al puerto PORTA y a la unidad ULPWU.

Vamos a hacerlo en mikroC...

```
// El pin PORTA.2 se configura como una entrada digital. Todos los demás pines del puerto
// PORTA son salidas digitales
```

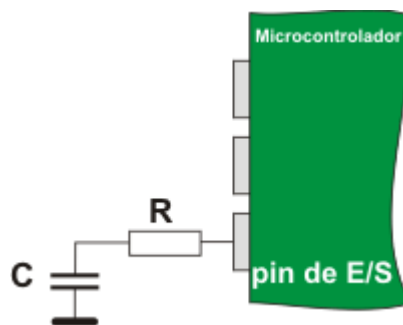
...

```
ANSEL = ANSELH = 0; // Todos los pines de E/S se configuran como digitales
PORTA = 0;           // Todos los pines del puerto PORTA se ponen a cero
TRISA = 0b00000100; // Todos los pines del puerto PORTA excepto el
                    // PORTA.2 se configuran como salidas
```

...

UNIDAD ULPWU

El microcontrolador se utiliza generalmente en los dispositivos que funcionan periódicamente y completamente independiente utilizando una fuente de alimentación de batería. En tal caso, el consumo de corriente mínimo es una de las prioridades. Los ejemplos típicos de tales aplicaciones son: termómetros, sensores de detección del fuego y similar. Es conocido que al reducir frecuencia de reloj se reduce el consumo de corriente, pues una de las soluciones más convenientes a este problema es bajar la frecuencia de reloj, o sea utilizar el cristal de cuarzo de 32KHz en vez de el de 20MHz.

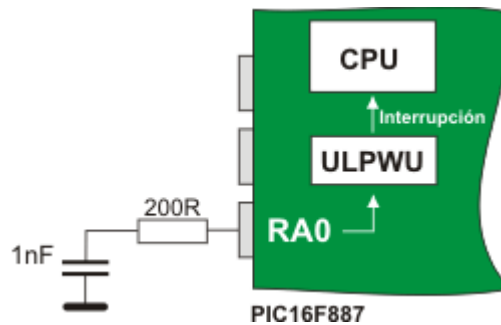


Al poner el microcontrolador en el modo de reposo es otro paso en la misma dirección. Aún ha quedado el problema de salir de este modo y poner el microcontrolador en modo normal de funcionamiento. Es obviamente necesario tener una señal externa en alguno de los pines. Esta señal debe ser generada por componentes electrónicos adicionales, lo que resulta en un consumo de energía más alto del dispositivo completo...

La solución perfecta sería que el microcontrolador saliera del modo de reposo periódicamente por sí mismo, lo que no es imposible. El circuito que lo habilita se muestra en la figura a la izquierda.

El principio de funcionamiento es simple:

Un pin se configura como salida y se le lleva un uno lógico (1). Esto causa una carga del capacitor. Inmediatamente después, el mismo pin se configura como entrada. El cambio de estado lógico habilita una interrupción y el microcontrolador entra en modo de reposo. Sólo ha quedado esperar que se descargue el capacitor por la corriente de fuga fluyendo por el pin de entrada. Después de la descarga, se produce una interrupción y el microcontrolador continúa con la ejecución de programa en modo normal. Todo el procedimiento se repite.



En teoría, esto es una solución perfecta. El problema es que todos los pines capaces de causar una interrupción son digitales y tienen una corriente de fuga relativamente alta cuando el voltaje sobre ellos no está cerca de los valores límites de Vdd (1) o VSS (0). En este caso, el condensador se descarga en poco tiempo ya que la corriente es de varias centenas de microamperios. Por esta razón se diseñó el circuito ULPWU, capaz de indicar una lenta caída de voltaje con un consumo de corriente mínimo.

La salida genera una interrupción, mientras que la entrada está conectada a uno de los pines del microcontrolador. Es el pin RA0. Refiriéndose a la Figura (R=200 ohms, C=1nF), el tiempo de descarga es aproximadamente 30mS, mientras que un consumo total de corriente del microcontrolador es 1000 veces más bajo (de varias centenas de nanoamperios).

Puerto PORTB y registro TRISB

El puerto PORTB es un puerto bidireccional, de 8 bits de anchura. Los bits del registro TRISB determinan la función de sus pines.

PORTB	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	Características
	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Nombre de bit

TRISB	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Características
	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Nombre de bit

Leyenda

- Bit no implementado
- R/W Bit de lectura/escritura
- (x) Después del reinicio, el estado de bit es desconocido
- (1) Después del reinicio, el bit está a uno

Similar al puerto PORTA, un uno lógico (1) en el registro TRISB configura el pin apropiado en el puerto PORTB y al revés. Los seis pines de este puerto se pueden comportar como las entradas analógicas (AN). Los bits del registro ANSELH determinan si estos pines serán configurados como entradas analógicas o entradas/salidas digitales:

RB0 = AN12 (determinado por el bit ANS12 del registro ANSELH)

RB1 = AN10 (determinado por el bit ANS10 del registro ANSELH)

RB2 = AN8 (determinado por el bit ANS8 del registro ANSELH)

RB3 = AN9 (determinado por el bit ANS9 del registro ANSELH)

RB4 = AN11 (determinado por el bit ANS11 del registro ANSELH)

RB4 = AN11 (determinado por el bit ANS11 del registro ANSELH)

Cada bit de este puerto tiene una función adicional relacionada a algunas unidades periféricas integradas, que vamos a describir en los siguientes capítulos.

Este puerto dispone de varias características por las que se distingue de otros puertos y por las que sus pines se utilizan con frecuencia:

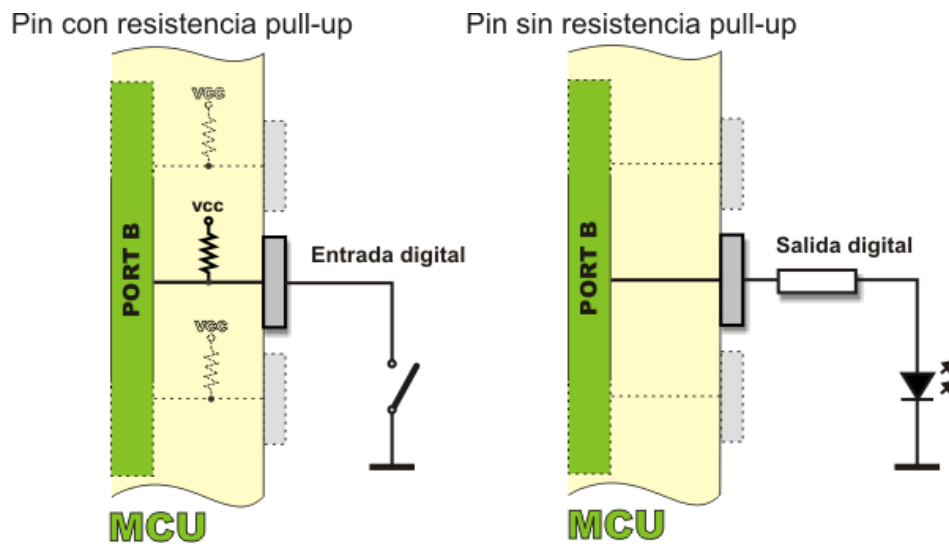
- Todos los pines del puerto PORTB tienen las resistencias *pull-up* integradas, que los hacen perfectos para que se conecten con los botones de presión (con el teclado), interruptores y optoacopladores. Con el propósito de conectar las resistencias a los puertos del microcontrolador, el bit apropiado del registro WPUB debe estar a uno.*

WPUB	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Características
	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Leyenda

R/W	Bit de lectura/escritura
(1)	Después del reinicio, el bit se pone a 1

Al tener un alto nivel de resistencia (varias decenas de kiloohmios), estas resistencias "virtuales" no afectan a los pines configurados como salidas, sino que sirven de un complemento útil a las entradas. Estas resistencias están conectados a las entradas de los circuitos lógicos CMOS. De lo contrario, se comportarían como si fueran flotantes gracias a su alta resistencia de entrada.



Además de los bits del registro WPUB, hay otro bit que afecta a la instalación de las resistencias pull-up. Es el bit RBPU del registro OPTION_REG.

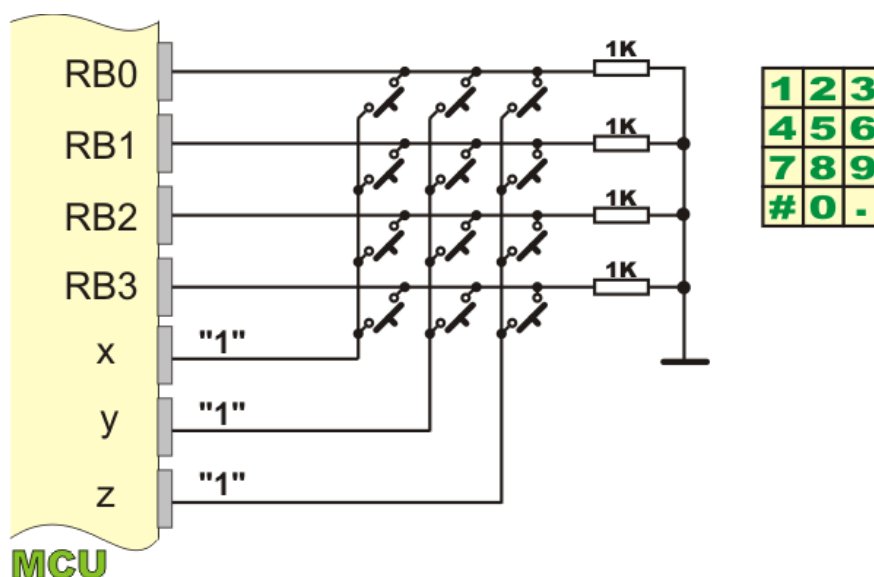
- Al estar habilitado, cada bit del puerto PORTB configurado como una entrada puede causar una interrupción al cambiar su estado lógico. Con el propósito de habilitar que los terminales causen una interrupción, el bit apropiado del registro IOCB debe estar a uno.

IOCB	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Nombre de bit

Leyenda

R/W	Bit de lectura/escritura
(0)	Después del reinicio, el bit se pone a cero

Gracias a estas características, los pines del puerto PORTB se utilizan con frecuencia para comprobar los botones de presión en el teclado ya que detectan cada apretón de botón infaliblemente. Por eso, no es necesario examinar todas las entradas una y otra vez.



Cuando los pines X,Y y Z se configuran como entradas de puesta a uno (1), sólo se necesita esperar una petición de interrupción que aparece al apretar un botón. Más tarde, se comprueba cuál botón fue activado al combinar ceros y unos en las entradas.

Vamos a hacerlo en mikroC...

/ El pin PORTB.1 se configura como entrada digital. Se produce una interrupción con cualquier cambio de su estado lógico. También tiene una resistencia pull-up. Todos los demás pines del puerto PORTB son entradas digitales. */*

```
...
ANSEL = ANSELH = 0; // Todos los pines de E/S se configuran como digitales
PORTB = 0;          // Todos los pines del puerto PORTB se ponen a cero
TRISB = 0b00000010; // Todos los pines del puerto PORTB excepto PORTB.1
                    // se configuran como salidas
RBPU = 0;           // Se habilitan las resistencias pull-up
WPUB1 = 1;         // La resistencia pull-up se conecta al pin PORTB.1
IOCB1 = 1;         // El pin PORTB.1 puede causar una interrupción por el
                    // cambio del estado lógico
RBIE = GIE = 1;    // Se habilita una interrupción
...
```

PIN RB0/INT

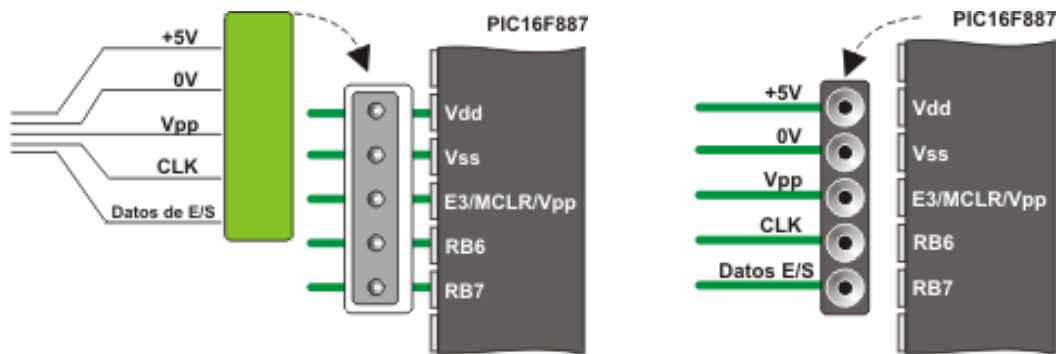
El pin RB0/INT es la única fuente “verdadera” de la interrupción externa. Se puede configurar de manera que reaccione al borde ascendente de señal (transición de cero a uno) o al borde descendente de señal (transición de uno a cero). El bit INTEDG del registro OPTION_REG selecciona la señal apropiada.

PINES RB6 Y RB7

El PIC16F887 no dispone de ningún pin especial para la programación (el proceso de escribir un programa en la ROM). Los pines que generalmente están disponibles como los pines de E/S de propósito general, se utilizan para este propósito. Para decir con más precisión, son los pines del puerto PORTB utilizados para la transmisión de señal de reloj (RB6) y de datos (RB7) al cargar el programa. Además, es necesario suministrar el voltaje de alimentación Vdd (5V) así como el voltaje apropiado Vpp (12-14V) para la programación de memoria FLASH. Durante la programación, el voltaje Vpp se aplica al pin MCLR. No se preocupe de los detalles relacionados a este proceso, tampoco se preocupe de cuál voltaje se aplica primero puesto que los componentes del programador se encargan de eso. Lo que es muy importante es que el programa se puede cargar al microcontrolador aún después de haber sido soldado en el dispositivo destino. Por supuesto, el programa cargado se puede cambiar de la misma manera. Esta función se le denomina ICSP (In-Circuit Serial Programming - Programación serial en circuito)

Para utilizarlo correctamente es necesario planificar con antelación. ¡Es pan comido! Sólo es necesario instalar un conector miniatura de 5 pines en el dispositivo destino para suministrar al microcontrolador un voltaje de programación necesario.

Para evitar la interferencia entre los voltajes y los componentes del dispositivo conectados a los pines del microcontrolador, todos los periféricos adicionales deben estar desconectados durante la programación (utilizando las resistencias o los puentes).



Como hemos visto, los voltajes aplicados a los pines del zócalo del programador son los mismos que los utilizados durante la programación ICSP.

Puerto PORTC y registro TRISC

El puerto PORTC es un puerto bidireccional, de 8 bits de anchura. Los bits del registro TRISC determinan la función de sus pines. Similar a otros puertos, un uno lógico (1) en el registro TRISC configura el pin apropiado del puerto PORTC como entrada.

PORTC	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	Prestaciones
	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Nombre de bit

TRISC	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Características
	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Nombre de bit

Leyenda

R/W	Bit de lectura/escritura
(x)	Después del reinicio, el estado de bit es desconocido
(1)	Después del reinicio, el bit se pone a uno

Todas las funciones adicionales del puerto PORTC se describen en los siguientes capítulos.

Puerto PORTD y registro TRISD

El puerto PORTD es un puerto bidireccional de 8 bits de anchura. Los bits del registro TRISD determinan la función de sus pines. Similar a otros puertos, un uno lógico (1) en el registro TRISD configura el pin apropiado del puerto PORTD como entrada.

	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	R/W (x)	Características
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Características
TRISD	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

R/W	Bit de lectura/escritura
(x)	Después del reinicio, el estado de bit es desconocido
(1)	Después del reinicio, el bit se pone a uno

Puerto PORTE y registro TRISE

El puerto PORTE es un puerto bidireccional, de 4 bits de anchura. Los bits del registro TRISE determinan la función de sus pines. Similar a otros puertos, un uno lógico (1) en el registro TRISE configura el pin apropiado del puerto PORTE como entrada.

					R/W (x)	R/W (x)	R/W (x)	R/W (x)	Características
PORTE	-	-	-	-	RE3	RE2	RE1	RE0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

					R (1)	R/W (1)	R/W (1)	R/W (1)	Características
TRISE	-	-	-	-	TRISE3	TRISE2	TRISE1	TRISE0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
R	Bit de lectura
(x)	Después del reinicio, el estado de bit es desconocido
(1)	Después del reinicio, el bit se pone a uno

La excepción es el pin RE3, que siempre está configurado como entrada.

Similar a los puertos PORTA y PORTB, en este caso los tres pines se pueden configurar como entradas analógicas. Los bits del registro ANSEL determinan si estos pines serán configurados como entradas analógicas (AN) o entradas/salidas digitales:

RE0 = AN5 (determinado por el bit ANS5 del registro ANSEL);

RE1 = AN6 (determinado por el bit ANS6 del registro ANSEL); y

RE2 = AN7 (determinado por el bit ANS7 del registro ANSEL).

Vamos a hacerlo en mikroC...

/ El pin PORTE.0 se configura como una entrada analógica mientras que los demás tres pines del mismo puerto se configuran como digitales */*

```

...
ANSEL = 0b00100000; // El pin PORTE.0 se configura como analógico
ANSELH = 0; // Todos los pines de E/S se configuran como digitales
TRISE = 0b00000001; // Todos los pines del puerto PORTE excepto el
// PORTE.0 se configuran como salidas
PORTE = 0; // Todos los bits del puerto PORTE se ponen a cero
...
    
```

Registros ANSEL y ANSELH

Los registros ANSEL y ANSELH se utilizan para configurar el modo de entrada de un pin de E/S como analógico o como digital.

	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Características
ANSEL	ANS7	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

			R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Características
ANSELH	-	-	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

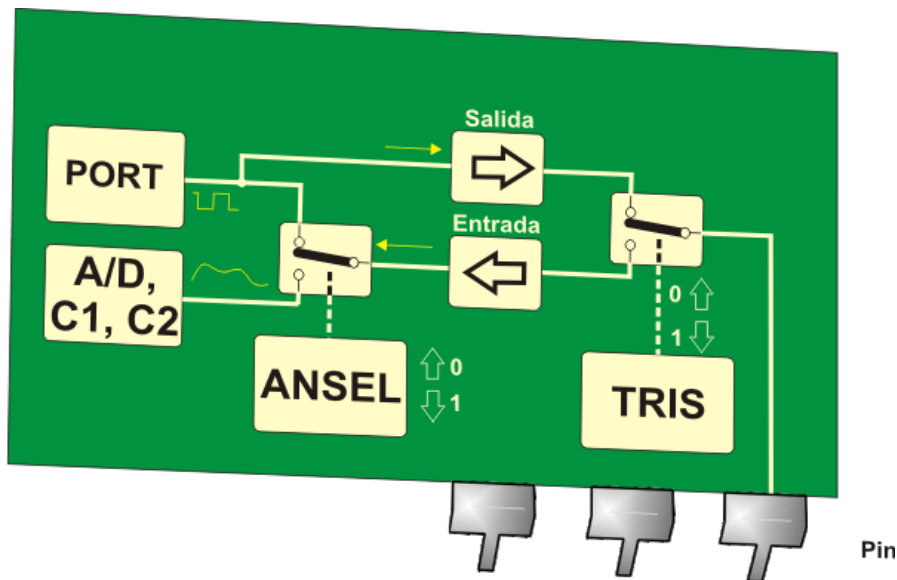
Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
(1)	Después del reinicio, el bit se pone a uno

La regla es la siguiente:

Para configurar un pin como una entrada analógica, el bit apropiado de los registros ANSEL o ANSELH se debe poner a uno (1). Para configurar un pin como una entrada/salida digital, el bit apropiado se debe poner a cero (0).

El estado lógico de los bits del registro ANSEL no tiene influencia en las funciones de salidas digitales. Al intentar a leer un pin de puerto configurado como una entrada analógica, el resultado es siempre 0.



Es probable que usted nunca vaya a escribir un programa que no utilice puertos, así que el esfuerzo para aprender todo sobre ellos en definitiva vale la pena. De todos modos, los puertos son probablemente los módulos más simples dentro del microcontrolador. Se utilizan de la siguiente manera:

- Al diseñar un dispositivo, seleccione un puerto por el que el microcontrolador comunicará al entorno periférico. Si usted utiliza sólo entradas/salidas digitales, seleccione cualquier puerto. Si utiliza alguna de las entradas analógicas, seleccione los puertos apropiados que soportan tal configuración de los pines (AN0-AN13).
- Cada pin del puerto se puede configurar como salida o como entrada. Los bits de los registros TRISA, TRISB, TRISC, TRISD y TRISE determinan cómo se comportarán los pines apropiados de los puertos PORTA, PORTB, PORTC, PORTD y PORTE. Simplemente...
- Si utiliza alguna de las entradas analógicas, primero es necesario poner a uno los bits apropiados de los registros ANSEL y ANSELH en el principio de programa.
- Si utiliza resistencias o botones de presión como una fuente de señal de entrada, conéctelos a los pines del puerto PORTB, ya que tienen las resistencias pull-up. El uso de estos registros está habilitado por el bit RBPU del registro OPTION_REG, mientras que la instalación de las resistencias individuales está habilitada por los bits del registro WPUB.
- Con frecuencia se necesita responder tan pronto como los pines de entrada cambien su estado lógico. Sin embargo, no es necesario escribir un programa para comprobar el estado lógico de los pines. Es mucho más simple conectar estas entradas a los pines del puerto PORTB y habilitar que ocurra una interrupción con cada cambio de voltaje. Los bits de los registros IOCB e INTCON se encargan de eso.

El microcontrolador PIC16F887 dispone de tres temporizadores/contadores independientes, denominados Timer0, Timer1 y Timer2. En este capítulo se presenta una descripción detallada de los mismos.

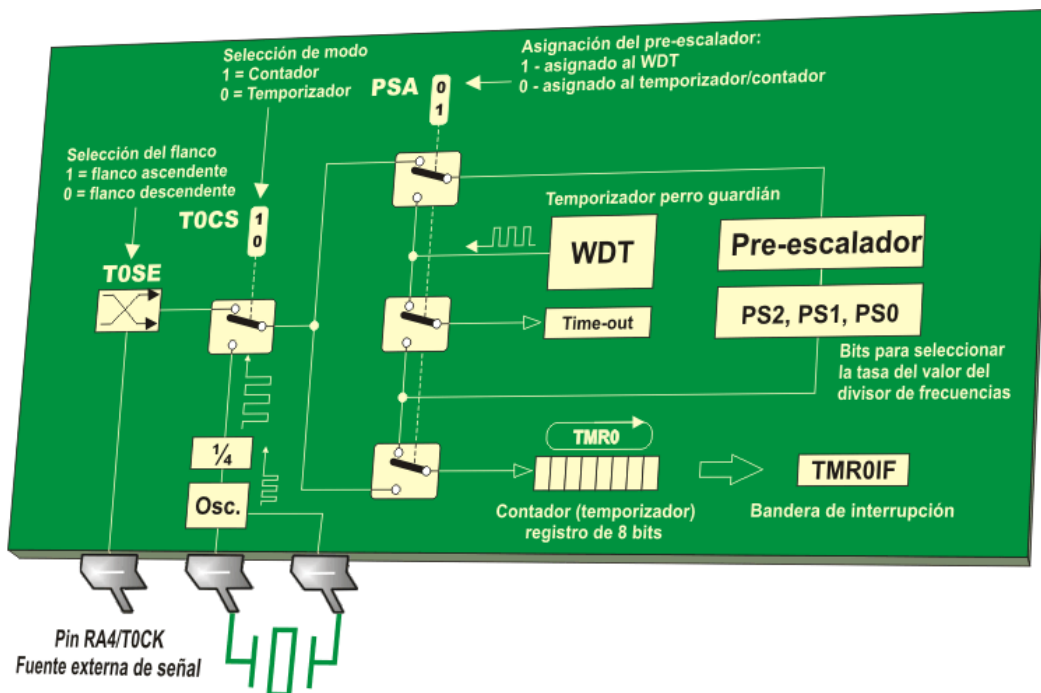
TEMPORIZADOR TIMER0

El temporizador Timer0 tiene una amplia gama de aplicaciones en la práctica. Sólo unos pocos programas no lo utilizan de alguna forma. Es muy conveniente y fácil de utilizar en programas o subrutinas para generar pulsos de duración arbitraria, en medir tiempo o en contar los pulsos externos (eventos) casi sin limitaciones.

El módulo del temporizador Timer0 es un temporizador/contador de 8 bits con las siguientes características:

- Temporizador/contador de 8 bits;
- Pre-escalador de 8 bits (lo comparte con el temporizador perro guardián);
- Fuente de reloj interna o externa programable;
- Generación de interrupción por desbordamiento; y
- Selección del flanco de reloj externo programable.

La siguiente figura muestra el esquema del temporizador Timer0 con todos los bits que determinan su funcionamiento. Estos bits se almacenan en el registro OPTION_REG.



Registro OPTION_REG

OPTION_REG	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	Características
	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

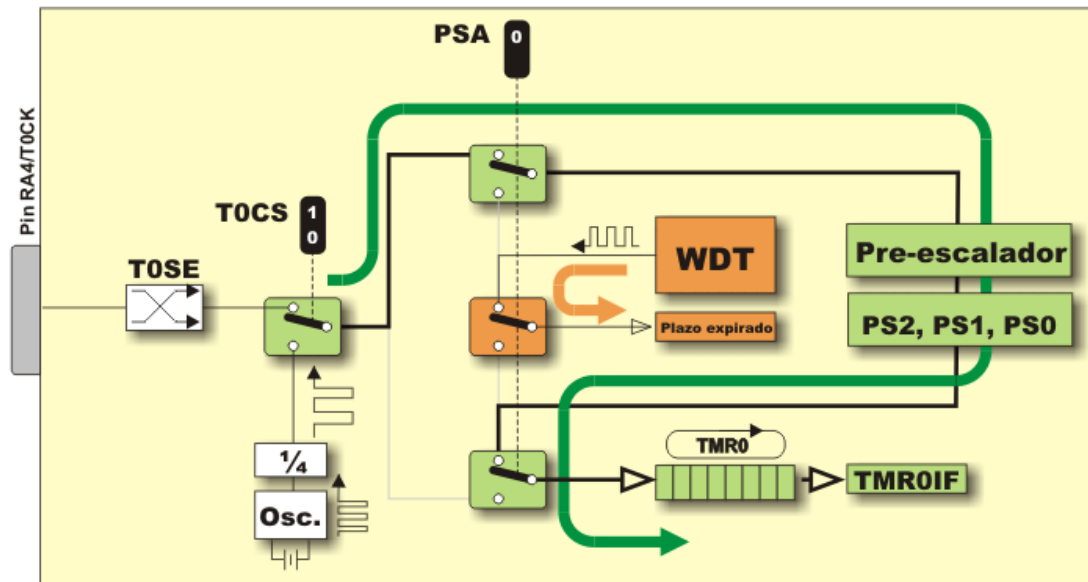
Leyenda

R/W Bit de lectura/escritura
(1) Después del reinicio, el bit se pone a uno

- **RBPB - PORTB Pull-up enable bit (resistencia Pull Up del puerto PORTB)**
 - 0 - Resistencias pull-up del puerto PORTB están deshabilitadas.
 - 1 - Pines del puerto PORTB pueden estar conectados a las resistencias pull-up.
- **INTEDG - Interrupt Edge Select bit (bit selector de flanco activo de la interrupción externa)**
 - 0 - Interrupción por flanco ascendente en el pin INT (0-1).
 - 1 - Interrupción por flanco descendente en el pin INT (1-0).
- **T0CS - TMR0 Clock Select bit (bit selector de tipo de reloj para el Timer0)**
 - 0 - Los pulsos se llevan a la entrada del temporizador/contador Timer0 por el pin RA4.
 - 1 - El temporizador utiliza los pulsos de reloj internos ($F_{osc}/4$).
- **T0SE - TMR0 Source Edge Select bit (bit selector de tipo de flanco)**
 - 0 - Incrementa en flanco descendente en el pin TMR0.
 - 1 - Incrementa en flanco ascendente en el pin TMR0.
- **PSA - Prescaler Assignment bit (bit de asignación del pre-escalador)**
 - 0 - Pre-escalador se le asigna al WDT.
 - 1 - Pre-escalador se le asigna al temporizador/contador Timer0.
- **PS2, PS1, PS0 - Prescaler Rate Select bit (bit selector del valor del divisor de frecuencias)**
 - El valor del divisor de frecuencias se ajusta al combinar estos bits. Como se muestra en la tabla a la derecha, la misma combinación de bits proporciona los diferentes valores del divisor de frecuencias para el temporizador/contador y el temporizador perro guardián, respectivamente.

PS2	PS1	PS0	TMR0	WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Cuando el bit PSA está a 0, el pre-escalador se le asigna al temporizador/contador Timer0, como se muestra en la siguiente figura.



Vamos a hacerlo en mikroC...

// En este ejemplo, Timer0 se configura como un temporizador y se le asigna un pre-escalador.

```
unsigned cnt;    // Declarar la variable cnt
```

```
void interrupt() { // Rutina de interrupción
    cnt++;          // Interrupción causa el incremento de cnt por 1
    TMR0 = 155;    // Temporizador (o contador) Timer0 devuelve su valor inicial
    INTCON = 0x20; // Bit T0IE está a 1, bit T0IF está a 0
}
```

```
void main() {
    OPTION_REG = 0x04; // Pre-escalador (1:32) se le asigna al temporizador Timer0
    TMR0 = 155;       // Temporizador T0 cuenta de 155 a 255
    INTCON = 0xA0;   // Habilitada la generación de interrupción para el
                    // temporizador Timer0
```

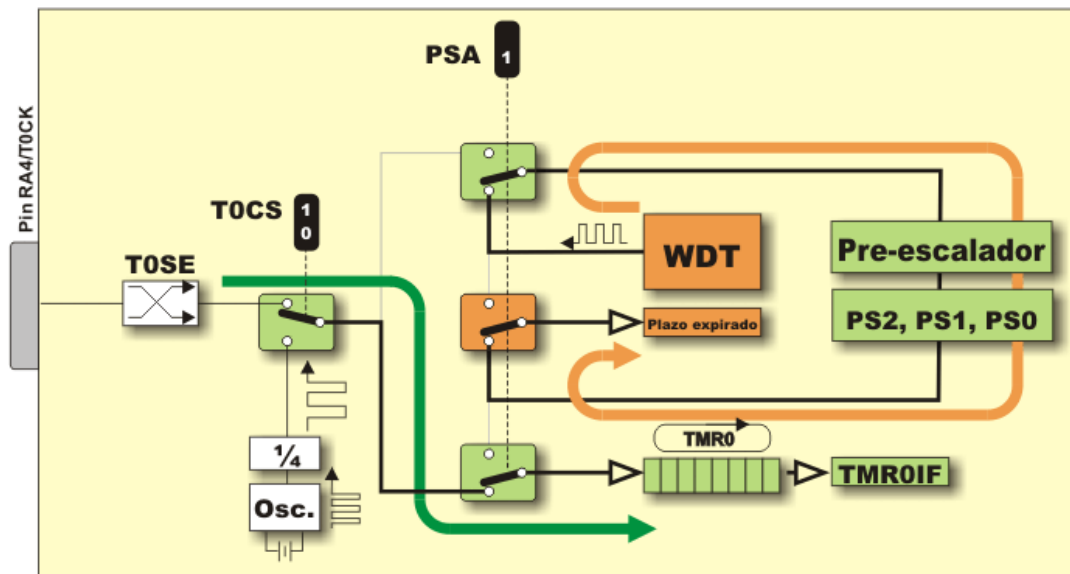
...

*// En el siguiente ejemplo, Timer0 se configura como un temporizador
// y se le asigna un pre-escalador.*

```
OPTION_REG = 0x20; // Pre-escalador (1:2) se le asigna al contador Timer0
TMR0 = 155;       // Contador T0 cuenta de 155 a 255
INTCON = 0xA0;   // Habilitada la generación de interrupción por el
                // temporizador Timer0
```

...

Cuando el bit PSA está a 1, el pre-escalador se le asigna al temporizador perro guardián como se muestra en la siguiente figura.



Vamos a hacerlo en mikroC...

// En este ejemplo, el pre-escalador (1:64) se le asigna al temporizador perro guardián.

```
void main() {
    OPTION_REG = 0x0E; // Pre-escalador se le asigna al WDT (1:64)
    asm CLRWDT;      // Comando en ensamblador para reiniciar el WDT
    ...
    ...
    asm CLRWDT;      // Comando en ensamblador para reiniciar el WDT
    ...
}
```

Aparte de lo dicho anteriormente, cabe destacar lo siguiente:

- Al asignarle el pre-escalador al temporizador/contador, el pre-escalador se pondrá a 0 con cualquier escritura en el registro TMR0.
- Al asignar el pre-escalador al temporizador perro guardián, tanto el WDT como el preescalador se pondrán a 0 con la instrucción CLRWDT.
- Al escribir en el registro TMR0, utilizado como un temporizador, no se inicia el conteo de los pulsos inmediatamente, sino con retraso de dos ciclos de instrucciones. Por consiguiente, es necesario ajustar el valor escrito en el registro TMR0.
- Al poner el microcontrolador en el modo de reposo se apaga el oscilador de reloj. No puede ocurrir el desbordamiento ya que no hay pulsos a contar. Es la razón por la que la interrupción por el desbordamiento del TMR0 no puede "despertar" al procesador del modo de reposo.
- Si se utiliza como un contador de reloj externo sin pre-escalador, la longitud de pulso mínima o tiempo muerto entre dos pulsos deberá ser $2 T_{osc} + 20 \text{ nS}$ (T_{osc} es el período de señal de reloj del oscilador).
- Si se utiliza como un contador de reloj externo con pre-escalador, la longitud de pulso mínima o tiempo muerto entre dos pulsos es sólo 10nS.
- El registro del pre-escalador de 8 bits no está disponible al usuario, lo que significa que no es posible leerlo o escribir en él directamente.
- Al cambiar de asignación del pre-escalador del Timer0 al temporizador perro guardián, es necesario ejecutar la siguiente secuencia de instrucciones escritas en ensamblador para impedir reiniciar el microcontrolador:
- BANKSEL TMR0

- CLRWDT ;PONER A CERO WDT
 - CLRF TMR0 ;PONER A CERO TMR0 Y PRE-ESCALADOR
 - BANKSEL OPTION_REG
 - BSF OPTION_REG,PSA ;ASIGNARLE EL PRE-ESCALADOR AL WDT
 - CLRWDT ;PONER A CERO WDT
 - MOVLW b'11111000' ;SELECCIONAR LOS BITS PS2,PS1,PS0 Y PONERLOS
 - ANDWF OPTION_REG,W ;A CERO POR LA INSTRUCCIÓN 'Y LÓGICO'
 - IORLW b'00000101' ;BITS PS2, PS1, Y PS0 PONEN EL VALOR
 - MOVWF OPTION_REG ;DEL DIVISOR DE FRECUENCIAS A 1:32
- De manera similar, al cambiar de asignación del pre-escalador del WDT al Timer0, es necesario ejecutar la siguiente secuencia de instrucciones, también escritas en ensamblador:
- BANKSEL TMR0
 - CLRWDT ;PONER A CERO WDT Y PRE-ESCALADOR
 - BANKSEL OPTION_REG
 - MOVLW b'11110000' ;SELECCIONAR SÓLO LOS BITS PSA,PS2,PS1,PS0
 - ANDWF OPTION_REG,W ;Y PONERLOS A CERO POR LA INSTRUCCIÓN 'Y LÓGICO'
 - IORLW b'00000011' ;VALOR DEL DIVISOR DE FRECUENCIAS ES 1:16
 - MOVWF OPTION_REG

Para utilizar el Timer0 apropiadamente, es necesario:

Paso 1: Seleccionar el modo:

- El modo de temporizador se selecciona por el bit TOSC del registro OPTION_REG (TOSC: 0=temporizador, 1=contador).
- Cuando se asigna el pre-escalador al temporizador/contador se debe poner a cero el bit PSA del registro OPTION_REG. El valor del divisor de frecuencias se configura al utilizar los bits PS2-PS0 del mismo registro.
- Al utilizar una interrupción, los bits GIE y TMR0IE del registro INTCON deben estar a uno.

Paso 2: Medir y contar

Para medir tiempo:

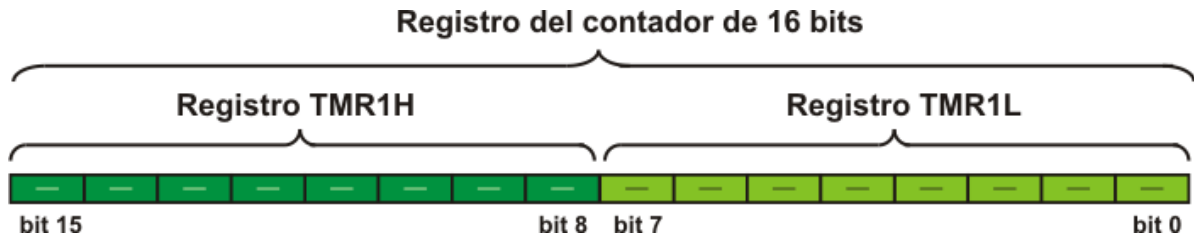
- Reiniciar el registro TMR0 o escribir un valor conocido en él.
- El tiempo transcurrido(en microsegundos al utilizar el oscilador de 4MHz) se mide al leer el registro TMR0.
- El bit de bandera TMR0IF del registro INTCON se pone a uno automáticamente siempre que ocurra el desbordamiento del registro TMR0. Si está habilitada, ocurre una interrupción.

Para contar pulsos:

- La polaridad de pulsos a contar en el pin RA4 se selecciona por el bit TOSE del registro OPTION_REG (TOSE: 0=pulsos positivos, 1=pulsos negativos).
- Varios pulsos se pueden leer del registro TMR0. El pre-escalador y la interrupción se utilizan de la misma forma que en el modo de temporizador.

TEMPORIZADOR TIMER1

El módulo del temporizador Timer1 es un temporizador/contador de 16 bits, lo que significa que consiste en dos registros (TMR1L y TMR1H). Puede contar hasta 65535 pulsos en un solo ciclo, o sea, antes de que el conteo se inicie desde cero.

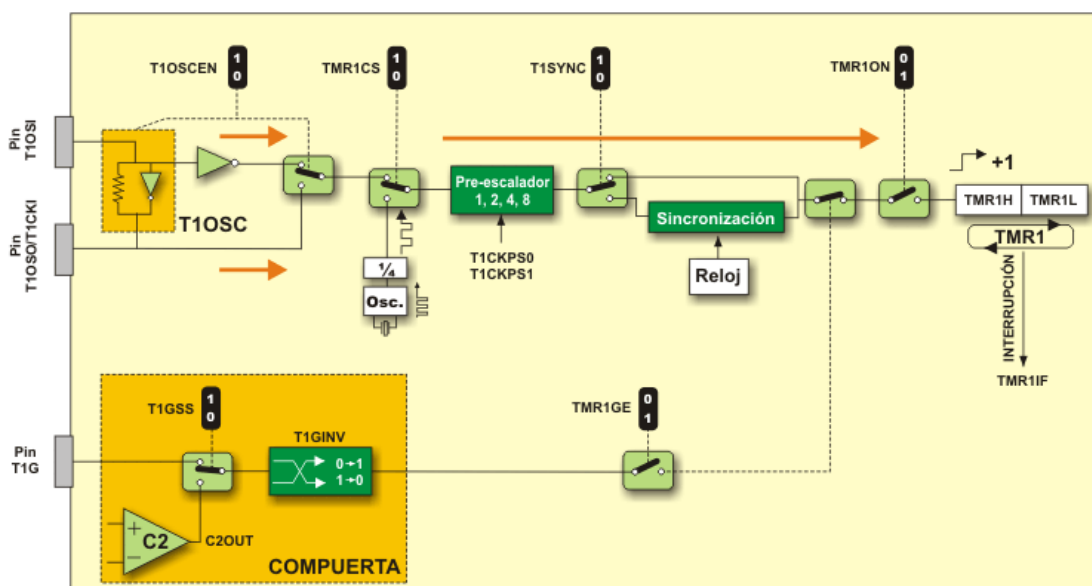


Similar al temporizador Timer0, estos registros se pueden leer o se puede escribir en ellos en cualquier momento. En caso de que ocurra un desbordamiento, se genera una interrupción si está habilitada.

El módulo del temporizador Timer1 puede funcionar en uno o dos modos básicos, eso es como un temporizador o como un contador. A diferencia del temporizador Timer0, cada uno de estos dos modos tiene funciones adicionales.

El temporizador Timer1 tiene las siguientes características:

- Temporizador/contador de 16 bits compuesto por un par de registros;
- Fuente de reloj interna o externa programable;
- Pre-escalador de 3 bits;
- Oscilador LP opcional;
- Funcionamiento síncrono o asíncrono;
- Compuerta para controlar el temporizador Timer1 (conteo habilitado) por medio del comparador o por el pin T1G;
- Interrupción por desbordamiento;
- "Despierta" al microcontrolador (salida del modo de reposo) por desbordamiento (reloj externo); y
- Fuente de reloj para la función de Captura/Comparación.



SELECCIÓN DE LA FUENTE DE RELOJ DEL TEMPORIZADOR TIMER1

El bit TMR1CS del registro T1CON se utiliza para seleccionar la fuente de reloj para este temporizador:

Fuente de reloj TMR1CS

Fosc/4	0
T1CKI pin	1

Al seleccionar la fuente de reloj interna, el par de registros TMR1H-TMR1L será incrementado con varios pulsos Fosc como es determinado por el pre-escalador.

Al seleccionar la fuente de reloj externa, este temporizador puede funcionar como un temporizador o un contador. Los pulsos en el modo temporizador pueden estar sincronizados con el reloj interno del microcontrolador o funcionar asíncronamente. En caso de que se necesite un oscilador del reloj externo y el microcontrolador PIC16F887 utilice el oscilador interno INTOSC con el pin RA6/OSC2/CLKOUT, el temporizador Timer1 puede utilizar el oscilador LP como una fuente de reloj.

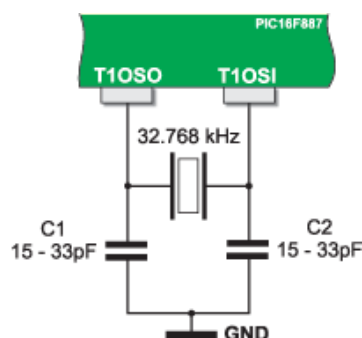
PRE-ESCALADOR DEL TEMPORIZADOR TIMER1

El temporizador Timer1 tiene un escalador completamente separado que permite dividir la frecuencia de entrada de reloj por 1,2,4 o 8. No es posible leer el pre-escalador o escribir en él directamente. De todas formas, el contador del pre-escalador se pone a 0 automáticamente después de escribir en los registros TMR1H o TMR1L.

OSCILADOR DEL TEMPORIZADOR TIMER1

Los pines RC0/T1OSO y RC1/T1OSI se utilizan para registrar los pulsos que vienen de los dispositivos periféricos, pero también tienen una función adicional. Como se puede ver en la siguiente figura, se configuran simultáneamente como entrada (pin RC1) y salida (pin RC0) del oscilador de cuarzo LP (Low Power - de bajo consumo) adicional. Este circuito está principalmente diseñado para funcionar a bajas frecuencias (hasta 200 KHz), exactamente para el uso de cristal de cuarzo de 32.768 KHz. Este cristal se utiliza en los relojes de cristal puesto que es fácil de obtener un pulso de duración de un segundo al dividir esta frecuencia.

Como el oscilador no depende del reloj interno, puede funcionar incluso en el modo de reposo. Se habilita al poner a uno el bit de control T1OSCEN del registro T1CON. El usuario debe proporcionar tiempo muerto por medio de software (unos pocos milisegundos) para habilitar que el oscilador se inicie apropiadamente.

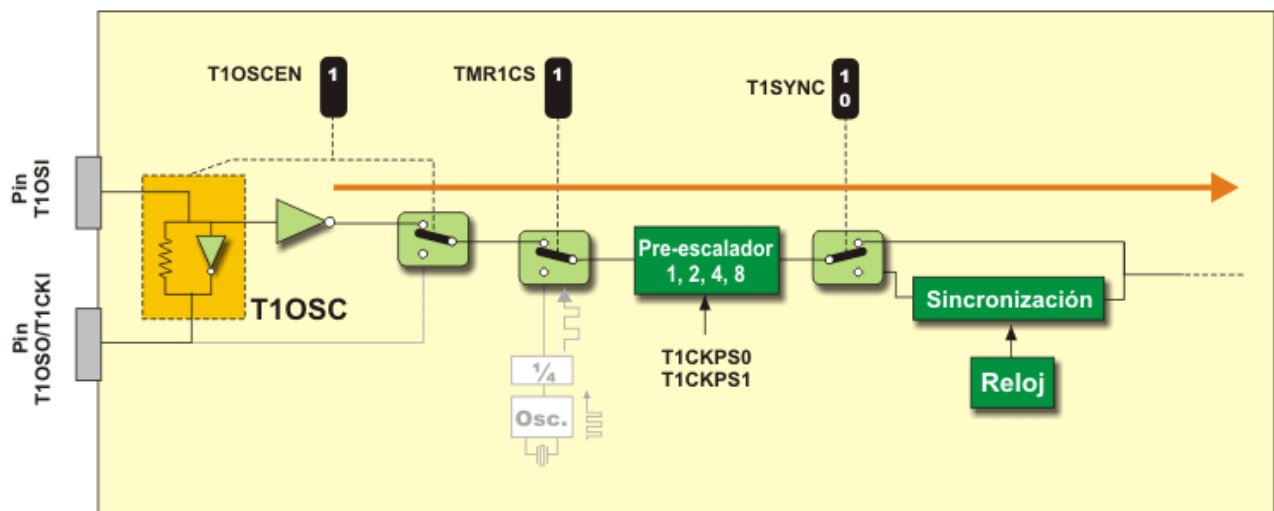


La siguiente tabla muestra los valores recomendados de los capacitores convenientes con el oscilador de cuarzo. No es necesario que estos valores sean exactos. De todas formas, la regla general es: cuánto más alta sea la capacidad, tanto más alta será la estabilidad, lo que a la vez prolonga el tiempo necesario para la estabilización del oscilador.

Oscilador	Frecuencia	C1	C2
LP	32 kHz	33 pF	33 pF
	100 kHz	15 pF	15 pF
	200 kHz	15 pF	15 pF

El consumo de corriente del microcontrolador se reduce a nivel más bajo en el modo de reposo ya que el consumidor de corriente principal - el oscilador - no funciona. Es fácil de poner al microcontrolador en este modo - al ejecutar la instrucción SLEEP. El problema es cómo despertar al microcontrolador porque sólo una interrupción puede producirlo. Como el microcontrolador “duerme”, se debe usar una interrupción causada por dispositivos periféricos para “despertarlo”. Se pone muy complicado si es necesario despertar al microcontrolador a intervalos de tiempo regulares...

Para resolver el problema, un oscilador de cuarzo LP (de bajo consumo de corriente) completamente independiente, capaz de funcionar en el modo de reposo, está integrado en el microcontrolador PIC16F887. Simplemente, un circuito anteriormente separado ahora está integrado en el microcontrolador y asignado al temporizador Timer1. El oscilador está habilitado al poner a 1 el bit T1OSCEN del registro T1CON. El bit TMR1CS del mismo registro se utiliza para habilitar que el temporizador Timer1 utilice secuencias de pulsos de ese oscilador.



- Una señal generada por este oscilador de cuarzo está sincronizada con el reloj del microcontrolador al poner a 0 el bit T1SYNC. En este caso, el temporizador no puede funcionar en modo de reposo porque el circuito para sincronización utiliza el reloj del microcontrolador.
- La interrupción por desbordamiento en el registro del temporizador Timer1 puede estar habilitada. Si el bit T1SYNC se pone a 1, tales interrupciones se producirán en el modo de reposo también.

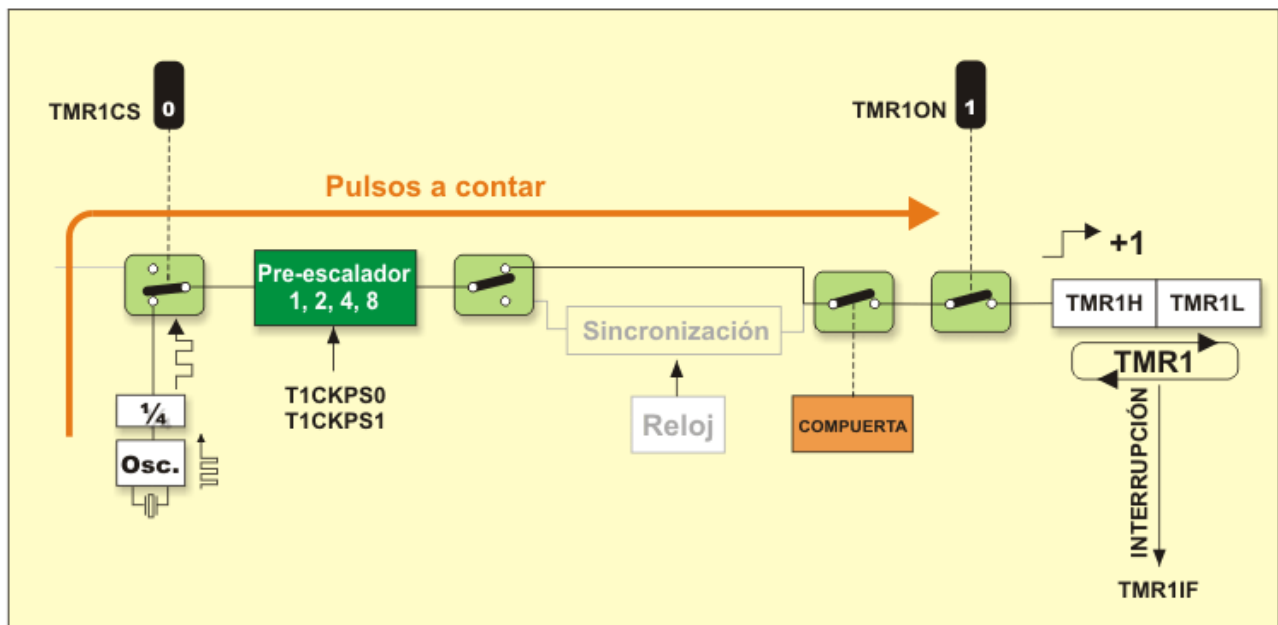
COMPUERTA DEL TEMPORIZADOR TIMER1

El pin TG1 o la salida del comparador C2 pueden ser una fuente de los pulsos que pasan por la compuerta del temporizador Timer1. Se configuran por software. Esta compuerta permite que el temporizador mida directamente la duración de los eventos externos al utilizar el estado lógico del pin T1G o los eventos analógicos al utilizar la salida del comparador C2. Refiérase a la Figura en la página anterior. Para medir duración de señal, basta con habilitar esta compuerta y contar los pulsos que pasan por ella.

TIMER1 EN EL MODO TEMPORIZADOR

Para seleccionar este modo, es necesario poner a 0 el bit TMR1CS. Después de eso, el registro de 16 bits será incrementado con cada pulso generado por el oscilador interno. Si se utiliza el cristal de cuarzo de 4 MHz, el registro será incrementado cada microsegundo.

En este modo, el bit T1SYNC no afecta al temporizador porque cuenta los pulsos de reloj interno. Como todos los dispositivos utilizan estos pulsos, no hace falta sincronizarlos.



El oscilador de reloj del microcontrolador no funciona durante el modo de reposo así que el desbordamiento en el registro del temporizador no puede causar interrupción.

Vamos a hacerlo en mikroC...

*// En este ejemplo, el TMR1 está configurado como un temporizador con el valor
// del preescalador 1:8. Cada vez que ocurra un desbordamiento de los registros TMR1H y
// TMR1L, se solicitará una interrupción.*

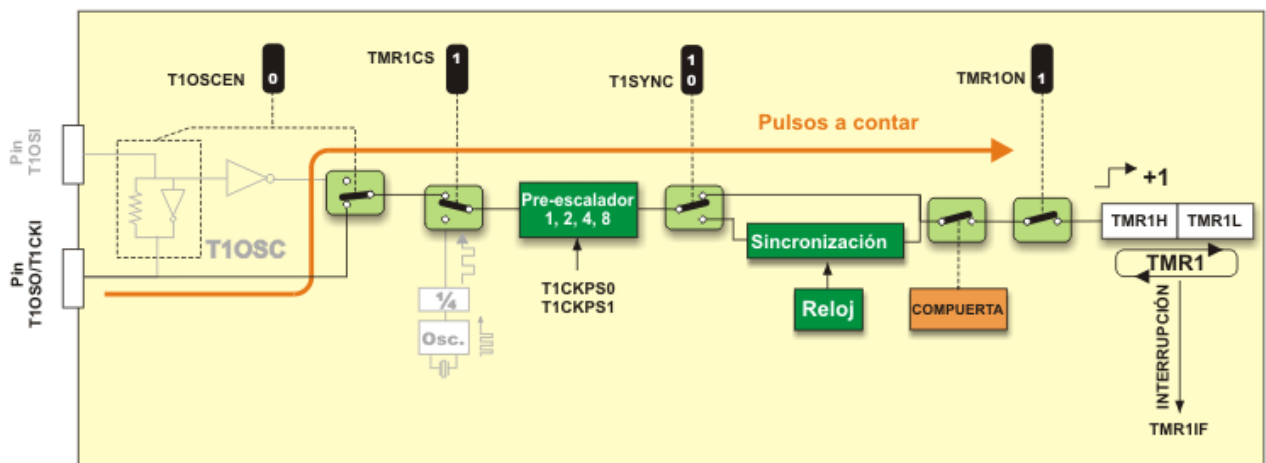
```
void main() {
  PIR1.TMR1IF = 0;    // Poner a 0 la bandera de bit del TMR1IF
  TMR1H = 0x22;      // Poner el valor inicial para el temporizador Timer1
  TMR1L = 0x00;
  TMR1CS = 0;        // Temporizador1 cuenta los pulsos del oscilador interno
}
```

```
T1CKPS1 = T1CKPS0 = 1; // El valor del pre-escalador asignada es 1:8
PIE1.TMR1IE = 1; // Interrupción habilitada por desbordamiento
INTCON = 0xC0; // Interrupción habilitada (bits GIE y PEIE)
TMR1ON = 1; // Encender el temporizador Timer1
...
```

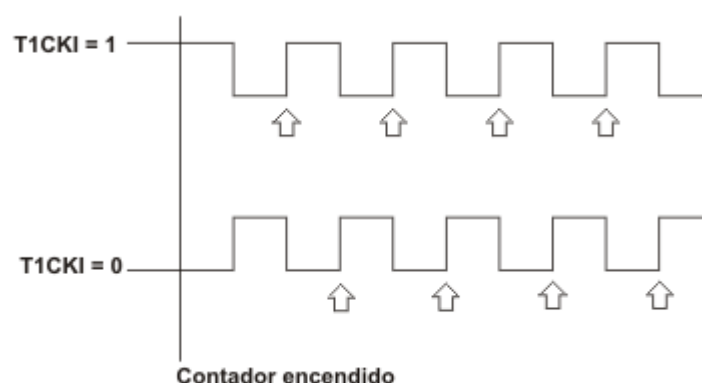
TIMER1 EN EL MODO CONTADOR

El temporizador Timer1 se pone a funcionar como un contador al poner a 1 el bit TMR1CS. Este bit cuenta los pulsos llevados al pin PC0/T1CKI y se incrementa en el flanco ascendente de la entrada del reloj externo T1CKI. Si el bit de control T1SYNC del registro T1CON se pone a 0, las entradas del reloj externo se sincronizarán en su camino al temporizador Timer1. En otras palabras, el temporizador Timer1 se sincroniza con el reloj interno del microcontrolador y se le denomina contador síncrono.

Al poner en modo de reposo el microcontrolador que funciona de esta manera, los registros del temporizador TMR1H y TMR1L no serán incrementados aunque los pulsos de reloj aparezcan en los pines de entrada. Como el reloj interno del microcontrolador no funciona en este modo, no hay entradas de reloj que se utilicen para la sincronización. De todas formas, el pre-escalador sigue funcionando siempre que haya pulsos de reloj en los pines, porque es un simple divisor de frecuencias.



Este contador detecta un uno lógico (1) en los pines de entrada. Cabe destacar que al menos un flanco ascendente debe ser detectado antes de empezar a contar los pulsos. Refiérase a la Figura a la izquierda. Las flechas en la figura indican los incrementos del contador.



Registro T1CON

R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON		Nombre de bit
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		

Leyenda

R/W (0)	Bit de lectura/escritura
(0)	Después del reinicio, el bit se pone a 0

T1GINV - Timer1 Gate Invert bit (Bit inversor de la compuerta del temporizador1) se comporta como un inversor del estado lógico en la compuerta formada por el pin T1G o la salida (C2OUT) del comparador C2. Este bit habilita al temporizador para con tar los pulsos cuando la compuerta esté a alto o a bajo.

- 1 - Temporizador 1 cuenta los pulsos cuando el pin T1G o el bit C2OUT estén a alto (1).
- 0 - Temporizador 1 cuenta los pulsos cuando el pin T1G o el bit C2OUT estén a bajo (0).

TMR1GE - Timer1 Gate Enable bit (Bit de habilitación de la compuerta del temporizador1) determina si la compuerta formada por el pin T1G o salida del comparador C2 (C2OUT) estará activa o no. Este bit es funcional sólo en caso de que el temporizador Timer1 esté encendido (el bit TMR1ON = 1). De lo contrario, este bit se ignora.

- 1 - Temporizador Timer1 está encendido sólo si la compuerta no está activa.
- 0 - Compuerta no afecta al temporizador Timer1.

T1CKPS1, T1CKPS0 - Timer1 Input Clock Prescale Select bits (Bits de selección del preescalador de señal de reloj del Temporizador1) determina el valor del divisor de frecuencias asignada al temporizador Timer1.

T1CKPS1	T1CKPS0	Valor del pre-escalador
0	0	1:1
0	1	1:2
1	0	1:4
1	1	1:8

T1OSCEN - LP Oscillator Enable Control bit (bit de habilitación del oscilador LP del Timer1)

- 1 - Oscilador LP está habilitado para el reloj del Timer1 (oscilador de bajo consumo y de frecuencia de 32.768 kHz)
- 0 - Oscilador LP está apagado.

T1SYNC - Timer1 External Clock Input Synchronization Control bit (Bit de control de sincronización de la señal de entrada) habilita la sincronización de la entrada del oscilador LP o de la entrada del pin T1CKI con el reloj interno del microcontrolador. Este bit se ignora al contar los pulsos desde el oscilador principal (el bit TMR1CS = 0).

- 1 - Entrada de reloj externa no está sincronizada.
- 0 - Entrada de reloj externa está sincronizada.

TMR1CS - Timer TMR1 Clock Source Select bit (bit de selección de la fuente de reloj del temporizador Timer1)

- 1 - Cuenta los pulsos por el pin T1CKI (por el flanco ascendente 0-1)
- 0 - Cuenta los pulsos del reloj interno del microcontrolador

TMR1ON - Timer1 On bit (TMR activo, hace entrar o no en funcionamiento el Timer1).

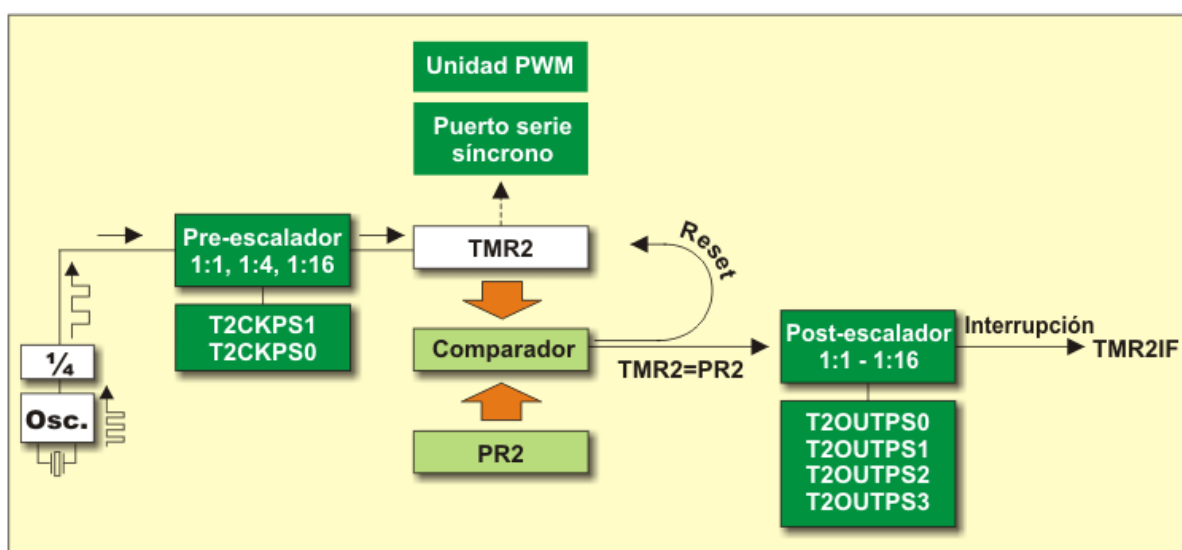
- 1 - Habilita el temporizador Timer1.
- 0 - Deshabilita el temporizador Timer1.

Para utilizar el Timer1 apropiadamente, es necesario hacer lo siguiente:

- Como no es posible apagar el pre-escalador, su valor debe estar ajustado a los bits T1CKPS1 y T1CKPS0 del registro T1CON (Refiérase a la tabla).
- Seleccionar el modo por el bit TMR1CS del registro T1CON. (TMR1CS: 0=la fuente de reloj es oscilador de cuarzo interno, 1= la fuente de reloj es oscilador de cuarzo externo).
- Al configurar el bit T1OSCEN del mismo registro, el oscilador está habilitado y los registros TMR1H y TMR1L se incrementan con cada pulso de reloj. Al poner este bit a 0, se detiene el conteo.
- Al reiniciar los registros del contador o al escribir en ellos, se reinicia el pre-escalador.
- Al llenar ambos registros del temporizador, se configura la bandera TMR1IF y el conteo empieza desde cero.

TEMPORIZADOR TIMER2

El módulo del temporizador Timer2 es un temporizador de 8 bits.



Los pulsos que vienen del oscilador de cuarzo primero pasan por el pre-escalador cuyo valor puede ser modificado al combinar los bits T2CKPS1 y T2CKPS0. La salida del preescalador se utiliza para incrementar el registro TMR2 empezando por 00h. Los valores del TMR2 y del PR2 se comparan constantemente y el registro TMR2 va incrementándose hasta alcanzar el valor del registro PR2. Cuando se igualan los valores de los registros, lo que será registrado por el comparador, el TMR2 se reinicia a 00h automáticamente. El postescalador del temporizador Timer2 se incrementa y su salida se utiliza para generar una interrupción si está habilitada.

Los ambos registros TMR y PR2 son de lectura y escritura. El conteo se puede detener al poner a 0 el bit TMR2ON, lo que resulta en un ahorro de energía.

El momento de reinicio del TMR2 se puede utilizar para determinar la velocidad de transmisión en baudios de la comunicación serie síncrona.

Varios bits del registro T2CON están en control del temporizador Timer2.

Registro T2CON

T2CON	-	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
	Bit 7	TOUTPS3 Bit 6	TOUTPS2 Bit 5	TOUTPS1 Bit 4	TOUTPS0 Bit 3	TMR2ON Bit 2	T2CKPS1 Bit 1	T2CKPS0 Bit 0	Nombre de bit

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
(0)	Después del reinicio, el bit se pone a 0

TOUTPS3 - TOUTPS0 - Timer2 Output Postcaler Select bits (bits de selección del rango del divisor del post-escalador para el Timer2) se utilizan para determinar el valor del post-escalador según la siguiente tabla:

TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	Valor del post-escalador
0	0	0	0	1:1
0	0	0	1	1:2
0	0	1	0	1:3
0	0	1	1	1:4
0	1	0	0	1:5
0	1	0	1	1:6
0	1	1	0	1:7
0	1	1	1	1:8
1	0	0	0	1:9
1	0	0	1	1:10
1	0	1	0	1:11
1	0	1	1	1:12
1	1	0	0	1:13
1	1	0	1	1:14
1	1	1	0	1:15
1	1	1	1	1:16

TMR2ON Timer2 On bit - (bit de activación del TIMR2) hace entrar en funcionamiento el temporizador Timer2.

- 1 - Habilita el funcionamiento del Timer2.
- 0 - Deshabilita el funcionamiento del Timer2.

T2CKPS1, T2CKPS0 - Timer2 Clock Prescaler bits (selección del rango del divisor del preescalador del Timer2) determina el valor del divisor de frecuencias:

T2CKPS1	T2CKPS0	Valor del pre-escalador
0	0	1:1
0	1	1:4
1	x	1:16

Al utilizar el temporizador Timer2 hay que saber varios detalles relacionados con sus registros:

- En el momento de encender una fuente de alimentación, el registro PR2 contiene el valor FFh.
- Tanto el pre-escalador como el post-escalador se borran al escribir en el registro TMR2.
- Tanto el pre-escalador como el post-escalador se borran al escribir en el registro T2CON.
- Al producirse cualquier reinicio, como puede anticiparse, tanto el pre-escalador como el post-escalador se borran.

Los módulos CCP pueden funcionar en muchos modos diferentes, por lo que se consideran los más complicados. Si usted intenta analizar su funcionamiento a base de tablas que describen las funciones de bits, comprenderá mejor de lo que le estamos hablando. Si utiliza alguno de los módulos CCP, primero seleccione el modo que necesita, analice la figura apropiada y entonces póngase a modificar los bits de registros. Si no...

MÓDULOS CCP

El módulo CCP (*Captura/Comparación/PWM*) es un periférico que le permite medir y controlar diferentes eventos.

El **modo de captura** proporciona el acceso al estado actual de un registro que cambia su valor constantemente. En este caso, es el registro del temporizador Timer1.

El **modo de comparación** compara constantemente valores de dos registros. Uno de ellos es el registro del temporizador Timer1. Este circuito también le permite al usuario activar un evento externo después de que haya expirado una cantidad de tiempo predeterminada.

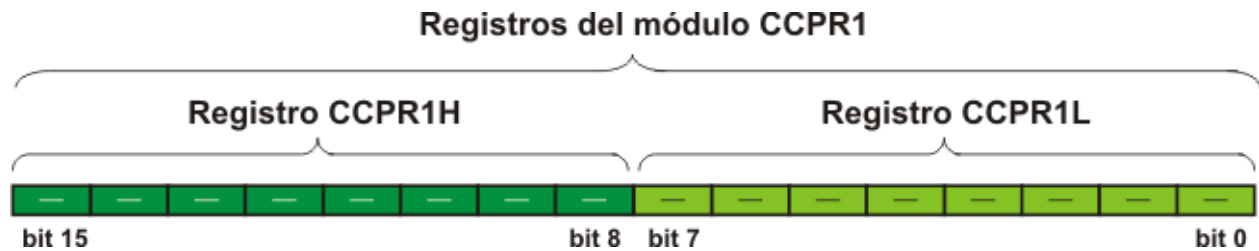
PWM (*Pulse Width Modulation* - modulación por ancho de pulsos) puede generar señales de frecuencia y de ciclo de trabajo variados por uno o más pines de salida.

El microcontrolador PIC16F887 dispone de dos módulos CCP - CCP1 y CCP2.

Ambos son idénticos en modo normal de funcionamiento, mientras que las características del PWM mejorado están disponibles sólo en el modo CCP1. Ésta es la razón por la que en este capítulo se describe detalladamente el funcionamiento del módulo CCP1. Con respecto al CCP2, se presentarán sólo las características que lo distinguen del CCP1.

MÓDULO CCP1

Una parte central de este circuito es un registro CCPR1 de 16 bits que consiste en registros CCPR1L y CCOR1H. Se utiliza para capturar y comparar sus valores con los números almacenados en el registro del temporizador Timer1 (TMR1H y TMR1L).



Si está habilitado por software, puede ocurrir el reinicio del temporizador Timer1 al igualarse los valores en modo de Comparación. Además, el módulo CCP1 puede generar señales PWM de frecuencia y de ciclo de trabajo variados.

Los bits del registro CCP1CON están en control del módulo CCP1.

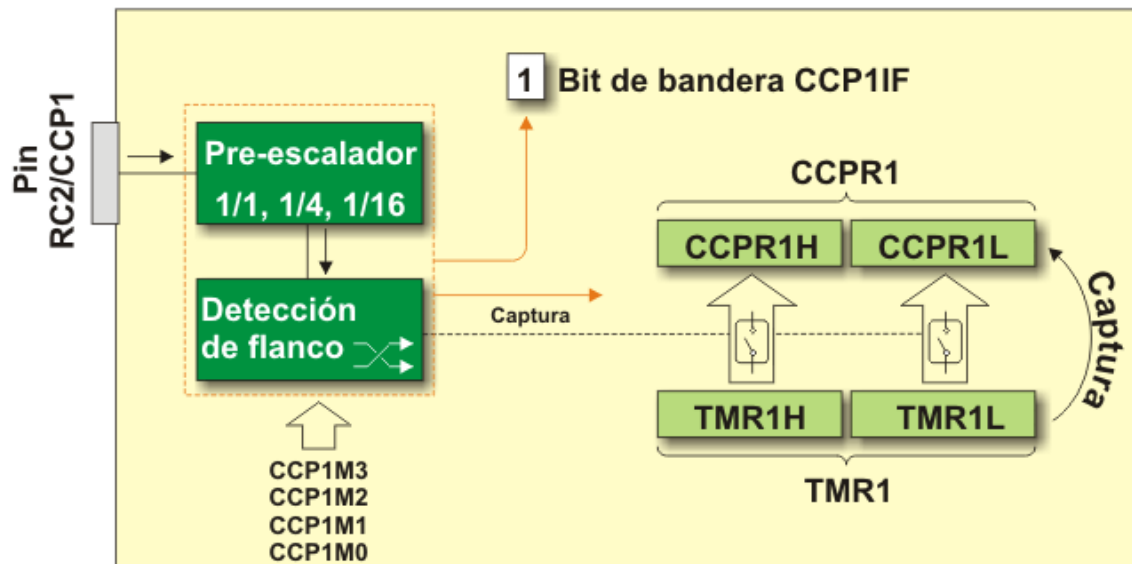
CCP1 EN MODO DE CAPTURA

En este modo, el registro del temporizador Timer1 (que consiste en los TMR1H y TMR1L) se copia al registro CCP1 (que consiste en los CCPR1H y CCPR1L) en las siguientes situaciones:

- Cada flanco ascendente (1 -> 0) en el pin RC2/CCP;
- Cada flanco descendente (0 -> 1) en el pin RC2/CCP1;
- Cada cuarto flanco ascendente (0 -> 1) en el pin RC2/CCP1; y
- Cada decimosexto flanco descendente (0 -> 1) en el pin RC2/CCP1.

Una combinación de cuatro bits (CCP1M3 - CCP1M0) del registro de control determina cuál de estos eventos causará transmisión de dato de 16 bits. Además, se deben cumplir los siguientes requisitos::

- El pin RC2/CCP1 debe estar configurado como entrada; y
- El Timer1 debe funcionar como temporizador o contador síncrono.



El bit de bandera CCP1IF se pone a uno después de acabar la captura. Si se pone a 1 el bit CCP1IE del registro PIE1, se producirá una interrupción.

En caso de que el módulo CCP1 esté en modo de captura, puede producirse una interrupción no deseada. Para evitarlo, antes de que ocurra un cambio en el registro de control se deben poner a 0 tanto el bit que habilita la interrupción CCP1IE, como el bit de bandera CCP1IF.

Las interrupciones no deseadas pueden producirse al cambiar el valor del pre-escalador. Para evitarlo, el módulo CCP1 debe estar apagado temporalmente antes de cambiar el valor del pre-escalador.

Se recomienda la siguiente secuencia de programa, escrita en ensamblador:

```
BANKSEL CCP1CON
```

```
CLRF CCP1CON ;REGISTRO DE CONTROL BORRADO
           ;MÓDULO CCP1 ESTÁ APAGADO
```

```
MOVLW XX ;NUEVO MODO DEL PRE-ESCALADOR ESTÁ SELECCIONADO
```

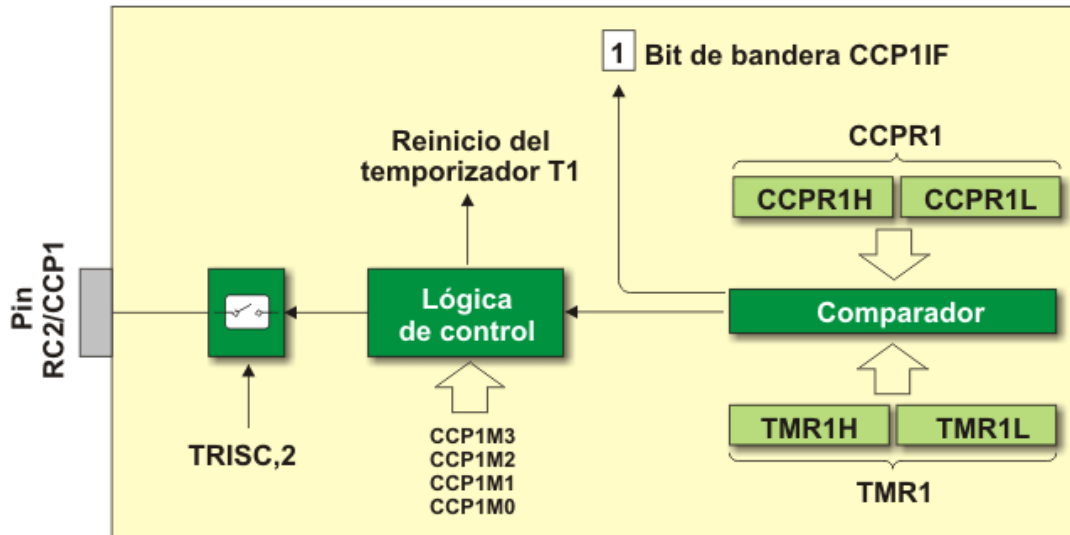
```
MOVWF CCP1CON ;EN EL REGISTRO DE CONTROL SE INTRODUCES UN NUEVO VALOR
           ;MÓDULO CCP1 SE ENCIENDE SIMULTÁNEAMENTE
```

Vamos a hacerlo en mikroC...

```
...
ASM {
  BANKSEL CCP1CON
  CLRF CCP1CON // REGISTRO DE CONTROL BORRADO
              // MÓDULO CCP1 ESTÁ APAGADO
  MOVLW XX // NUEVO MODO DEL PRE-ESCALADOR ESTÁ SELECCIONADO
  MOVWF CCP1CON // EN EL REGISTRO DE CONTROL SE INTRODUCES NUEVO VALOR
} // MÓDULO CCP1 SE ENCIENDE SIMULTÁNEAMENTE
...
```

CCP1 EN MODO DE COMPARACIÓN

En este modo, el valor almacenado en el registro CCP1 se compara constantemente al valor almacenado en el registro del temporizador Timer1. Al igualarse los valores, el estado lógico en el pin de salida puede ser cambiado, lo que depende del estado de bits en el registro de control (CCP1M3 - CCP1M0). El bit de bandera CCP1IF se pone a uno simultáneamente.

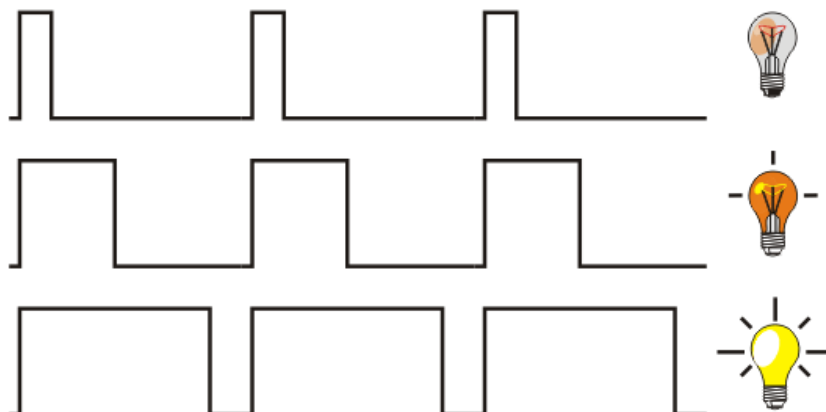


Para poner el módulo CCP1 en este modo de funcionamiento, se deben cumplir dos condiciones:

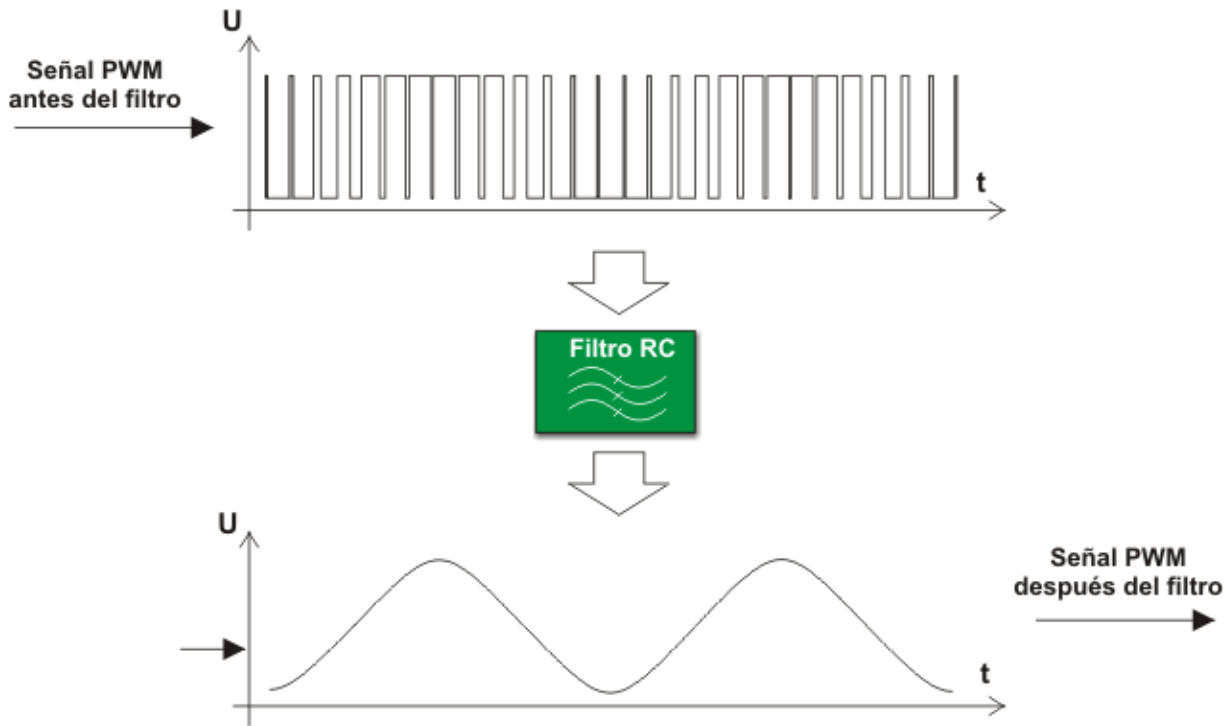
- El pin RC2/CCP1 debe estar configurado como salida; y
- El temporizador Timer1 debe estar sincronizado con el reloj interno.

CCP1 EN MODO PWM

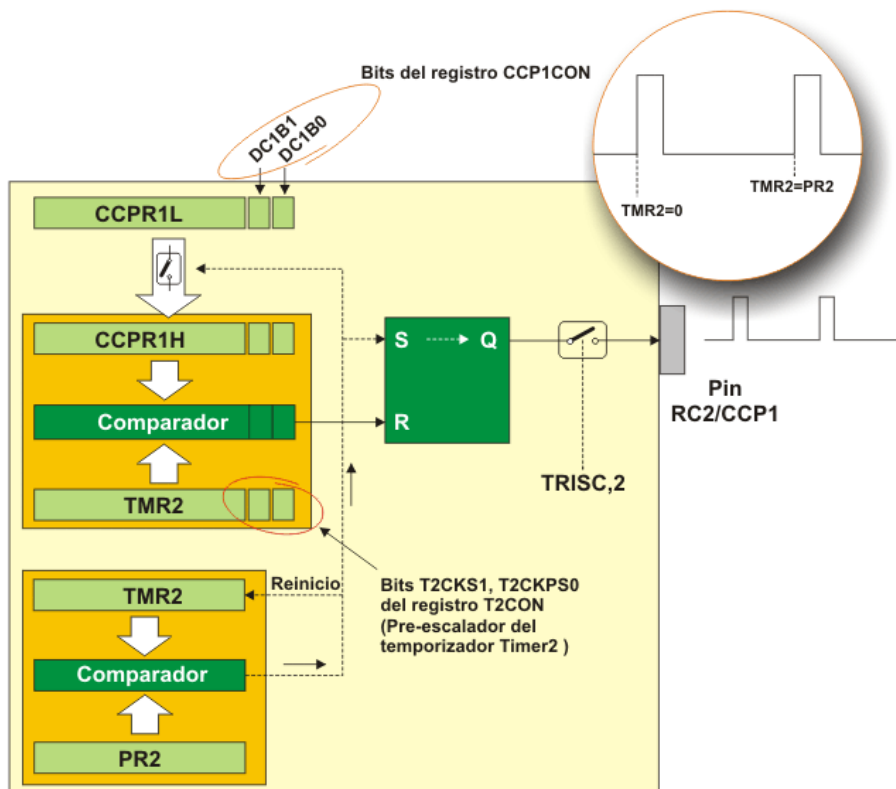
Las señales de frecuencia y de ciclo de trabajo variados tienen una amplia gama de aplicaciones en automatización. Un ejemplo típico es un circuito de control de potencia. Refiérase a la siguiente figura. Si un cero lógico (0) indica un interruptor abierto y un uno lógico (1) indica un interruptor cerrado, la potencia eléctrica que se transmite a los consumidores será directamente proporcional a la duración del pulso. Esta relación se le denomina *Ciclo de Trabajo*.



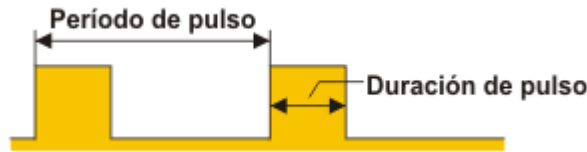
El otro ejemplo, común en la práctica, es el uso de señales PWM en un circuito para generar señales de forma de onda arbitraria como una onda sinusoidal. Vea la siguiente figura:



Los dispositivos que funcionan según este principio se utilizan con frecuencia en la práctica como variadores de frecuencia ajustable que controlan motores eléctricos (velocidad, aceleración, desaceleración etc.)



La Figura anterior muestra el diagrama de bloques del módulo CCP1 puesto en el modo PWM. Para generar un pulso de forma arbitraria en el pin de salida, es necesario ajustar el período de pulsos (frecuencia) y la duración de pulsos.



PERÍODO DE PWM

El período de pulso de salida (T) se determina por el registro PR2 del temporizador Timer2. El período de PWM se puede calcular por la siguiente ecuación:

$$\text{Período PWM} = (\text{PR2} + 1) * 4T_{\text{osc}} * \text{Valor de pre-escala del Timer2}$$

Si el período de PWM (T) es conocido, es fácil determinar la frecuencia de señal F, porque estos dos valores están relacionados por la ecuación $F=1/T$.

CICLO DE TRABAJO DE PWM

El ciclo de trabajo de PWM se especifica al utilizar en total 10 bits: los ocho bits más significativos del registro CCPR1L y los dos bits menos significativos adicionales del registro CCP1CON (DC1B1 y DC1B0). El resultado es un número de 10 bits dado por la siguiente fórmula:

$$\text{Ancho de pulsos} = (\text{CCPR1L}, \text{DC1B1}, \text{DC1B0}) * T_{\text{osc}} * \text{Valor de pre-escala del Timer2}$$

La siguiente tabla muestra cómo generar las señales PWM de diferentes frecuencias cuando el microcontrolador utiliza un cristal de cuarzo de 20 MHz ($T_{\text{osc}}=50\text{nS}$).

Frecuencia [KHz]	1.22	4.88	19.53	78.12	156.3	208.3
Pre-escalador del TMR2	16	4	1	1	1	1
Registro PR2	FFh	FFh	FFh	3Fh	1Fh	17h

Notas adicionales:

- El pin de salida se va a poner a 1 constantemente, si por error el ancho de pulso generado es más largo que el período de PWM.
- En esta aplicación, no se puede utilizar el post-escalador del temporizador Timer2 para generar períodos de PWM largos.

RESOLUCIÓN DE PWM

Una señal PWM no es nada más que una secuencia de pulsos que varían su ciclo de trabajo. Para una frecuencia específica (número de pulsos por segundo), hay un número limitado de combinaciones de ciclos de trabajo. Este número representa una resolución medida en bits. Por ejemplo, si una resolución es de 10 bits estarán disponibles 1024

ciclos de trabajo discretos; si una resolución es de 8 bits estarán disponibles 256 ciclos de trabajo discretos etc. En este microcontrolador la resolución es determinada por el registro PR2. El máximo valor se obtiene al usar el número FFh.

Frecuencias y resoluciones de PWM (Fosc = 20MHz):

Frecuencia de PWM	1.22kHz	4.88kHz	19.53kHz	78.12kHz	156.3kHz	208.3kHz
Pre-escala del temporizador	16	4	1	1	1	1
Valor del PR2	FFh	FFh	FFh	3Fh	1Fh	17h
Resolución máxima	10	10	10	8	7	6

Frecuencias y resoluciones de PWM (Fosc = 8MHz):

Frecuencia del PWM	1.22kHz	4.90kHz	19.61kHz	76.92kHz	153.85kHz	200.0kHz
Pre-escala del temporizador	16	4	1	1	1	1
Valor del PR2	65h	65h	65h	19h	0Ch	09h
Resolución máxima	8	8	8	6	5	5

Vamos a hacerlo en mikroC...

/ En este ejemplo, el módulo PWM está inicializado y ajustado para producir una secuencia de pulsos de ciclo de trabajo del 50%. Para este propósito, se utilizan las funciones PWM1_Init(), PWM1_Start() y PWM1_Set_Duty(). Todas las funciones las contiene la librería PWM del mikroC PRO for PIC. Sólo es necesario copiarlas al programa */*

```
unsigned short duty_c; // Definir la variable duty_c
```

```
void initMain() {
    ANSEL = ANSELH = 0; // Todos los pines de E/S se configuran como digitales
    PORTC = TRISC = 0; // Estado inicial de los pines de salida del puerto PORTC
    PWM1_Init(5000); // Inicialización del módulo PWM (5KHz)
}
```

```
void main() {
    initMain();
    duty_c = 127; // Valor inicial del ciclo de trabajo
    PWM1_Start(); // Iniciar el módulo PWM1
    PWM1_Set_Duty(duty_c); // Ajustar el ciclo de trabajo de PWM al 50%
    ...
}
```

Registro CCP1CON

CCP1CON	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

R/W Bit de lectura/escritura
(0) Después del reinicio, el bit se pone a cero

P1M1, P1M0 - PWM Output Configuration bits (bits de configuración del modo PWM) - El pin P1A es la entrada del módulo de Captura/Comparación en todos los modos, menos en modo PWM. Los pines P1B, P1C y P1D actúan como los pines de E/S del puerto D.

En modo PWM estos bits afectan al funcionamiento del módulo CCP1 como se muestra en la siguiente tabla:

P1M1	P1M0	Modo
PWM con una sólo salida		
0	0	Por el pin P1A sale una señal modulada. Pines P1B, P1C y P1D son entradas/salidas del puerto D.
Configuración Full Bridge - Forward (puente completo con salida directa)		
0	1	Por el pin P1D sale una señal modulada. Por el pin P1D sale una señal modulada. Pines P1B y P1C están inactivos.
Configuración Half Bridge (medio-puente)		
1	0	Por los pines P1A y P1B sale una señal modulada. Pines P1C y P1D son entradas/salidas del puerto D.
Configuración Full Bridge - Reverse (puente completo con salida inversa)		
1	1	Por el pin P1B sale una señal modulada. Pin P1C está activo. Pines P1A y P1D están inactivos.

DC1B1, DC1B0 - PWM Duty Cycle Least Significant bits (bits menos significativos del ciclo de trabajo de PWM) - Se utilizan sólo en el modo PWM y representan dos bits menos significativos de un número de 10 bits. Este número determina el ciclo de trabajo de la señal PWM. Los demás 8 bits se almacenan en el registro CCPR1L.

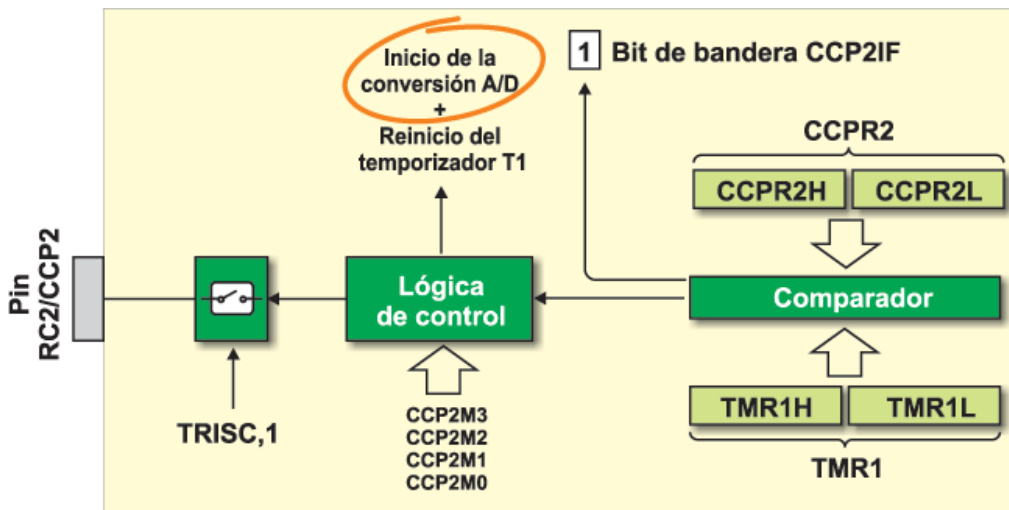
CCP1M3 - CCP1M0 - (bits de selección de modo del módulo CCP1) determina el modo del módulo CCP1.

CCP1M3	CCP1M2	CCP1M1	CCP1M0	Modo
0	0	0	0	Módulo está deshabilitado (reinicio).
0	0	0	1	No utilizado.
Modo de comparación				
0	0	1	0	El bit CCP1IF bit se pone a 1 al ocurrir una coincidencia.
0	0	1	1	No utilizado.
Modo de captura				
0	1	0	0	Cada flanco descendente en el pin CCP1.
0	1	0	1	Cada flanco ascendente en el pin CCP1.
Modo de captura				
0	1	1	0	Cada cuarto flanco ascendente en el pin CCP1.

0	1	1	1	Modo de captura Cada decimosexto flanco ascendente en el pin CCP1.
1	0	0	0	Modo de comparación La salida y el bit CCP1IF se ponen a 1 al ocurrir una coincidencia
1	0	0	1	Modo de comparación La salida se pone a 0 y el bit CCP1IF se pone a 1 al ocurrir una coincidencia.
1	0	1	0	Modo de comparación Llega la solicitud de interrupción y el bit CCP1IF se pone a 1 al ocurrir una coincidencia
1	0	1	1	Modo de comparación El bit CCP1IF se pone a 1, y los registros de temporizadores 1 o 2 se borran al ocurrir una coincidencia
1	1	0	0	Modo PWM Pines P1A y P1C están activos a nivel alto. Pines P1B y P1D están activos a nivel alto.
1	1	0	1	Modo PWM Pines P1A y P1C están activos a nivel alto. Pines P1B y P1D están activos a nivel bajo.
1	1	1	0	Modo PWM Pines P1A y P1C están activos a nivel bajo. Pines P1B y P1D están activos a nivel alto.
1	1	1	1	Modo PWM Pines P1A y P1C están activos a nivel bajo. Pines P1B y P1D están activos a nivel bajo.

MÓDULO CCP2

Con exclusión de los nombres diferentes de los registros y de los bits, este módulo es una muy buena copia del módulo CCP1 puesto en modo normal. La única diferencia significativa entre ellos es el funcionamiento en modo de comparación del módulo CCP2. La diferencia se refiere a la señal de reinicio del temporizador T1. Concretamente, si el convertidor A/D está habilitado, al igualarse los valores de los registros TMR1 y CCPR2, la señal de reinicio del temporizador T1 iniciará automáticamente la conversión A/D. Similar al módulo anterior, este circuito también está bajo el control de los bits del registro de control. Esta vez es el registro CCP2CON.



Registro CCP2CON

CCP2CON		-	-	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	Características
		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Nombre de bit
				R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

Leyenda

- Bit no implementado
- R/W Bit de lectura/escritura
- (0) Después del reinicio, el bit se pone a cero

DC2B1, DC2B0 - PWM Duty Cycle Least Significant bits (bits menos significativos del ciclo de trabajo de PWM) - Se utilizan sólo en modo PWM y representan dos bits menos significativos de un número de 10 bits. Este número determina el ciclo de trabajo de la señal PWM. Los demás 8 bits se almacenan en el registro CCPR2L.

CCP2M3 - CCP2M0 - CCP2 Mode Select bits (bits de selección de modo del módulo CCP2) determina el modo del módulo CCP2.

CCP2M3	CCP2M2	CCP2M1	CCP2M0	Modo
0	0	0	0	Módulo está deshabilitado (reinicio).
0	0	0	1	No utilizado.
0	0	1	0	No utilizado.
0	0	1	1	No utilizado.
0	1	0	0	Modo de Captura Cada flanco descendente en el pin CCP2.
0	1	0	1	Modo de Captura Cada flanco ascendente en el pin CCP2.
0	1	1	0	Modo de Captura Cada cuarto flanco ascendente en el pin CCP2.
0	1	1	1	Modo de Captura Cada decimosexto flanco ascendente en el pin CCP2.
1	0	0	0	Modo de comparación

				La salida y el bit CCP2IF se ponen a 1 al ocurrir una coincidencia.
				Modo de comparación
1	0	0	1	La salida se pone a 0 y el bit CCP2IF se pone a 1 al ocurrir una coincidencia
				Modo de comparación
1	0	1	0	Se produce una interrupción, el bit CCP2IF se pone a 1 y no hay cambio el pin CCP2 pin al ocurrir una coincidencia.
				Modo de comparación
1	0	1	1	Al ocurrir una coincidencia, el bit CCP2IF se pone a 1, los registros del temporizador 1 se borran y la conversión A/D se inicia si el convertidor A/D está habilitado.
1	1	x	x	Modo PWM

¿Cómo configurar e iniciar el módulo CCP1 para funcionar en modo PWM?

Para configurar e iniciar el módulo CCP1 para funcionar en modo PWM, siga los siguientes pasos:

- Deshabilitar el pin de salida del CCP1. Deberá estar configurado como entrada.
- Seleccionar el período de señal PWM al introducir el valor en el registro PR2.
- Configurar el módulo CCP1 para funcionar en modo PWM al combinar los bits del registro CCP1CON.
- Ajustar el ciclo de trabajo de señal PWM al introducir el valor en el registro CCPR1L y al utilizar los bits DC1B1 y DC1B0 del registro CCP1CON.
- Configurar e iniciar el temporizador Timer2:
 - Poner a cero el bit de bandera de interrupción TMR2IF en el registro PIR1
 - Ajustar el valor de división de frecuencia del temporizador Timer2 por los bits
 - T2CKPS1 y T2CKPS0 del registro T2CON.
 - Iniciar el temporizador Timer2 al poner a uno el bit TMR2ON del registro T2CON.
- Habilitar los pines de salida de PWM después de que haya sido acabado un ciclo de PWM:
 - Esperar el desbordamiento del temporizador Timer2 (el bit TMR2IF del registro PIR1 se pone a uno)
 - Configurar el pin apropiado como salida al poner a cero el bit en el registro TRIS.

MÓDULO CCP1 EN MODO MEJORADO

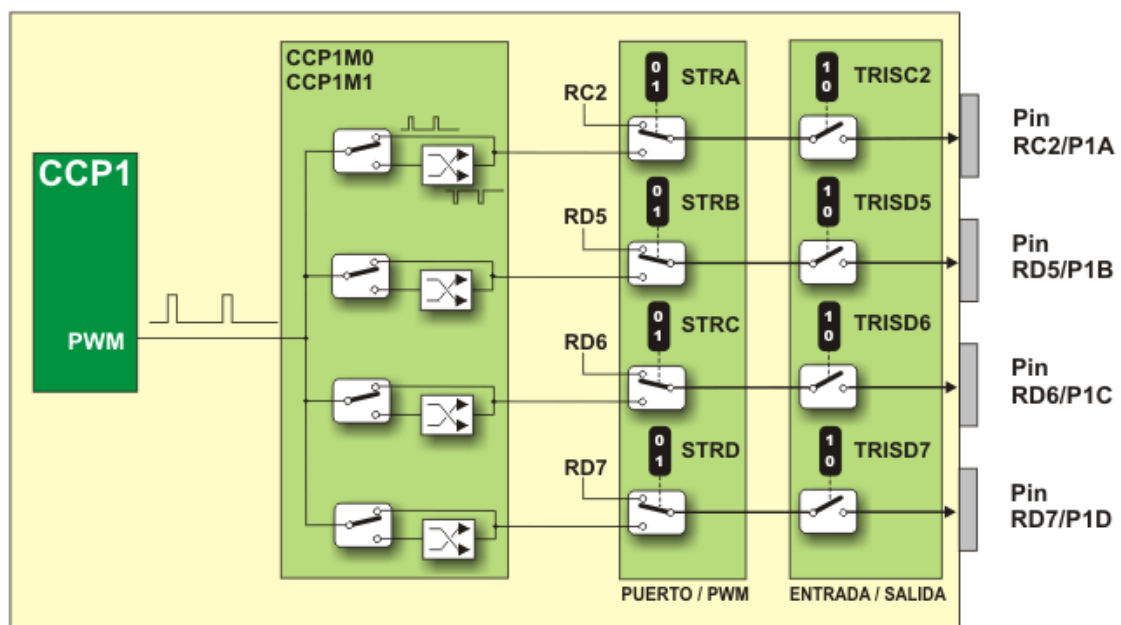
El módulo CCP1 es el único que se puede poner en modo mejorado. Este modo básicamente no difiere del modo normal del CCP1 y la mejora se refiere a la transmisión de la señal PWM a los pines de salida. ¿Por qué es eso tan importante? Por el uso cada vez más frecuente de los microcontroladores en los sistemas de control de motores eléctricos. Aquí no vamos a describir estos dispositivos, sin embargo si tiene la oportunidad de trabajar en el desarrollo de los dispositivos similares, reconocerá los elementos que se utilizaban hasta hace poco como los periféricos. Decimos &se

utilizaban& porque todos estos elementos ahora están integrados en el microcontrolador y pueden funcionar en varios modos diferentes.

MODO PWM CON UNA SALIDA

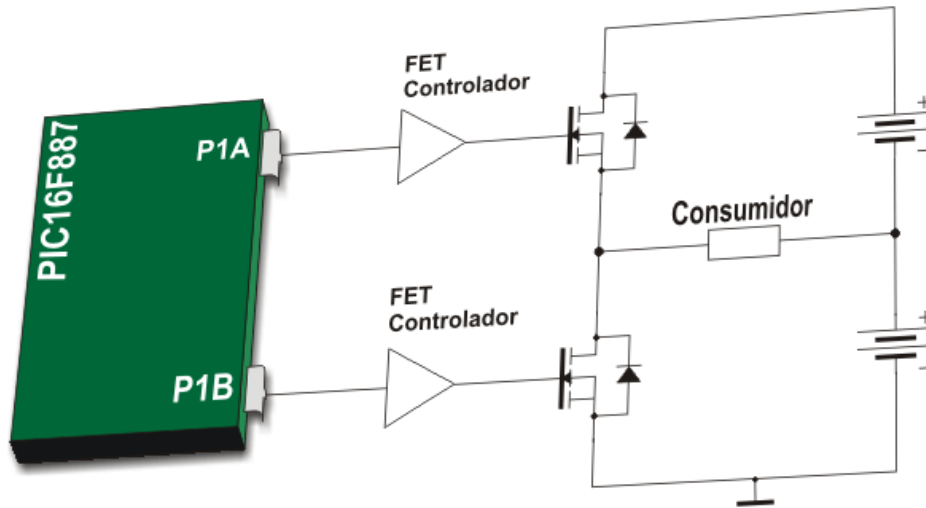
El modo PWM con una salida está habilitado sólo en el caso de que se pongan a cero los bits P1M1 y P1M0 en el registro CCP1CON. En tal caso, una señal PWM puede estar disponible simultáneamente en como máximo cuatro diferentes pines de salida. Además, la secuencia de señales PWM puede aparecer en forma de onda básica o invertida. La distribución de señales depende de los bits del registro PSTRCON, mientras que su polaridad depende de los bits CCP1M1 y CCP1M0 del registro CCP1CON.

Si se utiliza una salida invertida, los pines activos a nivel bajo y los pulsos que tienen la misma forma de onda se generan siempre en parejas: en los pines P1A y P1C así como en los pines P1B y P1D, respectivamente.

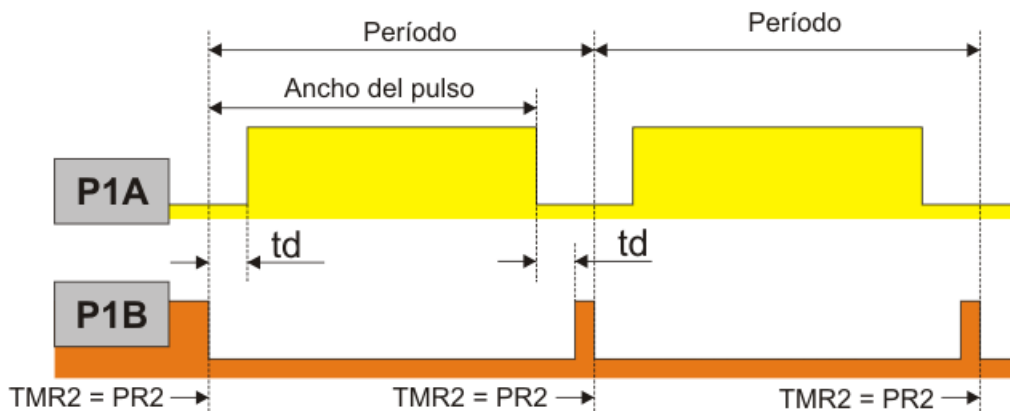


MODO DE MEDIO-PUENTE

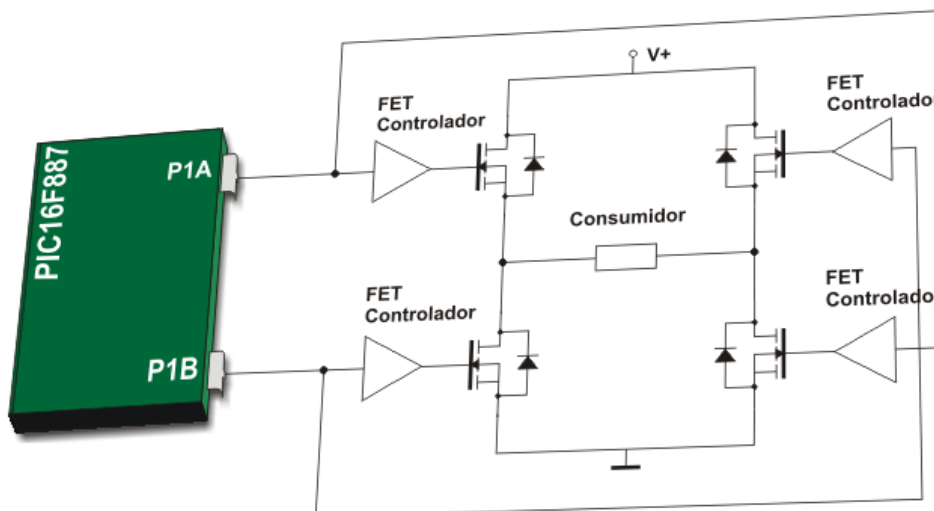
En cuanto al modo de medio-puente, la señal PWM es una salida en el pin P1A, mientras que a la vez la señal complementaria PWM es una salida en el pin P1B. Estos pulsos activan a los controladores MOSFET en modo de Medio-Puente que habilitan/deshabilitan el flujo de corriente por el dispositivo.



En este modo es muy peligroso encender los controladores MOSFET simultáneamente (el cortocircuito producido en aquel momento sería fatal). Para evitarlo, es necesario proporcionar un tiempo muerto entre encender y apagar los controladores. Este tiempo muerto está marcado con 'td' (*time delay*) en la siguiente figura. El problema se resuelve al utilizar los bits PDC0-PDC6 del registro PWM1CON.

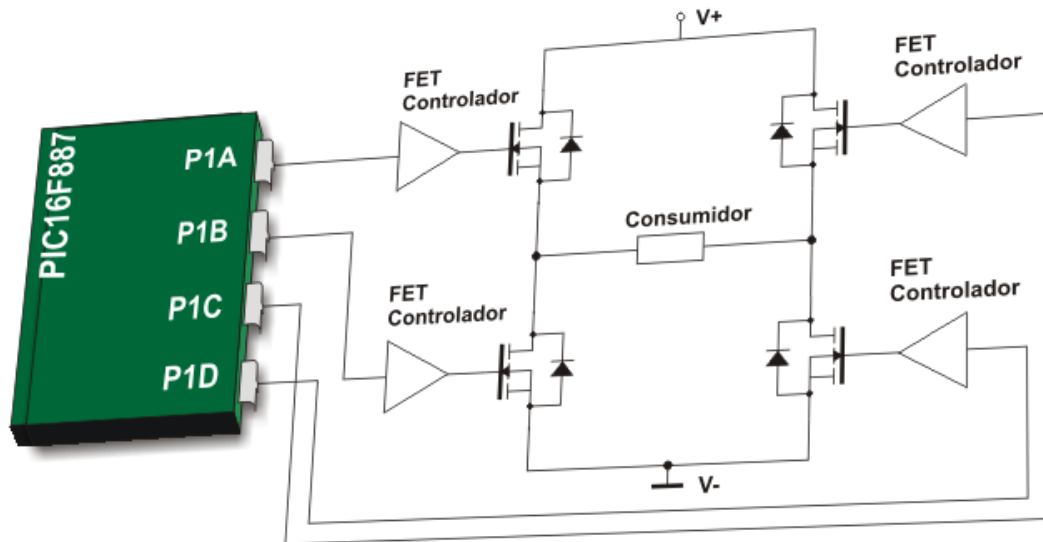


Como se muestra en la siguiente figura, el modo de medio-puente se puede utilizar para activar los controladores MOSFET en la configuración Puentes completo:



MODO PUENTE-COMPLETO

Todos los cuatro pines se utilizan como salidas en el modo Puente completo. En la práctica, este modo es utilizado con frecuencia para activar los motores, lo que proporciona un control simple y completo de velocidad y dirección de rotación. Hay dos configuraciones de este modo: Full Bridge-Forward (puente completo con salida directa) y Full Bridge-Reverse (puente completo con salida inversa).

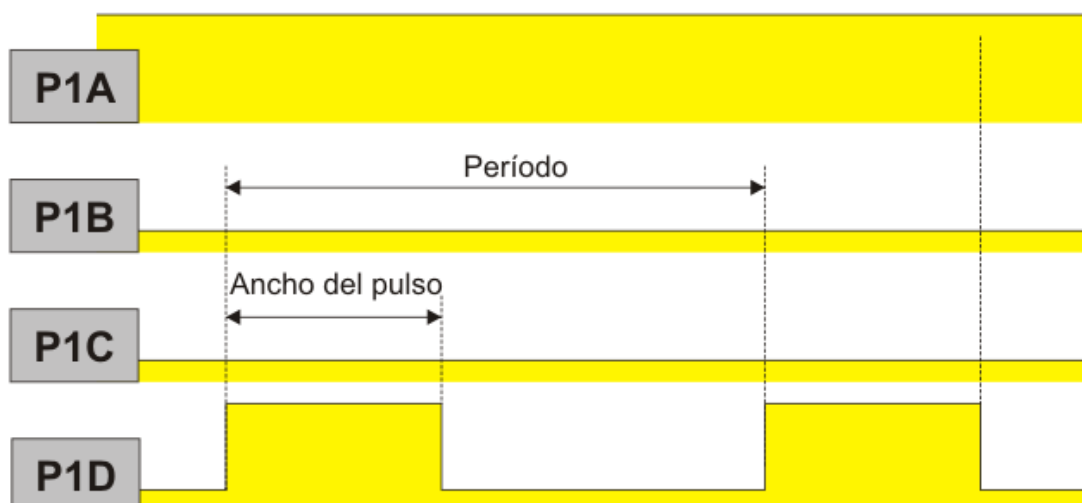


CONFIGURACIÓN PUENTE COMPLETO - DIRECTO

En modo *Directo* ocurre lo siguiente:

- Un uno lógico (1) aparece en el pin P1A (pin está activo a nivel alto);
- Secuencia de pulsos aparece en el pin P1D; y
- Un cero lógico (0) en los pines P1B y P1C (pines están activos a nivel bajo).

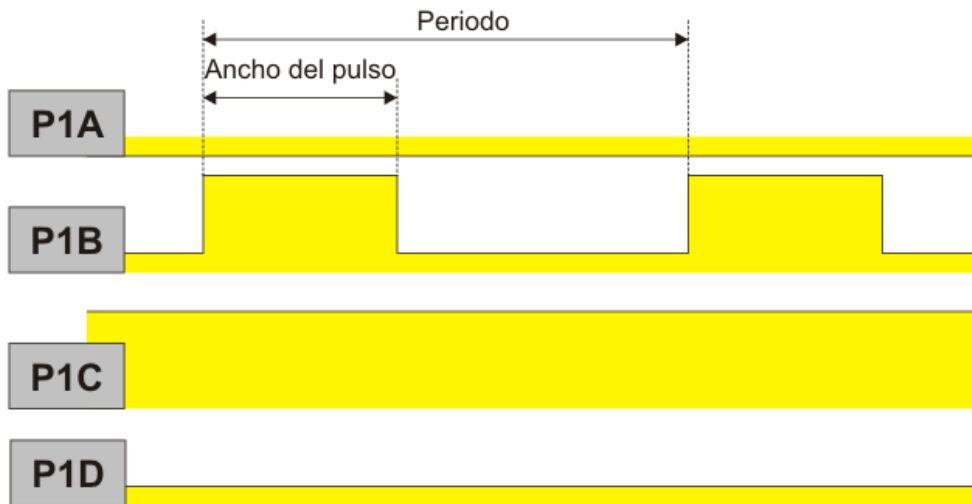
La siguiente figura muestra el estado de los pines P1A-P1D durante un ciclo PWM completo:



CONFIGURACIÓN PUENTE COMPLETO - INVERSO

Lo similar ocurre en modo *Inverso*, a menos que estos pines dispongan de funciones diferentes:

- Un uno lógico (1) aparece en el pin P1C (pin está activo a nivel alto);
- Secuencia de pulsos aparece en el pin P1B; y
- Un cero lógico (0) aparece en los pines P1A y P1D (pines están activos a nivel bajo).



Registro PWM1CON

PWM1CON								Características
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Nombre de bit
PRSEN	PDC6	PDC5	PDC4	PDC3	PDC2	PDC1	PDC0	
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

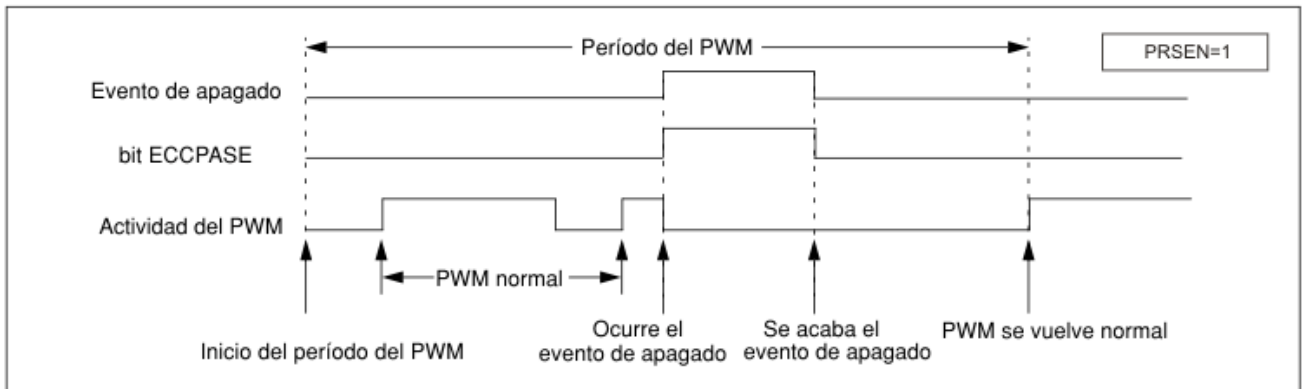
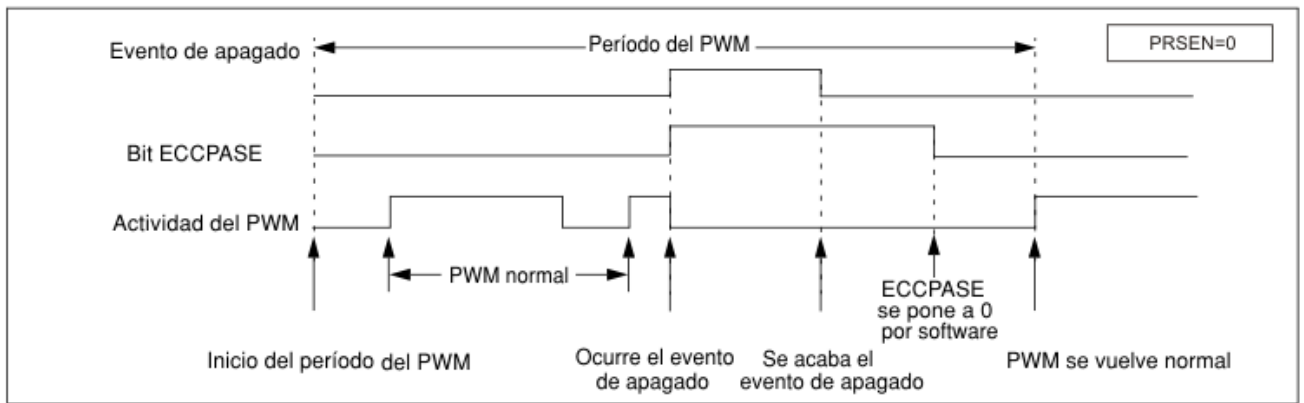
Leyenda

R/W Bit de lectura/escritura
(0) Después del reinicio, el bit se pone a cero

STRC PWM Restart Enable bit (Bit de habilitación del reinicio automático del PWM)

- 1 - Después de un apagado automático, el módulo PWM se reinicia automáticamente, y el bit ECCPASE del registro ECCPAS se pone a cero.
- 0 - Para iniciar el módulo PWM después de un apagado automático, el bit ECCPASE debe ponerse a cero por software.

PDC6 - PDC0 PWM Delay Count bits (Bits de configuración del tiempo muerto en el modo PWM) - El número binario de 7 dígitos determina el número de ciclos de instrucciones ($4 \times T_{osc}$) añadidos como tiempo muerto al activar los pines de entrada PWM.



Registro PSTRCON

PSTRCON		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Características
		-	-	-	STRSYNC	STRD	STRC	STRB	STRA	Nombre de bit
					R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (1)	

Legenda

- Bit no implementado
- R/W Bit de lectura/escritura
- (0) Después del reinicio, el bit se pone a cero
- (1) Después del reinicio, el bit se pone a uno

STRSYNC - Steering Sync bit (bit de sincronización de dirección) determina el momento de la dirección de los pulsos de PWM:

- 1 - La dirección ocurre después de que el registro PSTRCION haya sido cambiado, sólo si se ha completado la forma de onda del PWM.
- 0 - La dirección ocurre después de que el registro PSTRCION haya sido cambiado. La señal PWM en la salida del pin será cambiada inmediatamente sin reparar en si el ciclo anterior ha sido completado. Este procedimiento es útil cuando es necesario detener la transmisión de una señal PWM del pin.

STRD - Steering Enable bit D (bit D de habilitación de dirección) determina la función del pin P1D.

- 1 - El pin P1D tiene la forma de onda del PWM con polaridad determinada por los bits CCP1M0 y CCP1M1.
- 0 - Pin está configurado como entrada/salida general del puerto PORTD.

STRC Steering Enable bit C (bit C de habilitación de dirección) determina la función del pin P1C.

- 1 - El pin P1C tiene la forma de onda del PWM con polaridad determinada por los bits CCP1M0 y CCP1M1.
- 0 - Pin está configurado como entrada/salida general del puerto PORTD.

STRB - Steering Enable bit B (bit B de habilitación de dirección) determina la función del pin P1B.

- 1 - El pin P1B tiene la forma de onda del PWM con polaridad determinada por los bits CCP1M0 y CCP1M1.
- 0 - Pin está configurado como entrada/salida general del puerto PORTD.

STRA - Steering Enable bit A (bit A de habilitación de dirección) determina la función del pin P1A.

- 1 - El pin P1A tiene la forma de onda del PWM con polaridad determinada por los bits CCP1M0 y CCP1M1.
- 0 - Pin está configurado como entrada/salida general del puerto PORTC.

Registro ECCPAS

	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
ECCPAS	ECCPASE	ECCPAS2	ECCPAS1	ECCPAS0	PSSAC1	PSSAC0	PSSBD1	PSSBD0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

R/W (0)	Bit de lectura/escritura Después del reinicio, el bit se pone a cero
---------	---

ECCPASE - ECCP Auto-Shutdown Event Status bit (bit de estado del apagado automático) indica si ha ocurrido el apagado automático del módulo CCP (estado de Apagado):

- 1 - Módulo CCP está en estado de Apagado.
- 0 - Módulo CCP funciona normalmente.

ECCPAS2 - ECCPAS0 - ECCP Auto-Shutdown Source Select bits (Bits de selección de la fuente de apagado automático) selecciona la fuente de apagado automático.

ECCPAS2 ECCPAS1 ECCPAS0 Fuente del estado de apagado

0	0	0	Estado del apagado deshabilitado
0	0	1	Cambio de salida del comparador C1
0	1	0	Cambio de salida del comparador C2
0	1	1	Cambio de salidas de los comparadores C1 y C2
1	0	0	Cero lógico (0) en el pin INT
1	0	1	Cero lógico (0) en el pin INT o cambio de salida del comparador C1
1	1	0	Cero lógico (0) en el pin INT o cambio de salida del comparador C2
1	1	1	Cero lógico (0) en el pin INT o cambio de salidas de los comparadores C1 y C2

PSSAC1, PSSAC0 - Pins P1A, P1C Shutdown State Control bits (Bits de configuración de los pines P1A y P1C en modo de apagado) define el estado lógico de los pines P1A y P1C cuando el módulo CCP está en el estado de apagado.

PSSAC1 PSSAC0 Estado lógico de los pines

0	0	0
0	1	1
1	X	Alta impedancia (Tri-estado)

PSSBD1, PSSBD0 - Pins P1B, P1D Shutdown State Control bits (Bits de configuración de los pines P1B y P1D en modo de apagado) define el estado lógico de los pines P1B y P1D cuando el módulo CCP está en el estado de apagado.

PSSBD1 PSSBD0 Estado lógico de los pines

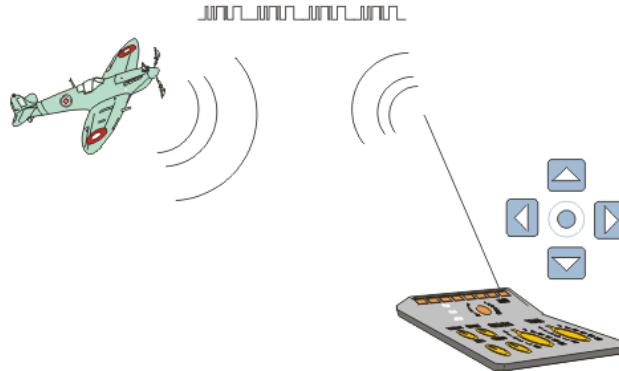
0	0	0
0	1	1
1	X	Alta impedancia (Tri-estado)

El microcontrolador PIC 16F887 dispone de varios módulos de comunicación serie independientes, además cada uno se puede configurar a funcionar en modos diferentes. Eso es lo que los hace insustituibles en muchos casos. Acuérdesse de lo que hemos dicho sobre los módulos CCP ya que lo mismo se aplica aquí. No se preocupe de los detalles del funcionamiento de todos los módulos, solo seleccione uno y utilice lo que realmente necesita.

MÓDULOS DE COMUNICACIÓN SERIE

El USART es uno de los primeros sistemas de comunicación serie. Las versiones nuevas de este sistema están actualizadas y se les denomina un poco diferente - EUSART.

EUSART



El módulo Transmisor/Receptor Universal Síncrono/Asíncrono mejorado (Enhanced Universal Synchronous Asynchronous Receiver Transmitter - EUSART) es un periférico de comunicación serie de entrada/salida. Asimismo es conocido como Interfaz de comunicación serie (Serial Communications Interface - SCI). Contiene todos los generadores de señales de reloj, registros de desplazamiento y búfers de datos necesarios para realizar transmisión de datos serie de entrada/salida, independientemente de la ejecución de programa del dispositivo. Como indica su nombre, aparte de utilizar el reloj para la sincronización, este módulo puede establecer la conexión asíncrona, lo que lo hace único para algunas aplicaciones. Por ejemplo, en caso de que sea difícil o imposible proporcionar canales especiales para transmisión y recepción de datos y señales de reloj (por ejemplo, mando a distancia de radio o infrarrojas), el módulo EUSART es definitivamente la mejor opción posible.

El EUSART integrado en el PIC16F887 posee las siguientes características:

- Transmisión y recepción asíncrona en modo *Full-duplex*;
- Caracteres de anchura de 8 – 9 bits programables;
- Detección de dirección en modo de 9 bits;
- Detección de errores por saturación del búfer de entrada; y
- Comunicación *Half Duplex* en modo síncrono.

EUSART EN MODO ASÍNCRONO

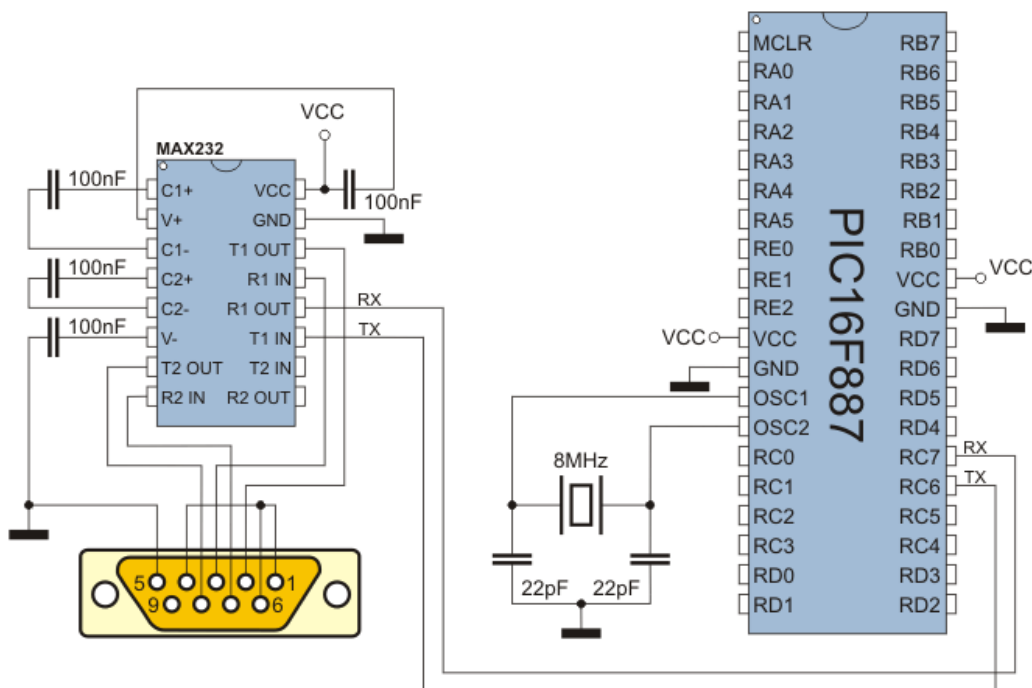
El EUSART transmite y recibe los datos utilizando la codificación de no retorno a cero - NRZ (non-return-to-zero). Como se muestra en la siguiente figura, no se utiliza una señal de reloj y los datos se transmiten de forma muy simple:



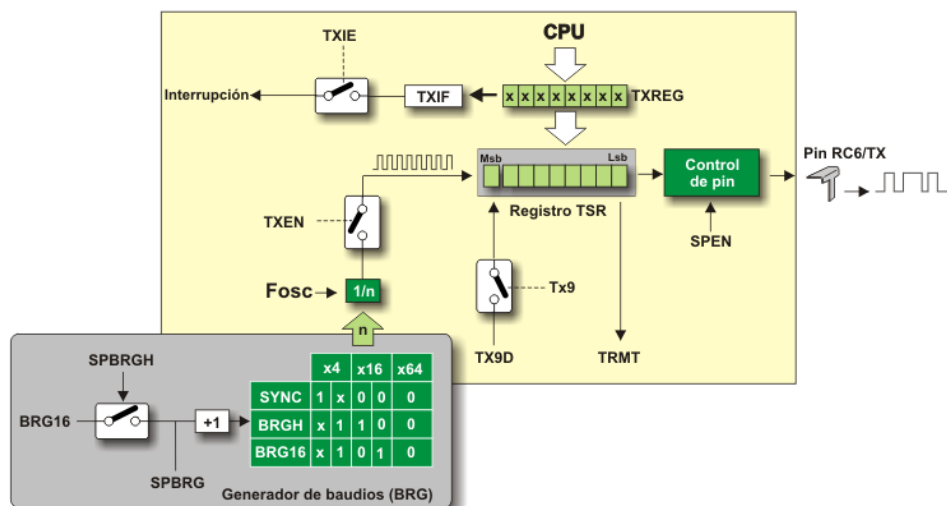
Cada dato se transmite de la siguiente forma:

- En estado inactivo la línea de datos permanece en estado alto (1);
- Cada transmisión de datos comienza con un bit de arranque (START), el cual, siempre es cero (0);
- Cada dato tiene un ancho de 8 o 9 bits (primero se transmite el bit menos significativo - LSB); y
- Cada transmisión de datos termina con un bit de parada (STOP), el cual, siempre es uno (1) La siguiente figura muestra cómo conectar de manera habitual un microcontrolador PIC que utiliza el módulo EUSART. El circuito RS-232 se utiliza como un convertidor de nivel de voltaje.

Figure below shows a common way of connecting PIC microcontroller that uses EUSART module. The RS-232 circuit is used as a voltage level converter.



EUSART EN MODO DE TRANSMISOR ASÍNCRONO



Para habilitar la transmisión de datos por medio del módulo EUSART, es necesario configurarlo para que funcione como un transmisor. En otras palabras, es necesario definir el estado de los siguientes bits:

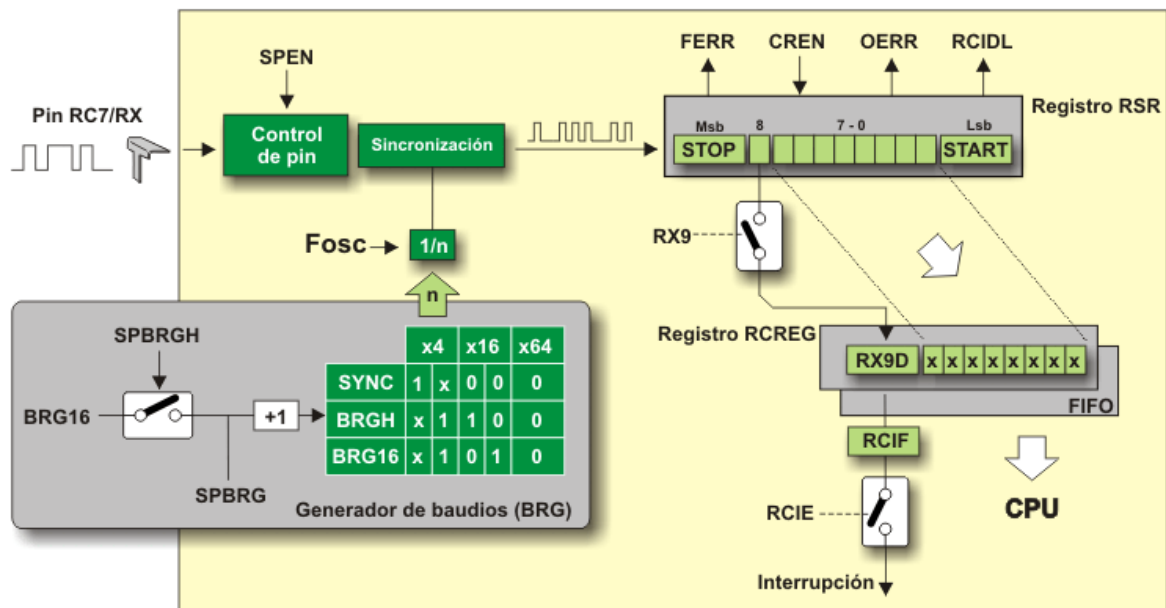
- **TXEN = 1** - El transmisor EUSART se habilita al poner a uno el bit TXEN del registro TXSTA.
- **SYNC = 0** - El EUSART se configura a funcionar en modo asíncrono al poner a cero el bit SYNC del registro TXSTA.
- **SPEN = 1** - Al poner a uno el bit SPEN del registro RCSTA, el EUSART está habilitado y el pin TX/CK se configura automáticamente como salida. Si el bit se utiliza simultáneamente para alguna función analógica, se debe deshabilitar al poner a cero el bit correspondiente del registro ANSEL.

La parte central del transmisor EUSART ocupa el registro de desplazamiento TSR que no está directamente disponible al usuario. Para iniciar la transmisión de datos, el módulo debe estar habilitado al poner a uno el bit TXEN del registro TXSTA. Los datos a enviar se deben escribir en el registro TXREG, lo que resultará en la siguiente secuencia de eventos:

- Byte será transmitido inmediatamente al registro de desplazamiento TSR;
- El registro TXREG permanece vacío, lo que indica la bandera de bit TXIF del registro PIR1. Si se pone a uno el bit TXIE del registro PIE1, se generará una interrupción. De todos modos, la bandera se pone a uno sin reparar en si una interrupción está habilitada o no y no se puede poner a cero por software, sino al escribir un dato nuevo en el registro TXREG.
- Dispositivos electrónicos de control "empujan" el dato hacia el pin TX en sincronización con señal de reloj interna: bit de arranque (START) (1).....datos....bit de parada (STOP) (1).
- Cuando el último bit abandona el registro TSR, el bit TRMT en el registro TXSTA se pone a cero automáticamente.
- Si mientras tanto se escribe un dato nuevo en el registro TXREG, todo el procedimiento se repite inmediatamente después de la transmisión del bit de parada del dato anterior.

Para transmitir un dato de 9 bits es necesario poner a uno el bit TX9 del registro TXSTA. El bit TX9D del registro TXSRTA es el noveno bit y el más significativo. Al transmitir un dato de 9 bits, el bit de datos TX9D deberá estar escrito antes de que se escriban los 8 bits menos significativos en el registro TXREG. Todos los nueve bits de datos se transmiten al registro de desplazamiento TFR inmediatamente después de que se acabe la escritura en el registro TXREG.

EUSART EN MODO DE RECEPTOR ASÍNCRONO



Similar al poner en marcha el transmisor del EUSART, para habilitar el receptor es necesario configurar los siguientes bits:

- **CREN = 1** - El receptor EUSART se habilita al poner a uno el bit CREN del registro RCSTA;
- **SYNC = 0** - El EUSART se configura a funcionar en modo asíncrono al poner a cero el bit SYNC del registro TXSTA; y
- **SPEN = 1** - Al poner a uno el bit SPEN del registro RCSTA, el EUSART está habilitado y el pin RX/DT se configura automáticamente como salida. Si el bit se utiliza simultáneamente para alguna función analógica, se debe deshabilitar al poner a cero el bit correspondiente del registro ANSEL.

Después de que se haya terminado el primer paso necesario y se haya detectado el bit de arranque (START), el dato se transmite al registro de desplazamiento RSR por el pin RX. Al haber recibido el bit de parada (STOP), ocurre lo siguiente:

- El dato se transmite automáticamente al registro RCREG (si está vacío);
 - El bit de bandera RCIF se pone a uno y ocurre una interrupción si está habilitada por el bit RCIE en el registro PIE1. Similar al transmisor, el bit de bandera se pone a cero sólo por software, o sea, al leer el registro RCREG. Tenga en cuenta que esto es un doble registro de tipo FIFO (primero en entrar, primero en salir - *first-in, first-out*), lo que permite almacenamiento de dos caracteres simultáneamente);
 - Si el registro RCREG está ocupado (contiene dos bytes) y el registro de desplazamiento detecta el nuevo bit de parada (STOP), el bit de sobrescritura OERR se pondrá a uno. En tal caso se pierde un dato nuevo que viene, y el bit OERR debe ponerse a cero por software al poner a cero y luego al poner a uno el bit CREN;
- Nota: No es posible recibir un dato nuevo sino hasta que el bit OERR esté a uno.**
- Si el bit de parada (STOP) está a cero (0), el bit FERR del registro RCSTA estará a uno, lo que indica un error en recepción; y

- Para habilitar la recepción de un dato de 9 bits, es necesario poner a uno el bit RX9 del registro RCSTA.

DETECCIÓN DE ERRORES EN RECEPCIÓN

El microcontrolador puede detectar automáticamente dos tipos de errores. El primero es denominado error de encuadre (*Framing error*). Ocurre cuando el receptor no detecta el bit de parada en un intervalo predeterminado de tiempo. Este error se indica mediante el bit FERR del registro RCSTA. Si el bit está a uno, el último dato recibido puede ser incorrecto. Cabe destacar lo siguiente:

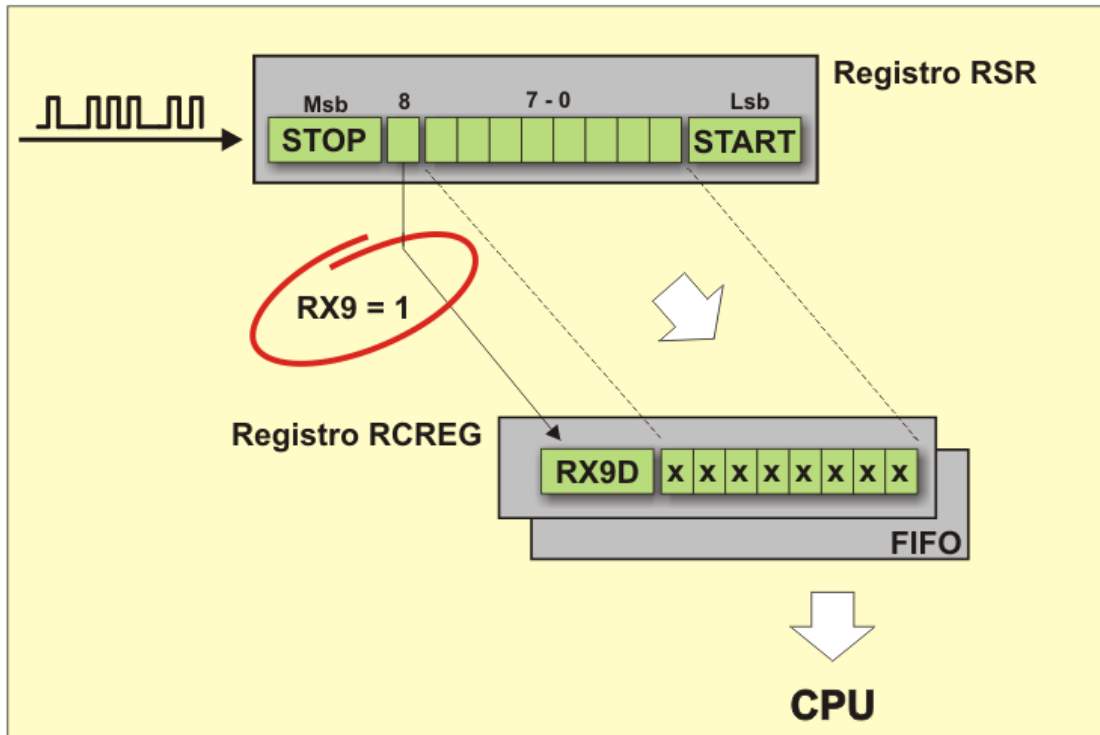
- El error de encuadre no genera por si mismo una interrupción;
- Si el bit está a uno, el último dato recibido contiene un error;
- El error de encuadre (bit está a uno) no impide la recepción de un dato nuevo;
- El bit FERR se pone a cero al leer el dato recibido, lo que significa que se debe hacer una verificación antes de leer el dato; y
- El bit FERR no se puede poner a cero por software. Si es necesario, se puede borrar al poner a cero al bit SPEN del registro RCSTA, lo cual, simultáneamente causa una reinicialización del sistema EUSART.

Otro tipo de error es denominado error de sobrescritura (Overrun Error). Como hemos mencionado anteriormente, el registro de tipo FIFO puede almacenar sólo dos caracteres. Un error de sobrescritura ocurre cuando el registro recibe el tercer carácter. Simplemente no hay espacio para almacenar un byte más, por lo que un error es inevitable. Cuando ocurre este error, el bit OERR del registro RCSTA se pone a uno. Las consecuencias son las siguientes:

- Los datos almacenados en los registros FIFO (2 bytes) se pueden leer normalmente;
- No se recibirán más datos hasta que el bit OERR esté puesto a cero; y
- A este bit no se le puede acceder directamente. Para borrarlo, es necesario poner a cero el bit CREN del registro RCSTA o reiniciar el sistema EUSART al poner a cero al bit SPEN del registro RCSTA.

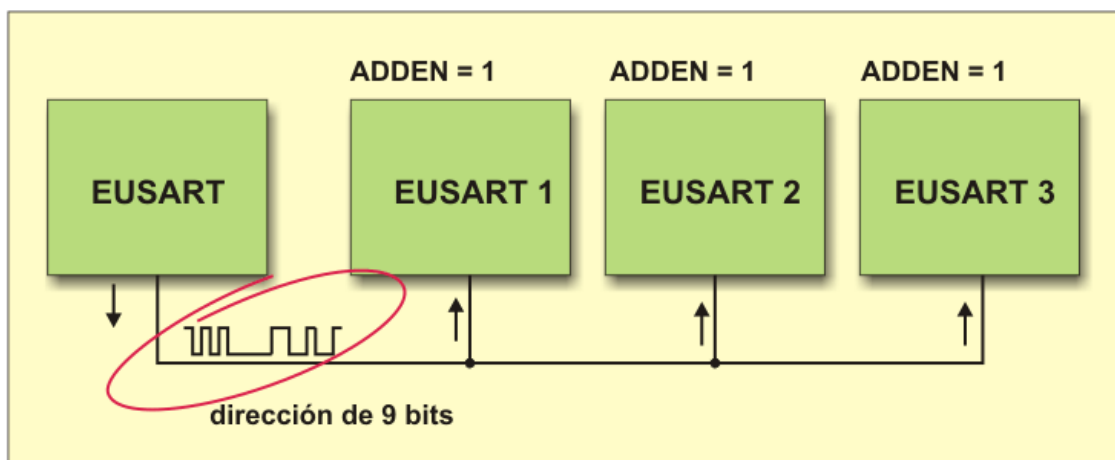
RECEPCIÓN DE DATOS DE 9 BITS

Aparte de recibir los datos de forma estándar de 8 bits, el sistema EUSART soporta la recepción de datos de 9 bits. En el lado del transmisor, el noveno bit “se adjunta” al byte original directamente antes del bit de parada. En el lado del receptor, al poner a uno el bit RX9 del registro RCSTA, el noveno bit de datos será automáticamente escrito en el bit RX9D del mismo registro. Después de almacenar este byte, es necesario tener cuidado en como leer estos bits - primero se debe leer el bit RX9D y luego los ocho (8) bits menos significativos del registro RCREG. De otra forma, el noveno bit será puesto a cero antes de ser leído.

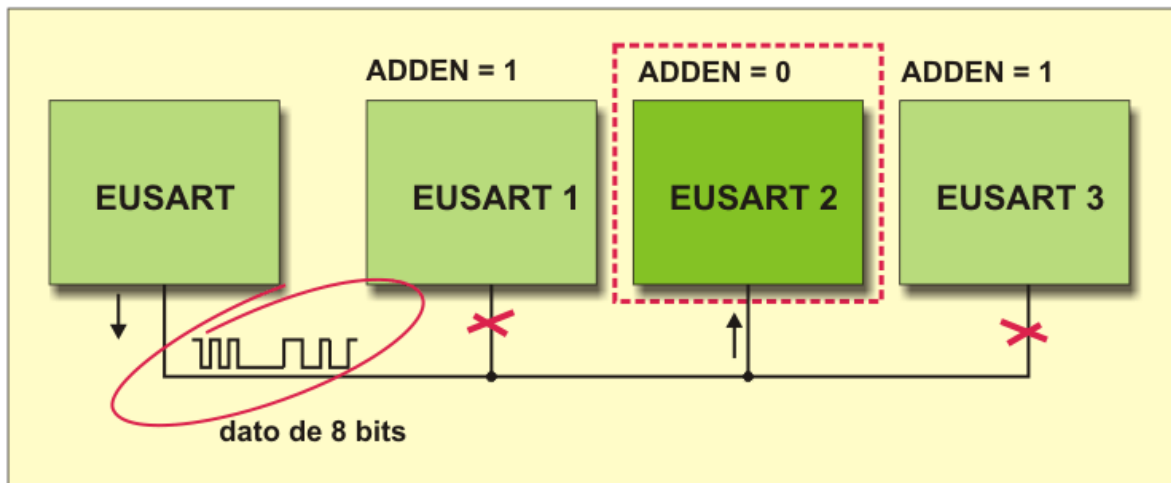


DETECCIÓN DE DIRECCIÓN

Cuando el bit ADDEN del registro RCSTA está a uno, el modulo EUSART es capaz de recibir sólo los datos de 9 bits, mientras que se ignoran todos los datos de 8 bits. Aunque parece una restricción, este modo habilita la comunicación serial entre varios microcontroladores. El principio de funcionamiento es muy simple. El dispositivo maestro envía un dato de 9 bits que representa la dirección de un microcontrolador esclavo. No obstante, todos deben tener el bit ADDEN puesto a uno, ya que de esta manera se habilita la detección de dirección. Todos los microcontroladores esclavos que comparten la misma línea de transmisión, reciben este dato (dirección) y verifican automáticamente si coincide con su propia dirección. El software, en el que ocurre la coincidencia de dirección, debe deshabilitar la detección de dirección, poniendo a cero el bit ADDEN.



El dispositivo maestro sigue enviando los datos de 8 bits al microcontrolador. Todos los datos que pasan por la línea de transmisión serán recibidos sólo por el módulo EUSART direccionado. Una vez recibido el último byte, el microcontrolador esclavo debe poner a uno el bit ADDEN para habilitar de nuevo la detección de dirección.



Registro TXSTA

TXSTA	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R (1)	R/W (0)	Características
	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit se pone a cero
(1)	Después del reinicio, el bit se pone a uno

CSRC - Clock Source Select bit - (bit de selección de la fuente de reloj) determina la fuente del reloj. Se utiliza sólo en modo síncrono.

- 1 - Modo *Maestro*. Reloj generado internamente por el generador de tasa de baudios.
- 0 - Modo *Esclavo*. Reloj proveniente de una fuente externa.

TX9 - 9-bit Transmit Enable bit (bit de habilitación del modo de 9 bits en transmisión)

- 1 - Se habilita el modo de 9 bits en transmisión por el sistema EUSART.
- 0 - Se habilita el modo de 8 bits en transmisión por el sistema EUSART.

TXEN - Transmit Enable bit (Bit de habilitación de transmisión)

- 1 - Transmisión habilitada.
- 0 - Transmisión deshabilitada.

SYNC - EUSART Mode Select bit (Bit de selección del modo EUSART)

- 1 - El EUSART funciona en modo síncrono.
- 0 - El EUSART funciona en modo asíncrono.

SENDB - Send Break Character bit (Bit de envío de carácter Break en modo asíncrono) se utiliza sólo en modo asíncrono y cuando se requiere obedecer el estándar de bus LIN.

- 1 - Se enviará un carácter *Break* en la próxima transmisión (se pone a 0 por hardware cuando finaliza el envío).
- 0 - Envío del carácter de transmisión *Break* completado.

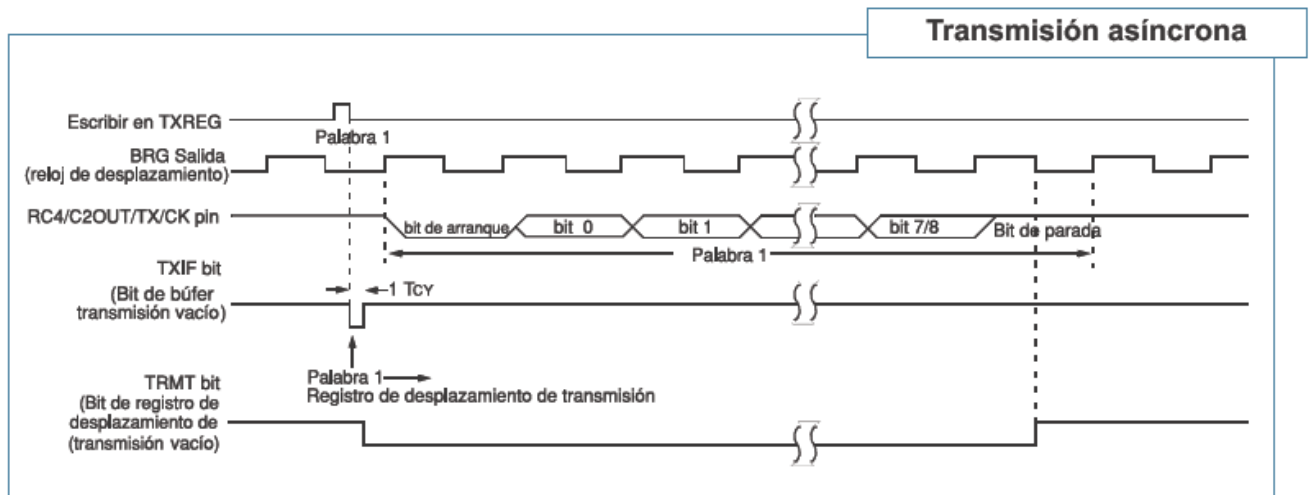
BRGH - High Baud Rate Select bit (bit de selección de modo de alta velocidad en modo asíncrono). Determina la velocidad de transmisión en modo síncrono. No afecta al EUSART en modo síncrono.

- 1 - EUSART funciona a alta velocidad.
- 0 - EUSART funciona a baja velocidad.

TRMT - Transmit Shift Register Status bit (bit de estado de registro de desplazamiento de transmisión)

- 1 - Registro TSR está vacío.
- 0 - Registro TSR está lleno.

TX9D - Ninth bit of Transmit Data (Valor del noveno bit en transmisión) Puede ser utilizado como dirección o bit de paridad o para distinguir entre dirección o dato en los buses maestro-esclavo).



Registro RCSTA

RCSTA	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R (0)	R (0)	R (x)	Características
	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit se pone a cero
(x)	Después del reinicio, el estado de bit es desconocido

SPEN - Serial Port Enable bit (bit de habilitación del puerto serie)

- 1 - Puerto serie habilitado. Los pines RX/DT y TX/CK se configuran automáticamente como entrada y salida, respectivamente.
- 0 - Puerto serie deshabilitado.

RX9 - (bit de habilitación del modo de 9 bits en recepción):

- 1 - Se habilita la recepción de datos de 9 bits por medio del sistema EUSART.
- 0 - Se habilita la recepción de datos de 8 bits por medio del sistema EUSART.

SREN - Single Receive Enable bit (bit de habilitación de la recepción simple). Es utilizado sólo en modo síncrono y en funcionamiento como Maestro.

- 1 - Recepción simple habilitada.
- 0 - Recepción simple deshabilitada.

CREN - Continuous Receive Enable bit (bit de habilitación de la recepción continua) actúa dependiendo del modo EUSART.

Modo asíncrono:

- 1 - Recepción habilitada.
- 0 - Recepción deshabilitada.

Modo síncrono:

- 1 - Se habilita la recepción continua hasta que el bit CREN esté a cero.
- 0 - No se habilita la recepción en forma continua.

ADDEN - Address Detect Enable bit (bit de habilitación de la detección de dirección) se utiliza sólo en modo de detectar la dirección.

- 1 - Habilita la detección de dirección (sólo se procesa un byte recibido en el registro de desplazamiento de recepción si el noveno bit está a uno)
- 0 - Detección de dirección deshabilitada (todos los bytes recibidos en el registro de desplazamiento de recepción son procesados independientemente del valor del noveno bit recibido). El noveno bit se utiliza como bit de paridad.

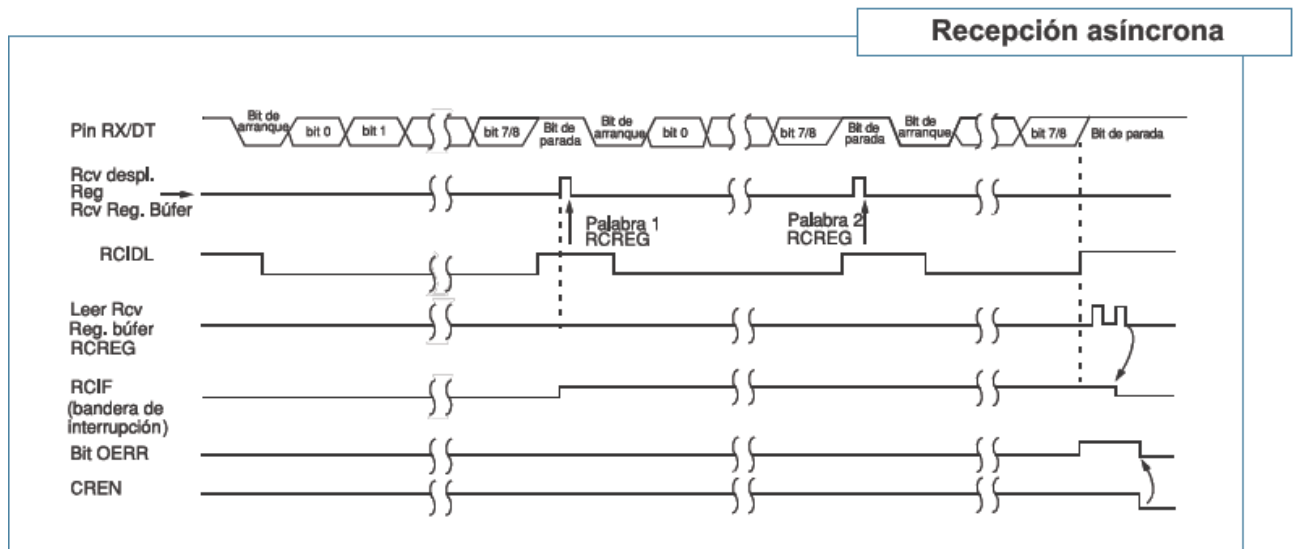
FERR - Framing Error bit (bit de error de encuadre)

- 1 - Se ha producido un error de encuadre en recepción.
- 0 - No se ha producido un error de encuadre.

OERR - Overrun Error bit (bit de error de sobrescritura).

- 1 - Se ha producido un error de sobrescritura en recepción.
- 0 - No se ha producido un error de sobrescritura.

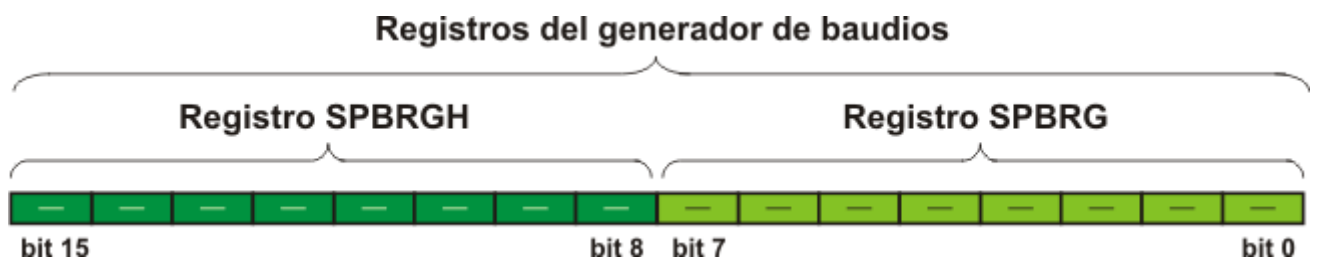
RX9D - Ninth bit of Received Data No se ha producido un error de sobrescritura.



GENERADOR DE BAUDIOS DEL EUSART (BRG)

Si mira atentamente al diagrama del receptor o transmisor EUSART asíncrono, verá que los ambos utilizan señal de reloj del temporizador local BRG para la sincronización. La misma fuente de reloj se utiliza también en modo síncrono.

El temporizador BRG consiste en dos registros de 8 bits haciendo un registro de 16 bits.



El valor de un número escrito en estos dos registros determinará la velocidad de transmisión en baudios. Adicionalmente, el bit BRGH del registro TXSTA y el bit BRGH16 del registro BAUDCTL, afectan la frecuencia de reloj utilizada para el cálculo de los baudios.

El valor de un número escrito en estos dos registros determinará la velocidad de transmisión en baudios. Adicionalmente, el bit BRGH del registro TXSTA y el bit BRGH16 del registro BAUDCTL, afectan la frecuencia de reloj utilizada para el cálculo de los baudios.

Bits			Modo BRG / EUSART	Fórmula de velocidad de transmisión en baudios
SYNC	BRG1G	BRGH		
0	0	0	de 8 bits /asíncrono	$F_{osc} / [64 (n + 1)]$
0	0	1	de 8 bits / asíncrono	$F_{osc} / [16 (n + 1)]$
0	1	0	de 16 bits / asíncrono	$F_{osc} / [16 (n + 1)]$
0	1	1	de 16 bits / asíncrono	$F_{osc} / [4 (n + 1)]$
1	0	X	de 8 bits / síncrono	$F_{osc} / [4 (n + 1)]$
1	1	X	de 16 bits / síncrono	$F_{osc} / [4 (n + 1)]$

Las tablas en las siguientes páginas contienen los valores que deben estar escritos en el registro de 16 bits SPBRG y asignados a los bits SYNC, BRGH y BRGH16 para obtener algunos valores de la velocidad de transmisión en baudios estándar. La fórmula para hacer el cálculo de la velocidad de transmisión en baudios:

$$Velocidad\ de\ transmisión\ en\ baudios\ deseada = \frac{F_{osc}}{64(SPBRGH:SPBRG - 1)}$$

$$SPBRGH:SPBRG = \frac{F_{osc}}{Velocidad\ de\ transmisión\ en\ baudios\ deseada} - 1$$

$$Error\ [%] = \frac{Velocidad\ de\ transmisión\ en\ baudios\ calculada - Velocidad\ de\ transmisión\ en\ baudios\ deseada}{Velocidad\ de\ transmisión\ en\ baudios\ deseada}$$

Velocidad de transmisión en baudios	SYNC = 0, BRGH = 0, BRG16 = 0											
	Fosc = 20 MHz			Fosc = 18.432 MHz			Fosc = 11.0592 MHz			Fosc = 11.0592 MHz		
	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)
300	-	-	-	-	-	-	-	-	-	-	-	-
1200	1221	1.73	255	1200	0.00	239	1200	0.00	143	1202	0.16	103
2400	2404	0.16	129	2400	0.00	119	2400	0.00	71	2404	0.16	51
9600	9470	-1.36	32	9600	0.00	29	9600	0.00	17	9615	0.16	12
10417	10417	0.00	29	10286	-1.26	27	10165	-2.42	16	10417	0.00	11
19.2k	19.53	1.73	15	19.2	0.00	14	19.2	0.00	8	-	-	-
57.6k	-	-	-	57.6k	0.00	7	57.6	0.00	2	-	-	-
115.2k	-	-	-	-	-	-	-	-	-	-	-	-

Velocidad de transmisión en baudios	SYNC = 0, BRGH = 0, BRG16 = 0											
	Fosc = 4 MHz			Fosc = 3.6864 MHz			Fosc = 2 MHz			Fosc = 1 MHz		
	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)
300	300	0.16	207	300	0.00	191	300	0.16	103	300	0.16	51
1200	1202	0.16	51	1200	0.00	47	1202	0.16	25	1202	0.16	12
2400	2404	0.16	25	2400	0.00	23	2404	0.16	12	-	-	-
9600	-	-	-	9600	0.00	5	-	-	-	-	-	-
10417	10417	0.00	5	-	-	-	10417	0.00	2	-	-	-
19.2k	-	-	-	19.2	0.00	2	-	-	-	-	-	-
57.6k	-	-	-	57.6k	0.00	0	-	-	-	-	-	-
115.2k	-	-	-	-	-	-	-	-	-	-	-	-

Velocidad de transmisión en baudios	SYNC = 0, BRGH = 1, BRG16 = 0											
	Fosc = 20 MHz			Fosc = 18.432 MHz			Fosc = 11.0592 MHz			Fosc = 8 MHz		
	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)
300	-	-	-	-	-	-	-	-	-	-	-	-
1200	-	-	-	-	-	-	-	-	-	-	-	-
2400	-	-	-	-	-	-	-	-	-	2404	0.16	207
9600	9615	0.16	129	9600	0.00	119	9600	0.00	71	9615	0.16	51
10417	10417	0.00	119	10378	-0.37	110	10473	0.53	65	10417	0.00	47
19.2k	19.23k	0.16	64	19.2	0.00	59	19.2k	0.00	35	19231	0.16	25
57.6k	56.82k	-1.36	21	57.6k	0.00	19	57.6k	0.00	11	55556	-3.55	8
115.2k	113.64k	-1.36	10	115.2k	0.00	9	115.2k	0.00	5	-	-	-

Velocidad de transmisión en baudios	SYNC = 0, BRGH = 1, BRG16 = 0											
	Fosc = 4 MHz			Fosc = 3.6864 MHz			Fosc = 2 MHz			Fosc = 1 MHz		
	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)
300	-	-	-	-	-	-	-	-	-	300	0.16	207
1200	1202	0.16	207	1200	0.00	191	1202	0.16	103	1202	0.16	51
2400	2404	0.16	103	2400	0.00	95	2404	0.16	51	2404	0.16	25
9600	9615	0.16	25	9600	0.00	23	9615	0.16	12	-	-	-
10417	10417	0.00	23	10473	0.00	11	10417	0.00	11	10417	0.00	5
19.2k	19.23k	0.16	12	19.2	0.00	11	-	-	-	-	-	-
57.6k	-	-	-	57.6k	0.00	3	-	-	-	-	-	-
115.2k	-	-	-	115.2k	0.00	1	-	-	-	-	-	-

Velocidad de transmisión en baudios	SYNC = 0, BRGH = 0, BRG16 = 1											
	Fosc = 20 MHz			Fosc = 18.432 MHz			Fosc = 11.0592 MHz			Fosc = 8 MHz		
	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)
300	300	-0.01	4166	300	0.00	3839	300	0.00	2303	299.9	-0.02	1666
1200	1200	-0.03	1041	1200	0.00	959	1200	0.00	575	1199	-0.08	416
2400	2399	-0.03	520	2400	0.00	479	2400	0.00	287	2404	0.16	207
9600	9615	0.16	129	9600	0.00	119	9600	0.00	71	9615	0.16	51
10417	10417	0.00	119	10378	-0.37	110	10473	0.53	65	10417	0.00	47
19.2k	19.23k	0.16	64	19.2k	0.00	59	19.2k	0.00	35	19.23k	0.16	25
57.6k	56.818	-1.36	21	57.6k	0.00	19	57.6k	0.00	11	55556	-3.55	8
115.2k	113.636	-1.36	10	115.2k	0.00	9	115.2k	0.00	5	-	-	-

Velocidad de transmisión en baudios	SYNC = 0, BRGH = 0, BRG16 = 1											
	Fosc = 4 MHz			Fosc = 3.6864 MHz			Fosc = 2 MHz			Fosc = 1 MHz		
	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)
300	300.1	0.04	832	300	0.00	767	299.8	-0.108	416	300.5	0.16	207
1200	1202	0.16	207	1200	0.00	191	1202	0.16	103	1202	0.16	51
2400	2404	0.16	103	2400	0.00	95	2404	0.16	51	2404	0.16	25
9600	9615	0.16	25	9600	0.00	23	9615	0.16	12	-	-	-
10417	10417	0.00	23	10473	0.53	21	10417	0.00	11	10417	0.00	5
19.2k	19.23k	0.16	12	19.2k	0.00	11	-	-	-	-	-	-
57.6k	-	-	-	57.6	0.00	3	-	-	-	-	-	-
115.2k	-	-	-	115.2k	0.00	1	-	-	-	-	-	-

Velocidad de transmisión en baudios	SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRGH16 = 1											
	Fosc = 20 MHz			Fosc = 18.432 MHz			Fosc = 11.0592 MHz			Fosc = 8 MHz		
	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)
300	300	0.00	16665	300	0.00	15359	300	0.00	9215	300	0.00	6666
1200	1200	-0.01	4166	1200	0.00	3839	1200	0.00	2303	1200	-0.02	1666
2400	2400	0.02	2082	2400	0.00	1919	2400	0.00	1151	2401	0.04	832
9600	9597	-0.03	520	9600	0.00	479	9600	0.00	287	9615	0.16	207
10417	10417	0.00	479	10425	0.08	441	10433	0.16	264	10417	0	191
19.2k	19.23k	0.16	259	19.2k	0.00	239	19.2k	0.00	143	19.23k	0.16	103
57.6k	57.47k	-0.22	86	57.6k	0.00	79	57.6k	0.00	47	57.14k	-0.79	34
115.2k	116.3k	0.95	42	115.2k	0.00	39	115.2k	0.00	23	117.6k	2.12	16

Velocidad de transmisión en baudios	SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRGH16 = 1											
	Fosc = 4 MHz			Fosc = 3.6864 MHz			Fosc = 2 MHz			Fosc = 1 MHz		
	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)	Velocidad de transmisión actual	Error %	SPBRG valor (decimal)
300	300	0.01	3332	300	0.00	3071	299.9	-0.02	1666	300.1	0.04	832
1200	1200	0.04	832	1200	0.00	767	1199	-0.08	416	1202	0.16	207
2400	2398	0.08	416	2400	0.00	383	2404	0.16	207	2404	0.16	103
9600	9615	0.16	103	9600	0.00	96	9615	0.16	51	9615	0.16	25
10417	10417	0.00	95	10473	0.53	87	10417	0.00	47	10417	0.00	23
19.2k	19.23k	0.16	51	19.2k	0.00	47	19.23k	0.16	25	19.23k	0.16	12
57.6k	58.82k	2.12	16	57.6k	0.00	15	55.56k	-3.55	8	-	-	-
115.2k	111.1k	-3.55	8	115.2k	0.00	7	-	-	-	-	-	-

Registro BAUDCTL

	R (0)	R (1)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
BAUDCTL	ABDOVF	RCIDL	-	SCKP	BRG16	-	WUE	ABDEN
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
								Nombre de bit

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit está a cero
(1)	Después de reinicio, el bit está a uno

ABDOVF - Auto-Baud Detect Overflow bit (bit de desbordamiento de auto-detección de la velocidad de transmisión) se utiliza sólo en modo asíncrono durante la detección de la velocidad de transmisión.

- 1 - Se ha producido desbordamiento durante la auto-detección.
- 0 - No se ha producido desbordamiento durante la auto-detección.

RCIDL - Receive Idle Flag bit No se ha producido desbordamiento durante la auto-detección.

- 1 - Receptor en estado inactivo. No hay operación de recepción en marcha.
- 0 - Se ha recibido el bit de arranque (START) y hay una operación de recepción en marcha.

SCKP - Synchronous Clock Polarity Select bit. (bit de selección de polaridad de la señal de reloj en modo síncrono). El estado lógico de este bit difiere dependiendo de cuál modo de EUSART está activo

Modo asíncrono:

- 1 - El dato invertido se transmite al pin RC6/TX/CK.
- 0 - El dato no invertido se transmite al pin RC6/TX/CK.

Modo síncrono:

- 1 - Sincronización en el flanco ascendente de la señal de reloj.
- 0 - Sincronización en el flanco descendente de la señal de reloj.

BRG16 16-bit Baud Rate Generator bit - (bit de habilitación del generador de velocidad de transmisión de 16 bits) determina si el registro SPBRGH se utilizará, o sea si el temporizador BGRG tendrá 8 o 16 bits.

- 1 - Se utiliza el generador de velocidad de transmisión de 16 bits
- 0 - Se utiliza el generador de velocidad de transmisión de 8 bits

WUE Wake-up Enable bit (bit de habilitación del modo de auto-activación en modo asíncrono):

- 1 - Modo de auto-activación habilitado. El receptor espera a que el flanco descendente aparezca en el pin RC7/RX/DT para que el microcontrolador se despierte del modo de reposo.
- 0 - Modo de auto-activación habilitado. El receptor funciona normalmente.

ABDEN - Auto-Baud Detect Enable bit (bit de habilitación de auto-detección de velocidad de transmisión) se utiliza sólo en modo asíncrono.

- 1 - Modo de auto-detección habilitado. Al detectar la velocidad de transmisión, el bit se pone a uno automáticamente.
- 0 - Modo de auto-detección deshabilitado.

Vamos a hacerlo en mikroC...

/ En este ejemplo, el módulo EUSART interno se inicializa y se ajusta para enviar el mensaje inmediatamente después de recibirlo. La velocidad de transmisión en baudios se ajusta a 9600 bps. El programa utiliza las siguientes rutinas de librería UART: UART1_init(), UART1_Write_Text(), UART1_Data_Ready(), UART1_Write() y UART1_Read().*/*

```
char uart_rd;
```

```
void main() {
    ANSEL = ANSELH = 0;           // Todos los pines se configuran como digitales
    C1ON_bit = C2ON_bit = 0;     // Deshabilitar los comparadores
    UART1_Init(9600);            // Inicializar el módulo UART a 9600 bps
    Delay_ms(100);                // Esperar a que señal de reloj del módulo UART se
                                // ponga estable
    UART1_Write_Text("Start");
    while (1) {                  // Bucle infinito
        if (UART1_Data_Ready()) { // Si el dato se ha recibido,
            uart_rd = UART1_Read(); // lea el dato recibido
            UART1_Write(uart_rd);   // y envíelo atrás por el UART
        }
    }
}
```

Transmisión serial asíncrona a través de los registros del módulo EUSART

1. La velocidad de transmisión deseada deberá estar ajustada a través de los bits BRGH (del registro TXSTA) y BRG16 (del registro BAUDCTL) y de los registros SPBRGH y SPBRG.
2. La velocidad de transmisión deseada deberá estar ajustada a través de los bits BRGH (del registro TXSTA) y BRG16 (del registro BAUDCTL) y de los registros SPBRGH y SPBRG.
3. La velocidad de transmisión deseada deberá estar ajustada a través de los bits BRGH (del registro TXSTA) y BRG16 (del registro BAUDCTL) y de los registros SPBRGH y SPBRG.
4. La transmisión de datos es habilitada poniendo a uno el bit TXEN del registro TXSTA. El bit TXIF del registro PIR1 está automáticamente puesto a uno.
5. Para que el bit TXEN cause una interrupción, tanto el bit TXIE del registro PIE1 como los bits GIE, PEIE del registro INTCON deberán estar puestos a uno.
6. En una transmisión de datos de 9 bits, el valor del noveno bit deberá estar escrito en el bit TX9D del registro TXSTA.
7. La transmisión comienza cuando se escribe el dato de 8 bits sobre el registro de recepción TXREG.

Recepción serial asíncrona a través de los registros del módulo EUSART:

1. La velocidad de transmisión deseada deberá estar ajustada a través de los bits BRGH (del registro TXSTA) y BRG16 (del registro BAUDCTL) y de los registros SPBRGH y SPBRG.
2. El bit SYNC (del registro TXSTA) deberá estar puesto a cero y el bit SPEN (del registro RCSTA) deberá estar puesto a uno a fin de habilitar el puerto serie.
3. Tanto el bit RCIE del registro PIE1 como los bits GIE y PEIE del registro INTCON deberán estar puestos a uno si se necesita habilitar que la recepción de dato cause una interrupción.
4. Para una recepción de datos de 9 bits, el bit RX9 (del registro RCSTA) deberá estar puesto a uno.
5. La recepción de datos es habilitada poniendo a uno el bit CREN del registro RCSTA.
6. El registro RCSTA deberá leerse para obtener información acerca de la ocurrencia de errores durante la recepción. El valor del noveno bit será almacenado en este registro en la recepción de datos de 9 bits.
7. El dato de 8 bits recibido será almacenado en el registro RCREG y deberá leerse para obtener dicho dato.

Ajustar el modo de detección de dirección:

1. La velocidad de transmisión deseada deberá estar ajustada a través de los bits BRGH (del registro TXSTA) y BRG16 (del registro BAUDCTL) y de los registros SPBRGH y SPBRG.
2. El bit SYNC (del registro TXSTA) deberá estar puesto a cero y el bit SPEN (del registro RCSTA) deberá estar puesto a uno (1) a fin de habilitar el puerto serie.
3. Tanto el bit RCIE del registro PIE1 como los bits GIE y PEIE del registro INTCON deberán estar puestos a uno si se necesita habilitar que la recepción de dato cause una interrupción.
4. El bit RX9 del registro RCSTA debe estar a uno.

5. El bit ADDEN del registro RCSTA debe estar a uno, lo que habilita que un dato sea reconocido como dirección.
6. La recepción de datos es habilitada poniendo a uno el bit CREN del registro RCSTA.
7. Tan pronto como se reciba un dato de 9 bits, el bit RCIF del registro PIR1 estará automáticamente puesto a uno. Si está habilitada se produce una interrupción.
8. El registro RCSTA deberá leerse para obtener información acerca de la ocurrencia de errores durante la transmisión. El noveno bit RX9D siempre estará a uno.
9. El dato de 8 bits recibido será almacenado en el registro RCREG y deberá leerse. Se deberá comprobar si la combinación de estos bits coincide con la dirección predefinida. Si coincide, es necesario poner a cero el bit ADDEN del registro RCSTA, lo que habilita la recepción de datos de 8 bits.

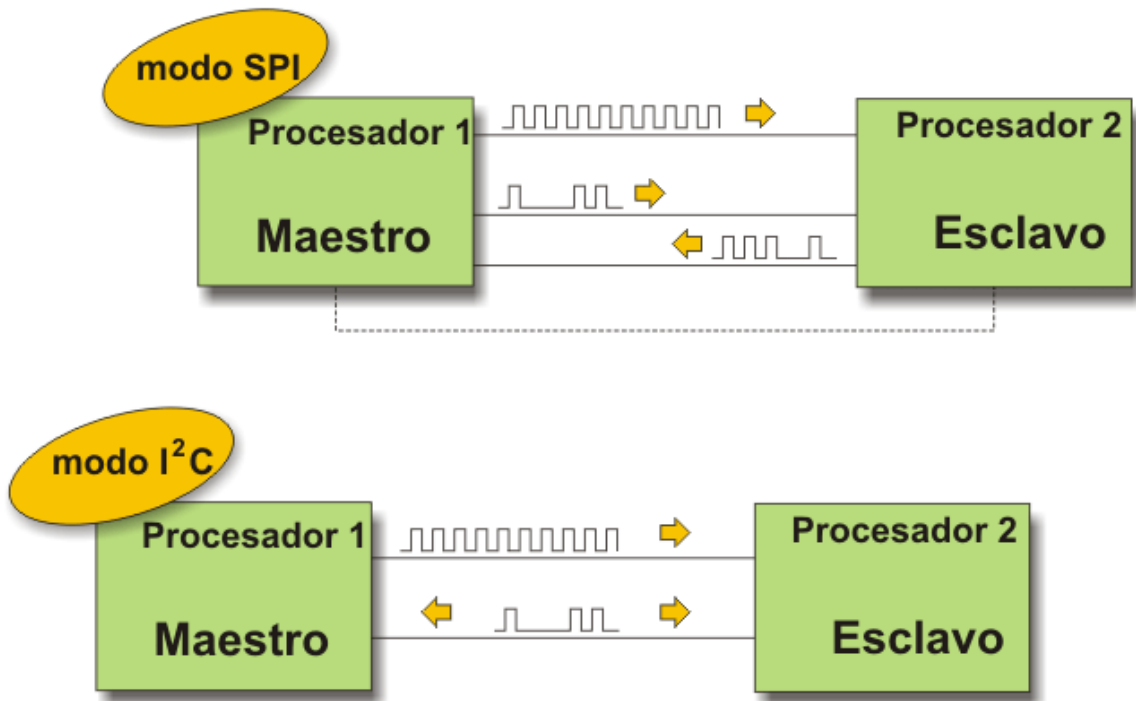
MÓDULO PUERTO SERIE SÍNCRONO MAESTRO (MSSP)

El MSSP (Puerto serie síncrono maestro - *Master Synchronous Serial Port*) es un módulo muy útil, y a la vez uno de los circuitos más complejos dentro del microcontrolador. Este módulo permite la comunicación de alta velocidad entre un microcontrolador y otros periféricos u otros microcontroladores al utilizar varias líneas de E/S (como máximo dos o tres líneas). Por eso, se utiliza con frecuencia para conectar el microcontrolador a los visualizadores LCD, los convertidores A/D, las memorias EEPROM seriales, los registros de desplazamiento etc. La característica principal de este tipo de comunicación es que es síncrona y adecuada para ser utilizada en sistemas con un sólo maestro y uno o más esclavos. Un dispositivo maestro contiene un circuito para generación de baudios y además, suministra señales de reloj a todos los dispositivos del sistema. Los dispositivos esclavos no disponen de un circuito interno para generación de señales de reloj. El módulo MSSP puede funcionar en uno de dos modos:

- modo SPI (Interfaz periférica serial - *Serial Peripheral Interface*); y
- modo I2C (Circuito inter-integrado - *Inter-Integrated Circuit*).

Como se muestra en la siguiente figura, un módulo MSSP representa sólo una mitad de un hardware necesario para establecer una comunicación serial, mientras que la otra mitad se almacena en el dispositivo con el que intercambia los datos. Aunque los módulos en ambas puntas de línea son los mismos, sus modos de funcionamiento difieren esencialmente dependiendo de si el módulo funciona como *Maestro* o como *Esclavo*:

Si el microcontrolador a ser programado controla otro dispositivo o circuito (periféricos), deberá funcionar como un dispositivo *maestro*. Este módulo generará señal de reloj cuando sea necesario, o sea sólo cuando se requiera recibir y transmitir los datos por software. Por consiguiente, el establecimiento de conexión depende únicamente del dispositivo maestro.



De lo contrario, si el microcontrolador a ser programado está integrado en un dispositivo más complejo (por ejemplo en una PC), deberá funcionar como un dispositivo esclavo. Como tal, un esclavo siempre tiene que esperar a que un dispositivo maestro envíe la solicitud de transmisión de datos.

MODO SPI

El modo SPI permite la transmisión y recepción simultánea de datos de 8 bits al utilizar tres líneas de entrada/salida

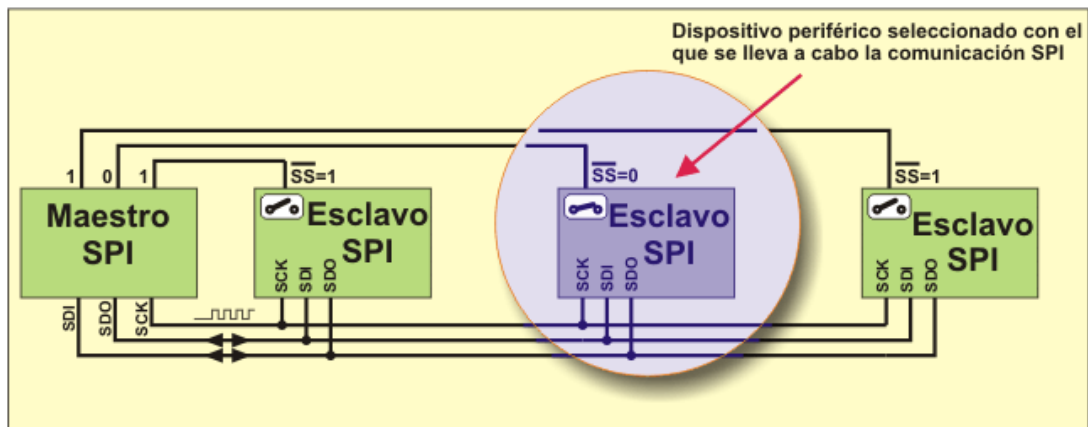
- **SDO** - *Serial Data Out* (salida de datos serie) - línea de transmisión;
- **SDI** - *Serial Data In* (entrada de datos serie) - línea de recepción; y
- **SCK** - *Serial Clock* (reloj de comunicación) - línea de sincronización.

Adicionalmente, hay una cuarta línea (SS) que se puede utilizar si el microcontrolador intercambia los datos con varios dispositivos periféricos. Refiérase a la siguiente figura.

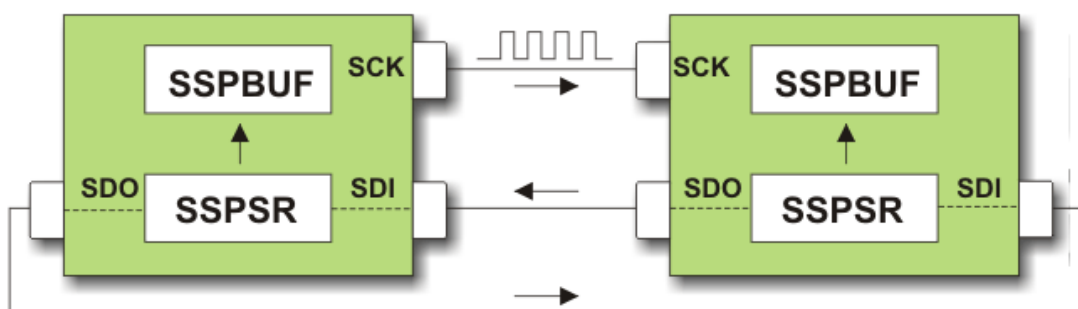
SS - *Slave Select (Selección de esclavo)* - Es una línea adicional utilizada para la selección de un dispositivo específico. Esta línea está activa sólo si el microcontrolador funciona como esclavo, o sea cuando el dispositivo externo - maestro requiere intercambiar los datos. Al funcionar en modo SPI, el módulo MSSP utiliza 4 registros en total:

- SSPSTAT - registro de estado
- SSPCON - registro de control
- SSPBUF - búfer serie de transmisión/recepción
- SSPSR - registro de desplazamiento (no es accesible directamente)

Los primeros tres registros son de lectura/escritura y se pueden modificar en cualquier momento, mientras que el cuarto, como no es accesible, se utiliza para convertir datos en formato serial.



Como se muestra en la siguiente figura, la parte central del módulo SPI consiste de dos registros conectados a los pines para recepción, transmisión y sincronización.



El registro de desplazamiento (SSPRS) está directamente conectado a los pines del microcontrolador y es utilizado para transmisión de datos en formato serie. El registro SSPRS dispone de la entrada y salida para desplazar los datos hacia dentro y hacia fuera del dispositivo. En otras palabras, cada bit que aparece en la entrada (línea de recepción) desplaza simultáneamente otro bit hacia la salida (línea de transmisión).

El registro SSPBUF (*Búfer*) es una parte de memoria utilizada para almacenar temporalmente los datos antes de que se envíen, o sea inmediatamente después de que se reciban. Después de que todos los 8 bits hayan sido recibidos, el byte se mueve del registro SSPRS al registro SSPBUF. Este proceso de crear un doble búfer para recibir los datos permite iniciar la recepción del próximo byte antes de leer los datos que se acaban de recibir. Durante la transmisión/recepción de datos se ignora un intento de escribir un dato en el registro SSPBUF. Desde el punto de vista de un programador, este registro se considera el más importante por haber sido accedido con más frecuencia. Concretamente, si dejamos aparte el ajuste del modo de funcionamiento, la transmisión de datos por el módulo SPI no es nada más que escritura y lectura de datos de este registro, mientras que las demás "acrobacias" como mover los registros, se llevan a cabo automáticamente por el hardware.

Vamos a hacerlo en mikroC...

/ En este ejemplo, el microcontrolador PIC (maestro) envía un byte de datos a un chip periférico (esclavo) por el módulo SPI. El programa utiliza las funciones de librería SPI SPI1_init() y SPI1_Write. */*

```
sbit Chip_Select at RC0_bit;           // Pin RC0 es un pin de seleccionar el chip
// periférico Selección_de_chip
```

```
sbit Chip_Select_Direction at TRISC0_bit; // Bit TRISC0 define el pin RC0 como entrada o salida
unsigned int value; // Dato a ser enviado es de tipo unsigned int
```

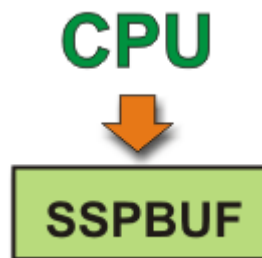
```
void main() {
  ANSEL = ANSELH = 0; // Todos los pines de E/S son digitales
  TRISB0_bit = TRISB1_bit = 1; // Configurar los pines RB0, RB1 como entradas
  Chip_Select = 0; // Seleccionar el chip periférico
  Chip_Select_Direction = 0; // Configurar el pin CS# como salida
  SPI1_Init(); // Inicializar el módulo SPI
  SPI1_Write(value); // Enviar el valor al chip periférico
  ...
}
```

Comunicación serial síncrona SPI

Antes de inicializar el módulo SPI, es necesario especificar varias opciones:

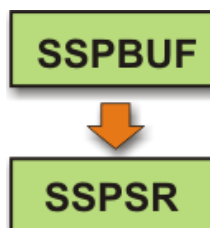
- Modo maestro TRISC.3=0 (pin SCK es salida de señal de reloj);
- Modo de esclavo TRISC.3=1 (pin SCK es entrada de señal de reloj);
- Fase de datos de entrada - la mitad o el final del tiempo de salida (bit SMP del registro SSPSTAT);
- Flanco de reloj (bit CKE del registro SSPSTAT);
- Velocidad de transmisión en baudios, los bits SSPM3-SSPM0 del registro SSPCON (sólo en modo Maestro);
- Selección de modo esclavo, bits SSPM3-SSPM0 del registro SSPCON (sólo en modo Esclavo)

El módulo se pone en marcha al poner a uno el bit SSPEN:



Paso 1.

Los datos a ser transmitidos deberán ser escritos en el registro del búfer SSPBUF. Si el módulo SPI funciona en modo maestro, el microcontrolador ejecutará automáticamente la secuencia de los siguientes pasos 2,3 y 4. Si el módulo SPI funciona en modo esclavo, el microcontrolador no ejecutará la secuencia de los siguientes pasos hasta que el pin SCK detecte señal de reloj.

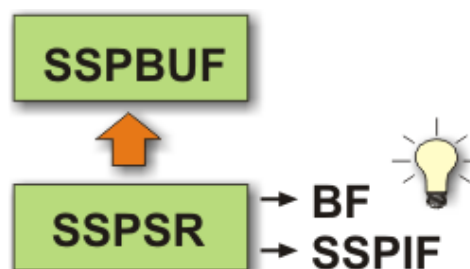


Paso 2.

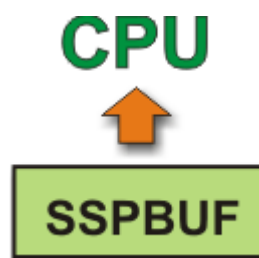
El dato se mueve al registro SSPSR y el contenido del registro SSPBUF no se borra.

**Paso3.**

El dato se desplaza hacia el pin de salida (primero se desplaza el bit más significativo - MSB), mientras que a la vez el registro se carga con los bits por el pin de entrada. En modo maestro el microcontrolador en si mismo genera señal de reloj, mientras que el modo esclavo utiliza señal de reloj externa (pin SCK).

**Paso4.**

El registro SSPSR está lleno después de que hayan sido recibidos 8 bits de datos, lo que se indica al poner a uno el bit BF del registro SSPSTAT y el bit SSPIF del registro PIR1. Los datos recibidos (un byte) son automáticamente movidos del registro SSPSR al registro SSPBUF. Como la transmisión de datos serial se realiza automáticamente, el resto de programa se ejecuta normalmente mientras que la transmisión de datos está en progreso. En este caso, la función del bit SSPIF es de generar una interrupción al acabar la transmisión de un byte.

**Paso5.**

Por último, el dato almacenado en el registro SSPBUF está listo para su uso y debe moverse al registro deseado.

Modo I²C

El modo I²C (Bus de circuito inter-integrado) es adecuado para ser utilizado cuando el microcontrolador debe intercambiar los datos con un circuito integrado dentro de un mismo dispositivo. Éstos son con frecuencia otros microcontroladores, o los circuitos integrados especializados y baratos que pertenecen a la nueva generación de así

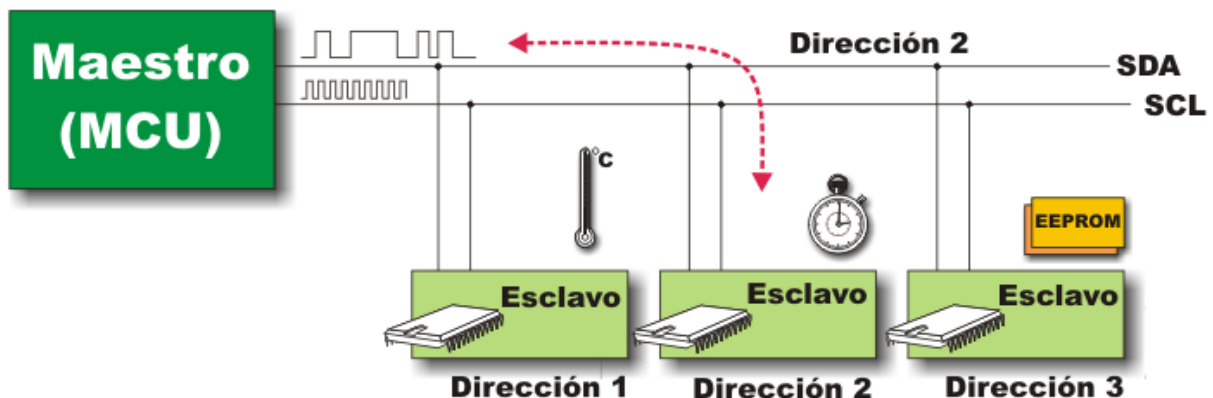
llamados "periféricos inteligentes" (memorias, sensores de temperatura, relojes de tiempo real etc.)

Similar a la comunicación serie en modo SPI, la transmisión de datos en modo I2C es síncrona y bidireccional. Esta vez sólo dos pines se utilizan para transmisión de datos. Éstos son los pines de SDA (Datos seriales) y SCL (Reloj serial). El usuario debe configurar estos pines como entradas o salidas por los bits TRISC.

Al observar las reglas particulares (protocolos), este modo habilita conectar simultáneamente de una manera simple hasta 112 diferentes componentes al utilizar sólo dos valiosos pines de E/S. Vamos a ver cómo funciona el sistema:

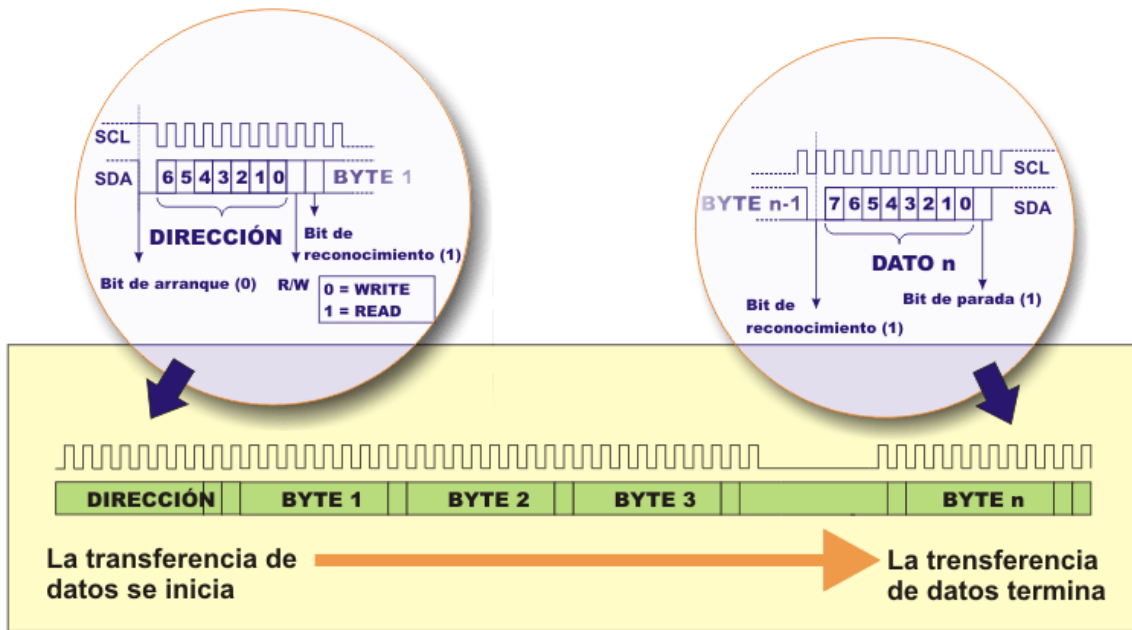
El reloj, necesario para sincronizar el funcionamiento de ambos dispositivos, siempre es generado por un dispositivo maestro (un microcontrolador) y su frecuencia directamente afecta a la velocidad de transmisión de datos. Aunque hay un protocolo que permite como máximo una frecuencia de reloj de 3,4 MHz (así llamado bus I2C de alta velocidad), este libro cubre sólo el protocolo utilizado con más frecuencia, con una frecuencia de reloj limitada a 100 KHz. La frecuencia mínima no está limitada.

Cuando los componentes maestro y esclavo están sincronizados por el reloj, el maestro siempre inicia cada intercambio de datos. Una vez que el módulo MSSP se ha habilitado, espera que ocurra una condición de arranque (Start condition). El dispositivo maestro primero envía el bit de arranque (está a cero) por el pin SDA, luego la dirección de 7 bits del dispositivo esclavo seleccionado, y por último, el bit que requiere al dispositivo escribir (0) o leer (1) el dato enviado. En otras palabras, los ocho bits se desplazan al registro SSPSR después de ocurrir una condición de arranque. Todos los dispositivos esclavos que comparten la misma línea de transmisión recibirán simultáneamente el primer byte, pero sólo el que contiene la dirección coincidente recibirá el dato entero.



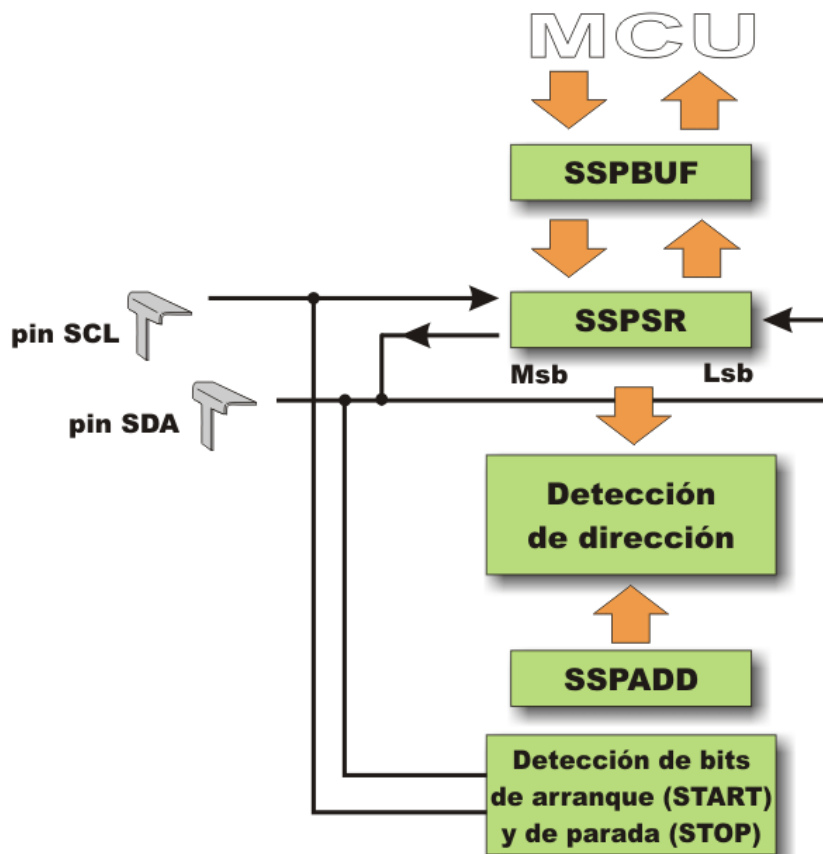
Una vez que el primer byte se ha enviado (sólo se transmiten datos de 8 bits), el maestro se pone en modo de recepción y espera el reconocimiento del dispositivo receptor acerca de la dirección coincidente.

Si el dispositivo esclavo envía un bit de reconocimiento (1) la transmisión de datos continuará hasta que el dispositivo maestro (microcontrolador) envíe el bit de parada (Stop).



Esto es una explicación simple de cómo se comunican dos componentes. Este microcontrolador es capaz de controlar las situaciones más complicadas cuando están conectados 1024 diferentes componentes (dirección de 10 bits), compartidos por varios dispositivos maestros diferentes. Por supuesto, estos dispositivos se utilizan pocas veces en la práctica por lo que no es necesario hablar de ellos detalladamente.

La siguiente figura muestra el diagrama de bloques del módulo MDSSP en modo I²C.



En una operación I²C con el módulo MSSP intervienen seis registros. Algunos de ellos se muestran en la Figura anterior.

- SSPCON
- SSPCON2
- SSPSTAT
- SSPBUF
- SSPSR
- SSPADD

Registro SSPSTAT

SSPSTAT		R/W (0)	R/W (0)	R (0)	R (0)	R (0)	R (0)	R (0)	R (0)	Características
		SMP	CKE	D/A	P	S	R/W	UA	BF	Nombre de bit
		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit se pone a cero

SMP Sample bit (Bit de muestra)

Modo maestro SPI - Este bit determina fase de datos de entrada.

- 1 - Estado lógico se lee al final del tiempo de salida.
- 0 - Estado lógico se lee en la mitad del tiempo de salida.

Modo esclavo SPI - Este bit debe ser borrado cuando SPI se emplea en modo esclavo.

Modo I²C (*maestro o esclavo*)

- 1 - Deshabilita control de variaciones para velocidad estándar (100kHz).
- 0 - Habilita control de variaciones para velocidad alta (400k Hz).

CKE - Clock Edge Select bit (bit de selección del flanco de reloj) selecciona el modo de sincronización.

CKP = 0:

- 1 - Dato transmitido en flanco ascendente de pulso de reloj (0 - 1).
- 0 - Dato transmitido en flanco descendente de pulso de reloj (1 - 0).

CKP = 1:

- 1 - Dato transmitido en flanco descendente de pulso de reloj (1 - 0).
- 0 - Dato transmitido en flanco ascendente de pulso de reloj (0 - 1).

D/A - Data/Address bit (bit de direcciones/datos) se utiliza sólo en modo I²C.

- 1 - Indica que el último byte recibido o transmitido es un dato.
- 0 - Indica que el último byte recibido o transmitido es una dirección.

P - Stop bit (bit de parada) se utiliza sólo en modo I²C.

- 1 - Bit de parada (STOP) se ha detectado.
- 0 - Bit de parada (STOP) no se ha detectado.

S - Start bit (bit de arranque) se utiliza sólo en modo I²C.

- 1 - Bit de arranque (START) se ha detectado.
- 0 - Bit de arranque (START) no se ha detectado.

R/W - Read Write bit (bit de información Lectura/Escritura) se utiliza sólo en modo I²C. Este bit contiene la información del bit de L/E después de la última dirección coincidente. Este bit es válido sólo desde la dirección coincidente hasta el siguiente bit de arranque, bit de parada o bit no ACK.

Modo I²C en modo esclavo

- 1 - Lectura de dato.
- 0 - Escritura de dato.

Modo I²C en modo esclavo

- 1 - Transmisión en progreso.
- 0 - Transmisión no está en progreso.

UA - Update Address bit (bit de activación de dirección) se utiliza sólo en modo I²C de 10 bits.

- 1 - Indica que es necesario actualizar la dirección en el registro SSPADD.
- 0 - Indica que la dirección es correcta y que no se necesita actualizarla.

BF Buffer Full Status bit (bit de estado de búfer lleno)

Durante la recepción de dato (en modos SPI e I²C)

- 1 - Recepción completa. El registro SSPBUF está lleno.
- 0 - Recepción no completa. El registro SSPBUF está vacío.

Durante la transmisión de dato (sólo en modo I²C)

- 1 - Transmisión de dato en progreso (no incluye el bit ACK y bits de parada).
- 0 - Transmisión de dato completa (no incluye el bit ACK y bits de parada).

Registro SSPSTAT

								Características
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Nombre de bit
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Leyenda

R/W (0)	Bit de lectura/escritura
(0)	Después del reinicio, el bit se pone a cero

WCOL Write Collision Detect bit (bit detector de colisión)

- 1 - Colisión detectada. En el registro SSPBUF se ha escrito cuando no se han cumplido las condiciones para iniciar una transmisión.
- 0 - No hay colisión.

SSPOV Receive Overflow Indicator bit (bit detector de desbordamiento en recepción)

- 1 - Se recibe un nuevo byte cuando el registro SSPBUF aún mantiene los datos anteriores. Como no hay espacio para recibir datos nuevos, uno de estos dos bytes debe ser borrado. En este caso, los datos almacenados en el registro SSPSR se pierden irremediablemente.
- 0 - Dato serial es recibido correctamente.

SSPEN - Synchronous Serial Port Enable bit (bit de habilitación del módulo SSP - puerto serie síncrono) determina la función de los pines del microcontrolador e inicializa el módulo MSSP:

En modo SPI

- 1 - Habilita el módulo MSSP y configura los pines SCK, SDO, SDI y SS como una fuente de pines del puerto serie.
- 0 - Deshabilita el módulo MSSP y configura estos pines como pines del puerto de E/S.

En modo I²C

- 1 - Habilita el módulo MSSP y configura los pines SDA y SCL como una fuente de pines del puerto serie.
- 0 - Deshabilita el módulo MSSP y configura estos pines como pines del puerto de E/S.

CKP - Clock Polarity Select bit (bit de selección de polaridad de reloj) no se utiliza en modo I²C maestro.

En modo SPI

- 1 - Para una señal de reloj, el estado inactivo es un nivel alto.
- 0 - Para una señal de reloj, el estado inactivo es un nivel bajo.

En modo I²C esclavo

- 1 - Señal de reloj habilitada.
- 0 - Mantiene la salida de señal de reloj en estado bajo. Se utiliza para proporcionar más tiempo para estabilización de datos.

SSPM3-SSPM0 - Synchronous Serial Port Mode Select bits. (bit de selección del modo del SSP (puerto serie síncrono). El modo SSP se determina al combinar los siguientes bits:

SSPM3	SSPM2	SSPM1	SSPM0	Modo
0	0	0	0	Modo maestro del SPI, reloj = Fosc/4.
0	0	0	1	Modo maestro del SPI, reloj = Fosc/16.
0	0	1	0	Modo maestro del SPI, reloj = Fosc/64.
0	0	1	1	Modo maestro del SPI, reloj = (TMR output)/2.
0	1	0	0	Modo esclavo del SPI, habilitado el pin de control SS.
0	1	0	1	Modo esclavo del SPI, deshabilitado el pin de control SS, SS se puede utilizar como pin de E/S.
0	1	1	0	Modo esclavo I2C, dirección de 7 bits utilizada.
0	1	1	1	Modo esclavo I2C, dirección de 10 bits utilizada.
1	0	0	0	Modo maestro I2C, reloj = Fosc / [4(SSPAD+1)].
1	0	0	1	Máscara utilizada en modo esclavo I2C.
1	0	1	0	No utilizado.
1	0	1	1	Modo maestro I2C controlado.
1	1	0	0	No utilizado.
1	1	0	1	No utilizado.
1	1	1	0	Modo esclavo I2C, dirección de 7 bits utilizada, los bits de arranque (START) y de parada (STOP) habilitan interrupción.
1	1	1	1	Modo esclavo I2C, dirección de 10 bits utilizada, los bits de arranque (START) y de parada (STOP) habilitan interrupción.

Registro SSPCON2

SSPCON2	R/W (0)	R (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit se pone a cero

GCEN - General Call Enable bit (bit de habilitación general)

Sólo en modo esclavo I²C

- 1 - Habilita interrupción cuando una dirección de llamada general es recibida en el SSPST (0000h).
- 0 - Deshabilita dirección de llamada general.

ACKSTAT - Acknowledge Status bit (bit de estado de reconocimiento)

Sólo en modo de transmisión maestro I²C

- 1 - Reconocimiento del esclavo no recibido.
- 0 - Reconocimiento del esclavo recibido.

ACKDT - Acknowledge data bit (bit de recepción)

Sólo en modo de recepción maestro I²C

- 1 - No reconocimiento.
- 0 - Reconocimiento.

ACKEN - Acknowledge Sequence Enable bit (bit de habilitación de secuencia de reconocimiento)

En modo de recepción maestro I²C

- 1 - Indica una secuencia de reconocimiento en los pines SDA y SCL y transmite el bit ACKDT. Automáticamente borrado por hardware.
- 0 - Secuencia de reconocimiento en reposo.

RCEN - Receive Enable bit (bit de habilitación de recepción)

Sólo en modo maestro I²C

- 1 - Habilita recepción en modo I²C.
- 0 - Recepción deshabilitada.

PEN - STOP condition Enable bit (bit de habilitación de condición de Parada)

Sólo en modo maestro I²C

- 1 - Indica una condición de Parada en los pines SDA y SCL. Luego, este bit es automáticamente borrado por hardware.
- 0 - Condición de Parada en reposo.

RSEN - Repeated START Condition Enabled bit (bit de habilitación de repetir condición de Arranque)

Sólo en modo maestro I²C

- 1 - Indica repetición de condición de Arranque en los pines SDA y SCL. Luego, este bit es automáticamente borrado por hardware.
- 0 - Condición de repetición de Arranque en reposo.

SEN - START Condition Enabled/Stretch Enabled bit (bit de habilitación de condición de Arranque)

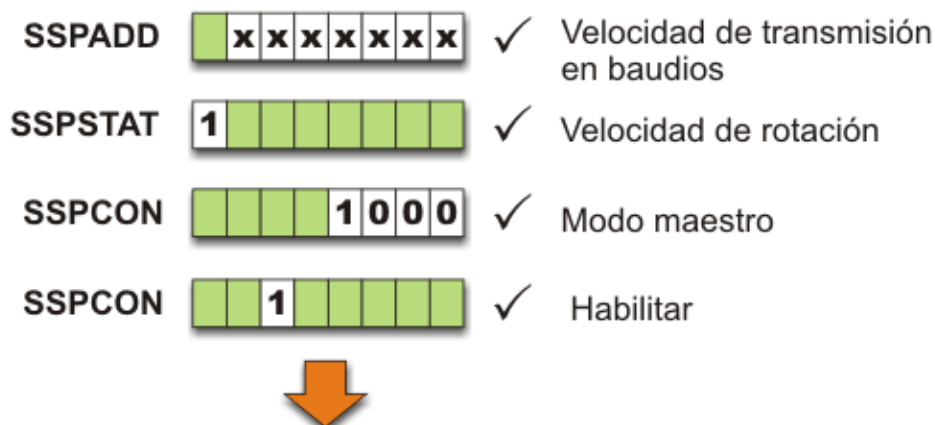
Sólo en modo maestro I²C

- 1 - Indica condición de Arranque en los pines SDA y SCL. Luego, este bit es automáticamente borrado por hardware.
- 0 - Condición de Arranque en reposo.

I²C en Modo Maestro

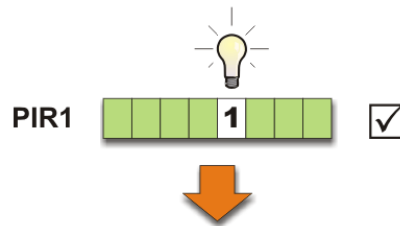
El caso más común es que un microcontrolador funciona como maestro y un periférico como esclavo. Es la razón por la que este libro sólo trata este modo. Se da por entendido que la dirección consiste en 7 bits y el dispositivo contiene un solo microcontrolador (dispositivo con maestro único).

Para habilitar el módulo MSSP en este modo, siga las siguientes instrucciones:



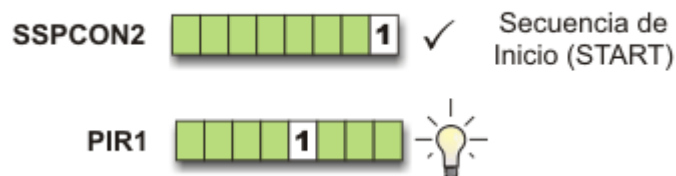
Ajuste la velocidad de transmisión (registro SSPADD), desactive el control de velocidad de rotación (al poner a uno el bit SMP del registro SSPSTAT) y seleccione el modo maestro (registro SSPCON). Después de finalizar todos los ajustes y habilitar el módulo (registro SSPCON: bit SSPEN), es necesario esperar a que los circuitos de control internos indiquen con una señal que todo esté preparado para transmisión de datos: o sea, que el bit SSPIF del registro PIR1 se haya puesto a uno.

Después de poner este bit a cero por software, el microcontrolador está listo para intercambiar los datos con los periféricos.

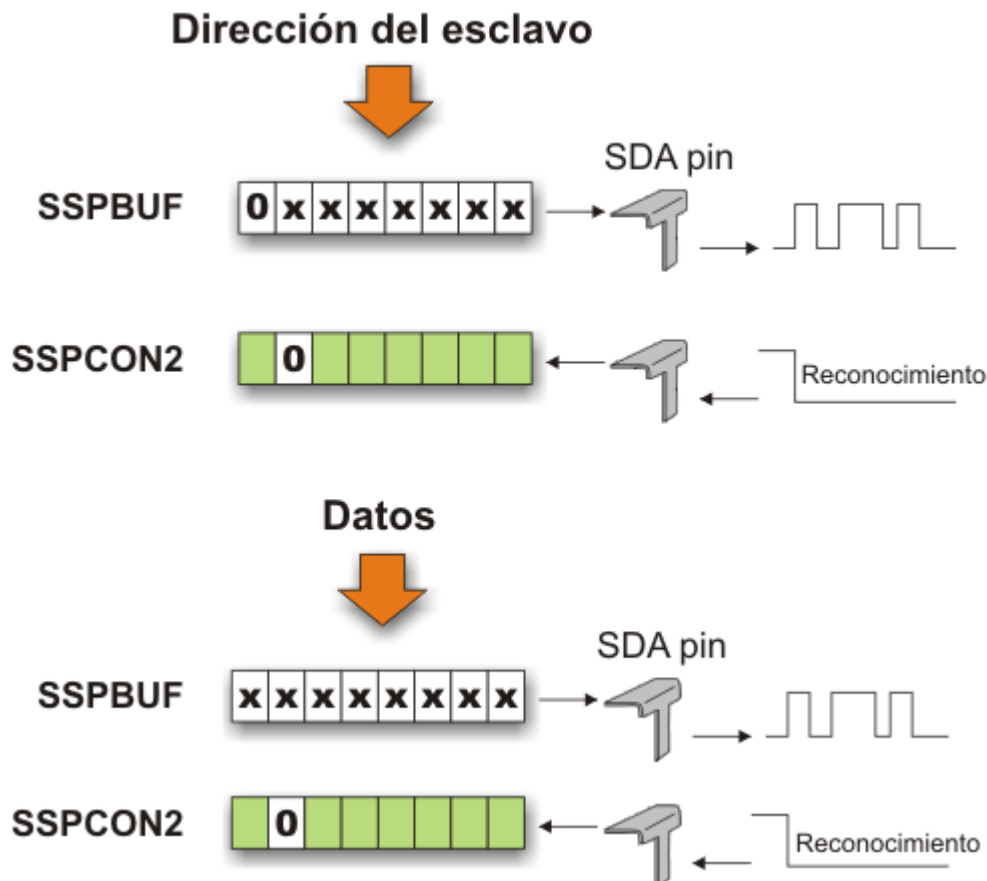


Transmisión de datos en Modo Maestro I²C

La transmisión de datos en el pin SDA se inicia con un cero lógico (0) que aparece al poner a uno el bit SPEN del registro SSPCON2. Sin embargo, aunque está habilitado, el microcontrolador tiene que esperar cierto tiempo antes de iniciar la comunicación. Se le denomina 'Condición de Inicio' durante la que se realizan las preparaciones y verificaciones internas. Si se cumplen con todas las condiciones, el bit SSPIF del registro PIR1 se pone a uno y la transmisión de datos se inicia en cuanto se cargue el registro SSPBUF.



Como máximo 112 circuitos integrados (dispositivos esclavos) pueden compartir simultáneamente la misma línea de transmisión. El primer byte de datos enviado por el dispositivo maestro contiene la dirección que coincide con una sola dirección del dispositivo esclavo. Todas las direcciones se enumeran en las hojas de datos respectivas. El octavo bit del primer byte de datos especifica la dirección de transmisión de datos, o sea si el microcontrolador va a enviar o recibir los datos. En este caso, como se trata de transmisión de datos, el octavo bit se pone a cero (0).



Cuando ocurre la coincidencia de direcciones, el microcontrolador tiene que esperar a que el dispositivo esclavo envíe el bit de reconocimiento, o sea que se ponga a cero el bit ASKSTAT del registro SSPCON2. Una vez que la coincidencia de direcciones ha ocurrido apropiadamente, todos los bytes de datos se transmiten de la misma manera.

La transmisión de datos termina al poner a uno el bit SEN del registro SSPCON2. Ocurre la condición de parada (STOP), lo que habilita que el pin SDA reciba una secuencia de pulsos:

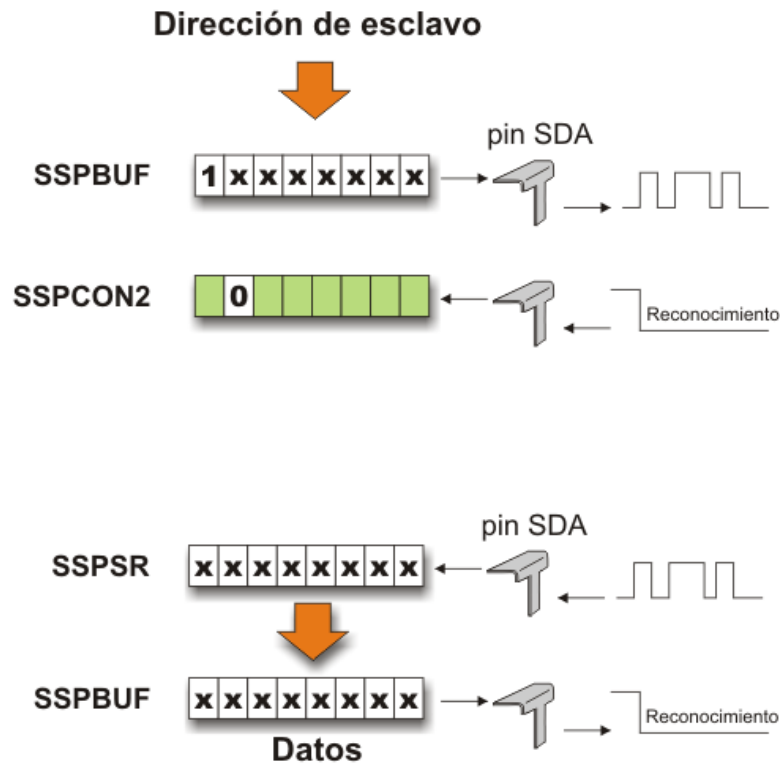
Inicio - Dirección - Reconocimiento - Dato - Reconocimiento Dato - Reconocimiento - Parada!

Recepción de datos en Modo Maestro I²C

Las preparaciones para recibir los datos son similares a las de transmitir los datos, con excepción de que el último bit del primer byte enviado (el que contiene la dirección) se ponga a uno lógico (1). Eso especifica que el dispositivo maestro espera recibir los datos del dispositivo esclavo direccionado. Con respecto al microcontrolador, ocurre lo siguiente:

Después de hacer las pruebas internas y poner a uno el bit de arranque (START), el dispositivo esclavo envía byte por byte. Estos bytes se almacenan en el registro serial SSPSR. Después de recibir el último - octavo bit, cada dato se carga en el registro SSPBUF del que se puede leer. Al leer este registro, se envía automáticamente el bit de reconocimiento, lo que significa que el dispositivo maestro está listo para recibir los nuevos datos.

Al igual que en el caso de la transmisión, la recepción de datos termina al poner a uno el bit de parada (STOP):

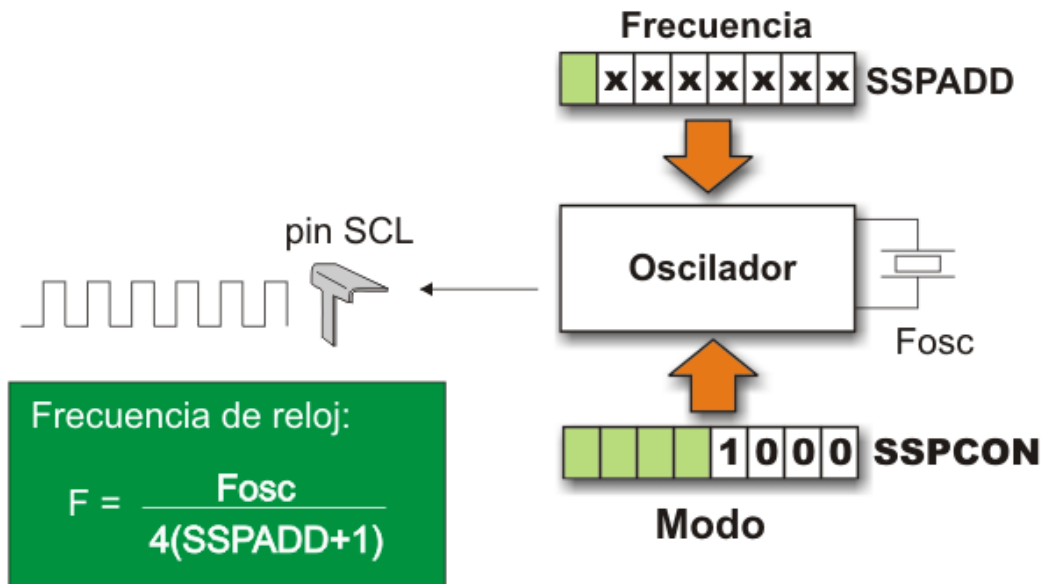


Inicio - Dirección - Reconocimiento - Dato - Reconocimiento Dato - Reconocimiento - ¡Parada!

En esta secuencia de pulsos, el bit de reconocimiento se envía al dispositivo *esclavo*.

Generador de baudios

Para sincronizar la transmisión de datos, todos los eventos que ocurren en el pin SDA deben estar sincronizados con la señal de reloj generada en el dispositivo maestro. Esta señal de reloj se genera por un simple oscilador cuya frecuencia depende de la frecuencia del oscilador principal del microcontrolador, del valor que se introduce al registro SSPADD y así como del modo SPI actual. La frecuencia de señal de reloj del modo descrito en este libro depende del cristal de cuarzo seleccionado y del registro SPADD. La fórmula utilizada para hacer el cálculo de frecuencia de reloj es:



Vamos a hacerlo en mikroC...

/ En este ejemplo, el microcontrolador PIC está conectado a la memoria EEPROM 24C02 por los pines SCL y SDA. El programa envía un byte de dato a la dirección 2 de la EEPROM. Entonces, el programa lee este dato por el modo I2C de la EEPROM y lo envía al puerto PORTB para comprobar si el dato se ha escrito con éxito. El byte para direccionar la EEPROM está compuesto por 7 bits de la dirección (1010001) y el bit que determina lectura o escritura del dato (LSB - bit menos significativo).*/*

```
void main(){
    ANSEL = ANSELH = PORTB = TRISB = 0; // Todos los pines son digitales. Los pines del
        // puerto PORTB son salidas.
    I2C1_Init(100000); // Inicializar I2C con reloj deseado

    // Inicio del bloque de sentencias para escribir un byte en la memoria EEPROM.

    I2C1_Start(); // Señal de inicio de I2C
    I2C1_Wr(0xA2); // Enviar byte por I2C (dirección de dispositivo + W)
    I2C1_Wr(2); // Enviar byte (dirección de la localidad EEPROM)
    I2C1_Wr(0xF0); // Enviar los datos a escribir
    I2C1_Stop(); // Señal de parada de I2C
    Delay_100ms();

    // En el siguiente bloque de sentencias se determina la dirección 2 de la que se leerá el dato

    I2C1_Start(); // Señal de inicio de I2C
    I2C1_Wr(0xA2); // Enviar byte por I2C (dirección de dispositivo + W)
    I2C1_Wr(2); // Enviar byte (dirección de dato)

    // La dirección está determinada y el dato está listo para ser leído

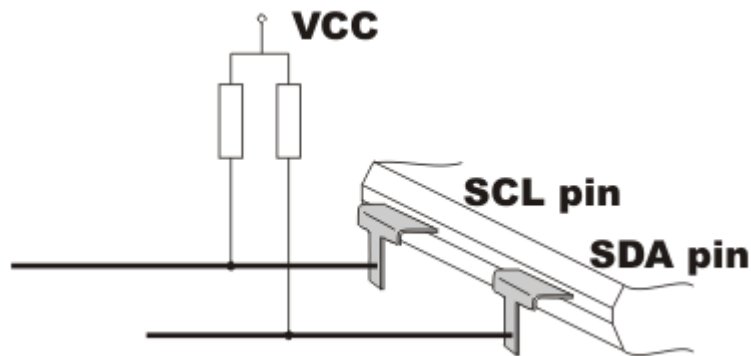
    I2C1_Repeated_Start(); // Se vuelve a generar el inicio de señal I2C
    I2C1_Wr(0xA3); // Enviar byte (dirección de dispositivo + R)
    PORTB = I2C1_Rd(0u); // Leer el dato (reconocimiento NO)
    I2C1_Stop(); // Señal de parada de I2C
}
```

NOTAS ÚTILES ...

Cuando el microcontrolador se comunica con un periférico, puede ocurrir un fallo en la transmisión de datos por alguna razón. En este caso, es recomendable comprobar el estado de algunos bits que pueden aclarar el problema. En la práctica, el estado de estos bits se comprueba al ejecutar una pequeña subrutina después de transmisión y recepción de cada byte (por si acaso).

WCOL (SPCON,7) - Si intenta escribir un dato nuevo al registro SSPBUF mientras que otra transmisión/recepción de datos está en progreso, el bit WCOL se pone a uno y el contenido del registro SSPBUF se queda sin cambios. No hay escritura. Luego, el bit WCOL debe ser borrado por el software.

BF (SSPSTAT,0) - Al transmitir los datos, este bit se pone a uno durante la escritura en el registro SSPBUF y se queda puesto a uno hasta que el byte en formato serial se desplace del registro SSPSR. En modo de recepción, este bit se pone a uno al cargar un dato o una dirección al registro SSPBUF. Se pone a cero después de leer el registro SSPBUF.



SSPOV (SSPCON,6) - En modo de recepción, este bit se pone a uno al recibir un nuevo byte en el registro SSPSR por medio de la comunicación serial, todavía sin haber leído el dato anteriormente recibido del registro SSPBUF.

Pines SDA y SCL - Cuando el módulo SSP está habilitado, estos pines se vuelven a las salidas de Drenaje Abierto. Esto significa que deben estar conectados a resistencias conectados a la otra punta al polo positivo de la fuente de alimentación.

Para establecer la comunicación serial en modo I2C, se debe realizar lo siguiente:

Ajustar el módulo y enviar la dirección:

- Introducir en el registro SSPADD el valor para definir la velocidad de transmisión en baudios.
- Poner a uno el bit SMP del registro SSPSTAT para desactivar el control de la velocidad de rotación.
- Introducir el valor binario 1000 a los bits SSPM3-SSPM0 del registro SSPCON1 para seleccionar el modo Maestro.
- Poner a uno el bit SEN del registro SSPCON2 (secuencia de Inicio - START).

- El bit SSPIF se pone a uno automáticamente en final de la secuencia de Inicio cuando el módulo está listo para funcionar. Se deberá poner a cero.
- Introducir la dirección de esclavo al registro SSPBUF.
- Cuando se envía un byte, el bit SSPIF (interrupción) se pone a uno automáticamente después de haber recibido el bit de reconocimiento del dispositivo esclavo.

Transmitir los datos:

- Introducir en el registro SSPBUF los datos a enviar.
- Cuando se envía un byte, el bit SSPIF (interrupción) se pone a uno automáticamente después de haber recibido el bit de reconocimiento del dispositivo esclavo.
- La condición de Parada (STOP) se debe iniciar al poner a uno el bit PEN del registro SSPCON para informar al dispositivo Esclavo que la transmisión de datos se acabó.

Recibir los datos:

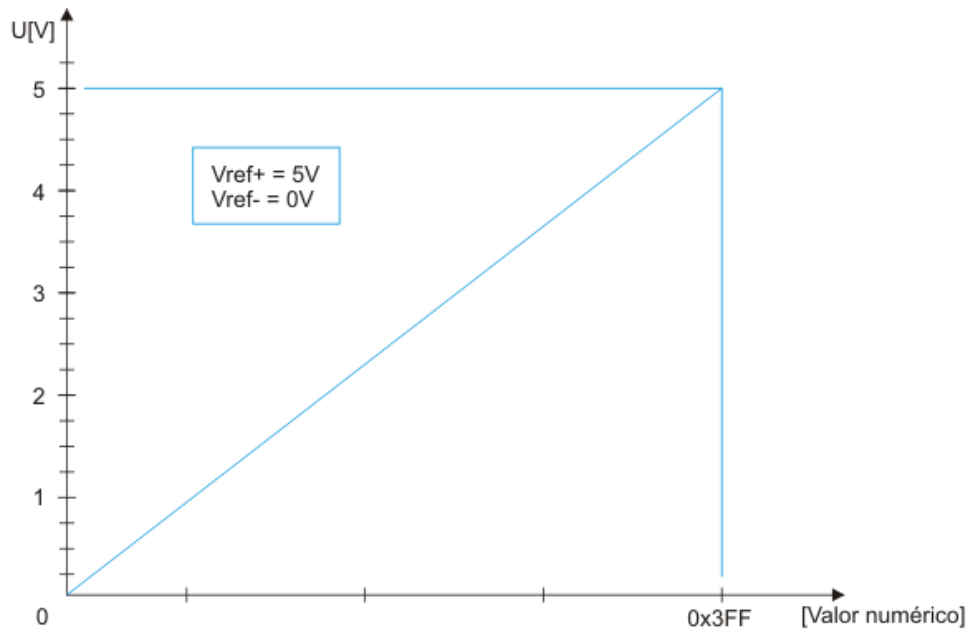
- Poner a uno el bit RSEN del registro SSPCON2 para habilitar la recepción.
- El bit SSPIF indica con su estado lógico la recepción de datos. Después de leer los datos del registro SSPBUF, el bit ACKEN del registro SSPCON2 debe ponerse a uno para habilitar el envío del bit de reconocimiento.
- La condición de Parada (STOP) se debe iniciar al poner a uno el bit PEN del registro SSPCON para informar al dispositivo Esclavo que la transmisión se acabó.

Aparte de disponer de un gran número de líneas digitales de E/S utilizadas para la comunicación con los periféricos, el PIC16F887 contiene 14 entradas analógicas. Debido a éstas, el microcontrolador no sólo puede reconocer si un pin es llevado a bajo o alto (0 o +5V), sino que puede medir con precisión el voltaje y convertirlo en un valor numérico, o sea, en formato digital.

MÓDULOS ANALÓGICOS

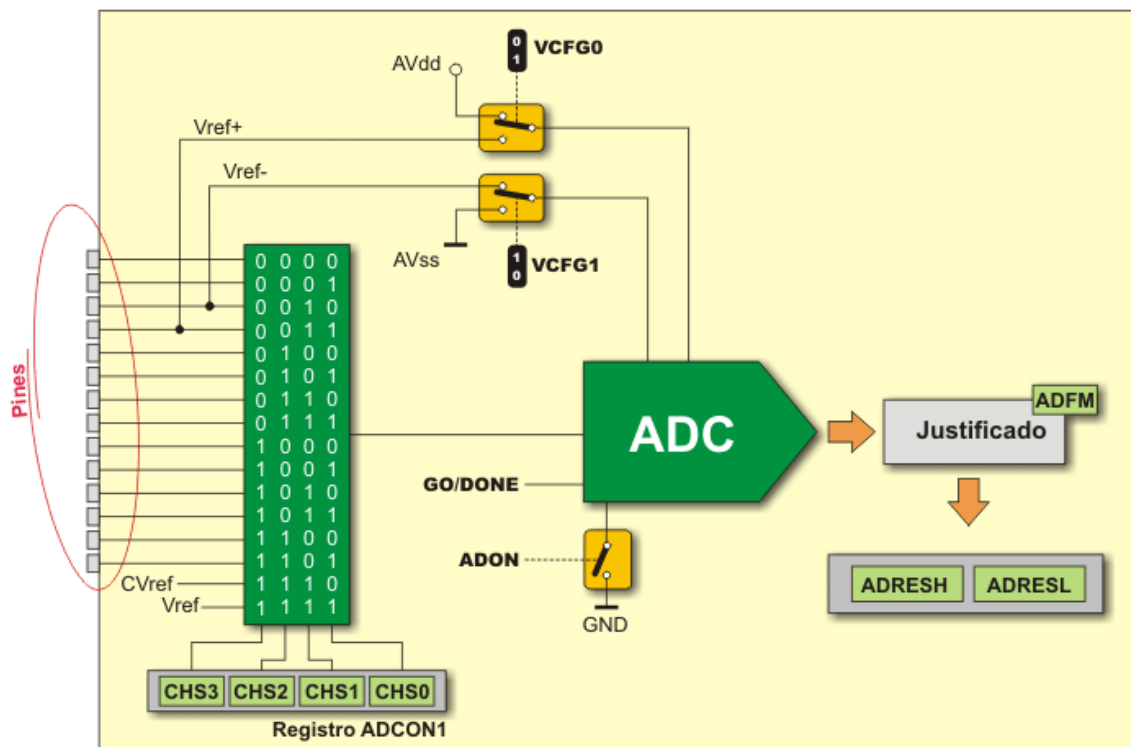
El módulo del convertidor A/D dispone de las siguientes características:

- El convertidor genera un resultado binario de 10 bits utilizando el método de aproximaciones sucesivas y almacena los resultados de conversión en los registros ADC (ADRESL y ADRESH);
- Dispone de 14 entradas analógicas separadas;
- El convertidor A/D convierte una señal de entrada analógica en un número binario de 10 bits;
- La resolución mínima o calidad de conversión se puede ajustar a diferentes necesidades al seleccionar voltajes de referencia Vref- y Vref+.



CONVERTIDOR A/D

Aunque a primera vista parece muy complicado utilizar un convertidor A/D, en realidad es muy simple. De hecho resulta más simple utilizar un convertidor A/D que los temporizadores o módulos de comunicación serie.

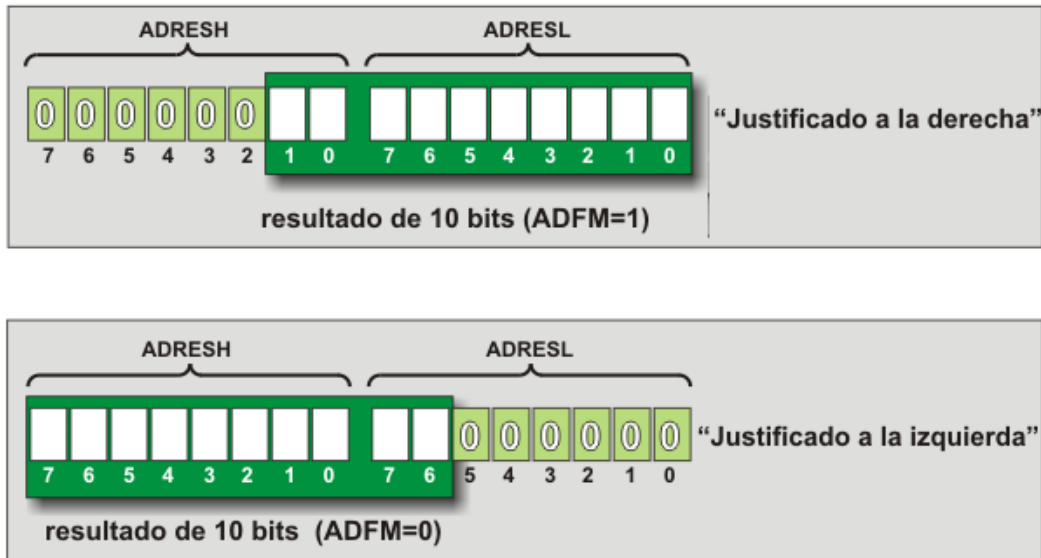


El funcionamiento del convertidor A/D está bajo el control de los bits de cuatro registros:

- ADRESH Registro alto del resultado de la conversión A/D;
- ADRESL Registro bajo del resultado de la conversión A/D;
- ADCON0 Registro de control 0; y
- ADCON1 Registro de control 1.

Registros ADRESH y ADRESL

El resultado obtenido después de convertir un valor analógico en digital es un número de 10 bits que se almacenará en los registros ADRESH y ADRESL. Hay dos maneras de manejarlo: justificación a la izquierda y a la derecha que simplifica en gran medida su uso. El formato del resultado de la conversión depende del bit ADFM del registro ADCON1. En caso de que no se utilice el convertidor A/D, estos registros se pueden utilizar como registros de propósito general.



REQUERIMIENTOS DE ADQUISICIÓN A/D

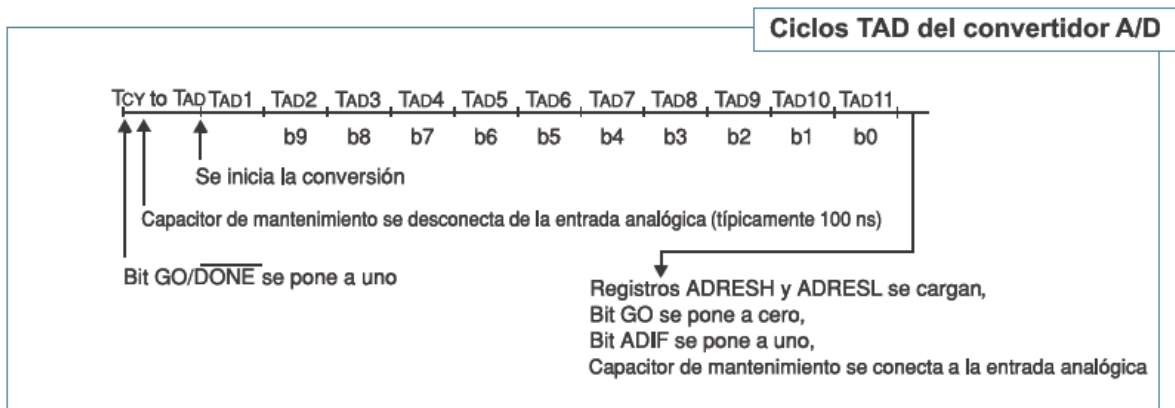
Para que el convertidor A/D alcance su exactitud especificada, es necesario proporcionar un cierto tiempo muerto entre seleccionar una entrada analógica específica y la medición misma. Este tiempo se le denomina "tiempo de adquisición" y generalmente depende de la impedancia de la fuente. Se utiliza una ecuación para hacer cálculo de tiempo de adquisición con precisión, cuyo valor mínimo es de 20uS aproximadamente. Por consiguiente, para realizar una conversión con precisión, no se olvide este detalle.

RELOJ PARA LA CONVERSIÓN A/D

El tiempo necesario para realizar una conversión A/D cuyo resultado es 1 bit se define en unidades de TAD. Se requiere que sea como mínimo 1,6 uS. Para realizar una conversión completa de 10 bits se requiere un poco más tiempo de lo esperado, son 11 TAD. Como la frecuencia de reloj así como la fuente de conversión A/D son determinadas por software, es necesario seleccionar una de las combinaciones de los bits disponibles ADCS1 y ADCS0 antes de empezar a medir voltaje en una de las entradas analógicas. Estos bits se almacenan en el registro ADCON0.

Fuente de reloj de ADC	ADCS1	ADCS0	Frecuencia de dispositivo (Fosc)			
			20 Mhz	8 Mhz	4 Mhz	1 Mhz
Fosc/2	0	0	100 nS	250 nS	500 nS	2 uS
Fosc/8	0	1	400 nS	1 uS	2 uS	8 uS
Fosc/32	1	0	1.6 uS	4 uS	8 uS	32 uS
Frc	1	1	2 - 6 uS	2 - 6 uS	2 - 6 uS	2 - 6 uS

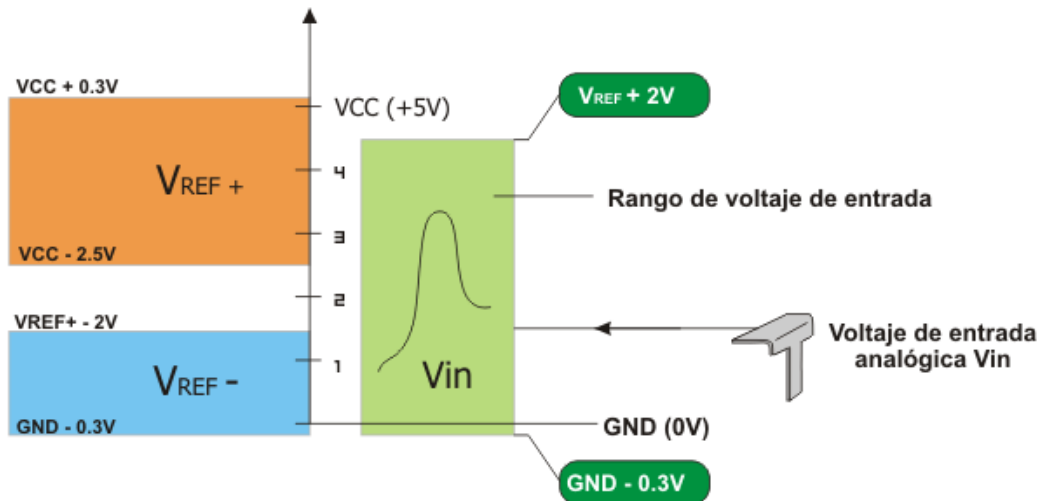
Cualquier cambio de la frecuencia de reloj del microcontrolador afectará a la frecuencia de reloj de la conversión A/D, lo que puede perjudicar al resultado de la conversión A/D. En la siguiente tabla se muestran las características de la frecuencia del dispositivo. Los valores en las celdas sombreadas están fuera del rango recomendado.



¿CÓMO UTILIZAR EL CONVERTIDOR A/D?

Para llevar a cabo una conversión A/D sin problemas así como para evitar los resultados inesperados, es necesario considerar lo siguiente:

- El convertidor A/D no hace diferencia entre señales digitales y analógicas. Para evitar errores en medición o dañar el chip, los pines se deben configurar como entradas analógicas antes de que empiece el proceso de conversión. Los bits utilizados para este propósito se almacenan en los registros TRIS y ANSEL (ANSELH);
- Al leer el estado de puerto con las entradas analógicas, el estado de los bits correspondientes se leerá como cero lógico (0), sin reparar en el valor del voltaje real en el pin; y
- Hablando en términos generales, la medición de voltaje en el convertidor está basado en comparar voltaje de entrada con una escala interna que tiene 1023 grados ($2^{10} - 1 = 1023$). El grado más bajo de esta escala representa el voltaje Vref-, mientras que el grado más alto se refiere al voltaje Vref+. La siguiente figura muestra los voltajes de referencia seleccionables así como sus valores máximos y mínimos.



Registro ADCON0

ADCON0	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

R/W Bit de lectura/escritura
(0) Después del reinicio, el bit se pone a cero

ADCS1, ADCS0 - A/D Conversion Clock Select bits (bits de selección de reloj de conversión A/D) selecciona la frecuencia de reloj utilizada para sincronización interna del convertidor A/D. Asimismo afecta a la duración de la conversión.

ADCS1	ADCS2	Reloj
0	0	Fosc/2
0	1	Fosc/8
1	0	Fosc/32
1	1	RC *

* Señal de reloj se genera por el oscilador interno RC que está integrado en el convertidor.

CHS3-CHS0 - Analog Channel Select bits (bits de selección de canal analógico) selecciona un pin o un canal analógico para la conversión A/D, o sea para medir el voltaje:

CHS3	CHS2	CHS1	CHS0	Canal	Pin
0	0	0	0	0	RA0/AN0
0	0	0	1	1	RA1/AN1
0	0	1	0	2	RA2/AN2
0	0	1	1	3	RA3/AN3
0	1	0	0	4	RA5/AN4

0	1	0	1	5	RE0/AN5
0	1	1	0	6	RE1/AN6
0	1	1	1	7	RE2/AN7
1	0	0	0	8	RB2/AN8
1	0	0	1	9	RB3/AN9
1	0	1	0	10	RB1/AN1 0
1	0	1	1	11	RB4/AN1 1
1	1	0	0	12	RB0/AN1 2
1	1	0	1	13	RB5/AN1 3
1	1	1	0		CVref
1	1	1	1		Vref = 0.6V

GO/DONE - A/D Conversion Status bit (bit de estado de la conversión A/D) determina el estado actual de de la conversión:

- 1 - La conversión A/D está en progreso.
- 0 - La conversión A/D ha finalizado. El bit se pone a cero automáticamente por hardware cuando la conversión A/D finaliza.

ADON - A/D On bit (bit de encendido A/D) habilita el convertidor A/D.

- 1 - Convertidor A/D está habilitado.
- 0 - Convertidor A/D está deshabilitado.

Vamos a hacerlo en mikroC...

/ Este código es un ejemplo de leer el valor analógico del canal 2 y de visualizarlo en los puertos PORTB y PORTC como número binario de 10 bits. */*

```
#include <built_in.h>
unsigned int adc_rd;

void main() {
    ANSEL = 0x04;           // Configurar AN2 como pin analógico
    TRISA = 0xFF;          // PORTA se configura como entrada
    ANSELH = 0;            // Configurar los demás pines AN como E/S digitales
    TRISC = 0x3F;          // Pines RC7 y RC6 se configuran como salidas
    TRISB = 0;             // PORTB se configura como salida

    do {
        temp_res = ADC_Read(2); // Obtener el resultado de 10 bits de la conversión AD
        PORTB = temp_res;       // Enviar los 8 bits más bajos al PORTB
        PORTC = temp_res >> 2; // Enviar los 2 bits más significativos a los RC7, RC6
    } while(1);               // Quedarse en el bucle
}
```

Registro ADCON1

R/W (0)		R/W (0)		R/W (0)		Features	
ADCON1	ADFM	-	VCFG1	VCFG0	-	-	-
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
							Bit 0

Legend

-	Bit is unimplemented
R/W (0)	Readable/Writable bit After reset, bit is cleared

ADFM - A/D Result Format Select bit (bit de selección del formato del resultado de la conversión A/D)

- 1 - Resultado de conversión está justificado a la derecha. No se utilizan los seis bits más significativos del registro ADRESH.
- 0 - Resultado de conversión está justificado a la izquierda. No se utilizan los seis bits menos significativos del registro ADRESL.

VCFG1 - Voltage Reference bit (bit de configuración de voltaje de referencia) selecciona la fuente de voltaje de referencia bajo que se necesita para el funcionamiento del convertidor A/D.

- 1 - Voltaje de referencia bajo se aplica al pin Vref-
- 0 - Voltaje de alimentación Vss se utiliza como una fuente de voltaje de referencia bajo.

VCFG0 - Voltage Reference bit (bit de configuración de voltaje de referencia) selecciona la fuente de voltaje de referencia alto que se necesita para el funcionamiento del convertidor A/D.

- 1 - Voltaje de referencia alto se aplica al pin Vref+.
- 0 - Voltaje de alimentación Vdd se utiliza como una fuente de voltaje de referencia alto.

Para medir el voltaje en un pin de entrada por medio del convertidor A/D, se debe realizar lo siguiente:

Paso 1 - Configuración del puerto:

- Escribir un uno lógico (1) a un bit del registro TRIS, lo que resulta en configurar el pin apropiado como una entrada.
- Escribir un uno lógico (1) a un bit del registro ANSEL, lo que resulta en configurar el pin apropiado como una entrada analógica.

Paso 2 - Configuración del módulo de la conversión A/D:

- Configurar voltaje de referencia en el registro ADCON1.
- Seleccionar una señal de reloj de la conversión A/D en el registro ADCON0.
- Seleccionar uno de los canales de entrada CH0-CH13 del registro ADCON0.
- Seleccionar el formato de dato por medio de ADFM del registro ADCON1.
- Habilitar el convertidor A/D al poner a uno el bit ADON del registro ADCON0.

Paso 3 - Configuración de la interrupción (opcionalmente):

- Poner a cero el bit ADIF.
- Poner a uno los bits ADIE, PEIE y GIE.

Paso 4 - Tiempo de espera para que transcurra el tiempo de adquisición (aproximadamente 20uS).

Paso 5 - Inicio de la conversión poniendo a uno el bit GO/DONE del registro ADCON0.

Paso 6 - Esperar a que la conversión A/D finalice.

- Es necesario comprobar en el bucle de programa si el bit GO/DONE está a cero o esperar que se produzca una interrupción (deberá estar anteriormente habilitada).

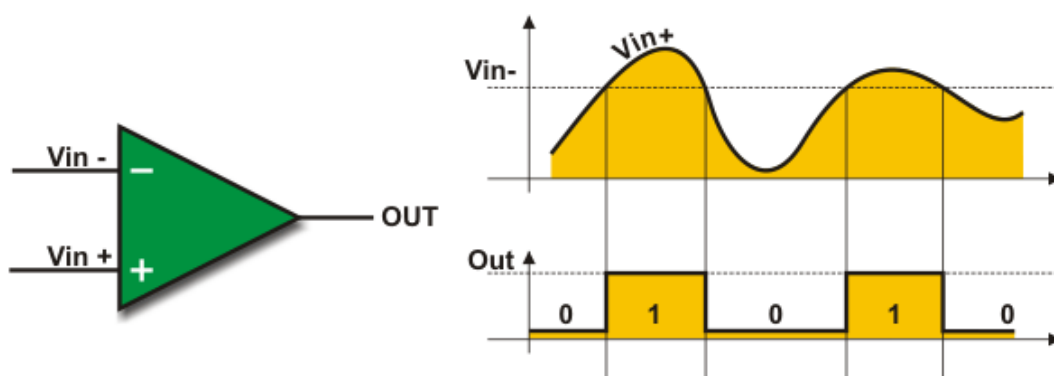
Paso 7 - Lectura del resultado de la conversión A/D:

- Leer los registros ADRESH y ADRESL.

COMPARADOR ANALÓGICO

Aparte del convertidor A/D, hay otro módulo, que hasta hace poco ha sido incorporado sólo en los circuitos integrados que pertenecen a los llamados “componentes analógicos”. Debido al hecho de que casi no hay ningún dispositivo automático complejo que en cierto modo no utilice estos circuitos, dos comparadores de alta calidad, junto con los componentes adicionales están integrados en el microcontrolador y conectados a sus pines.

¿Cómo funciona un comparador? Básicamente, el comparador analógico es un amplificador que compara la magnitud de voltajes en dos entradas. Dispone de dos entradas y una salida. Dependiendo de cuál voltaje de entrada es más alto (valor analógico), un cero lógico (0) o un uno lógico (1) (valores digitales) será la salida.



- Cuando el voltaje analógico en V_{in-} es más alto que el voltaje analógico en V_{in+} , la salida del comparador estará a un nivel digital bajo.
- Cuando el voltaje analógico en V_{in+} es más alto que el voltaje analógico en V_{in-} , la salida del comparador estará a un nivel digital alto.

El microcontrolador PIC16F887 dispone de dos de estos comparadores de voltaje cuyas entradas están conectadas a los pines de E/S RA0-RA3, mientras que las salidas están conectadas a los pines RA4 y RA5. Además, hay una fuente de voltaje de referencia interna en el chip mismo, la que vamos a discutir más tarde.

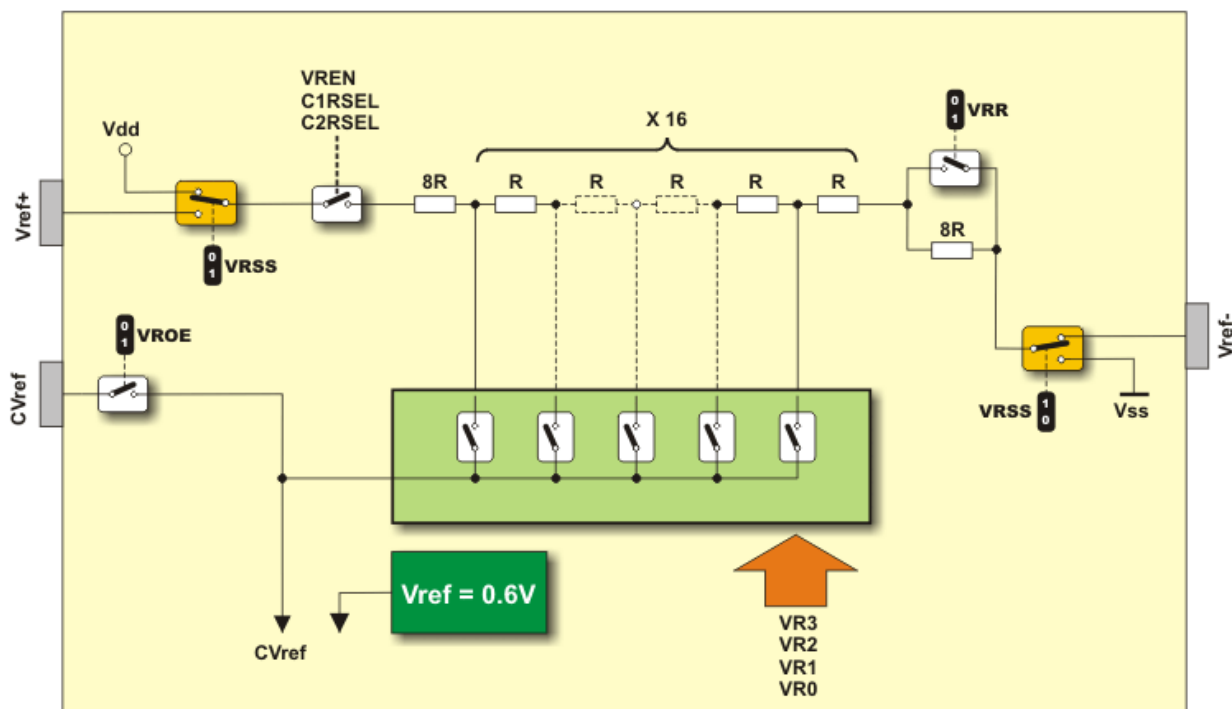
Estos dos circuitos están bajo el control de los bits almacenados en los siguientes registros:

- CM1CON0 está en control del comparador C1;
- CM2CON0 está en control del comparador C2;
- CM2CON1 está en control del comparador C2;

FUENTE INTERNA DE VOLTAJE DE REFERENCIA

Uno de dos voltajes analógicos proporcionados en las entradas del comparador es por lo general estable e inalterable. Es denominado 'voltaje de referencia'(Vref). Para generarlo, se pueden utilizar tanto una fuente de voltaje externa como una fuente de voltaje interna especial. El voltaje de referencia Vref se deriva después de seleccionar una fuente, por medio de una red en escalera que consiste en 16 resistencias, formando un divisor de voltaje. La fuente de voltaje es seleccionable por el bit VRSS del registro VRCON.

Además, la fracción de voltaje proporcionada por la red de resistencias es seleccionable por los bits VR0-VR3 y utilizada como voltaje de referencia. Vea la siguiente figura:



El voltaje de referencia del comparador dispone de dos gamas con 16 diferentes niveles de voltaje cada una. La selección de gama es controlada por el bit VRR del registro $VRCON$. El voltaje de referencia seleccionado $CVref$ puede ser la salida al pin $RA2/AN2$ si el bit $VROE$ se pone a uno.

Aunque la idea principal era obtener el voltaje de referencia variable para el funcionamiento de módulos analógicos, de ese modo se obtiene un simple convertidor A/D. Este convertidor es muy útil en algunas situaciones. Su funcionamiento está bajo el control del registro VRCON.

COMPARADORES E INTERRUPCIÓN

Siempre que haya un cambio del estado lógico en la salida de un comparador, el bit de bandera CMIF del registro PIR se pone a uno. Ese cambio también causará una interrupción si los siguientes bits se ponen a uno:

- El bit CMIE del registro PIE = 1;
- El bit PEIE del registro INTCON = 1; y
- El bit GIE del registro INTCON = 1.

Si una interrupción está habilitada, un cambio en la salida de un comparador cuando el microcontrolador está en modo de reposo, puede causar que el microcontrolador salga de reposo y vuelva a funcionar en modo normal.

FUNCIONAMIENTO EN MODO DE REPOSO (SLEEP MODE)

Si está habilitado antes de entrar en modo de reposo, el comparador se queda activo durante el modo de reposo. Si el comparador no se utiliza para "despertar" el dispositivo, el consumo de corriente se puede reducir en modo de reposo al apagar el comparador. Esto se lleva a cabo al poner a cero el bit CxON del registro CMxCON0.

Para que el comparador "despierte" al microcontrolador del modo de reposo, el bit CxIE del registro IE2 y el bit PEIE del registro INTCON deberán ponerse a uno. La instrucción que sigue a la instrucción Sleep siempre se ejecuta al salir del modo de reposo. Si el bit GIE del registro INTCON se pone a uno, el dispositivo ejecutará la rutina de servicio de interrupción.

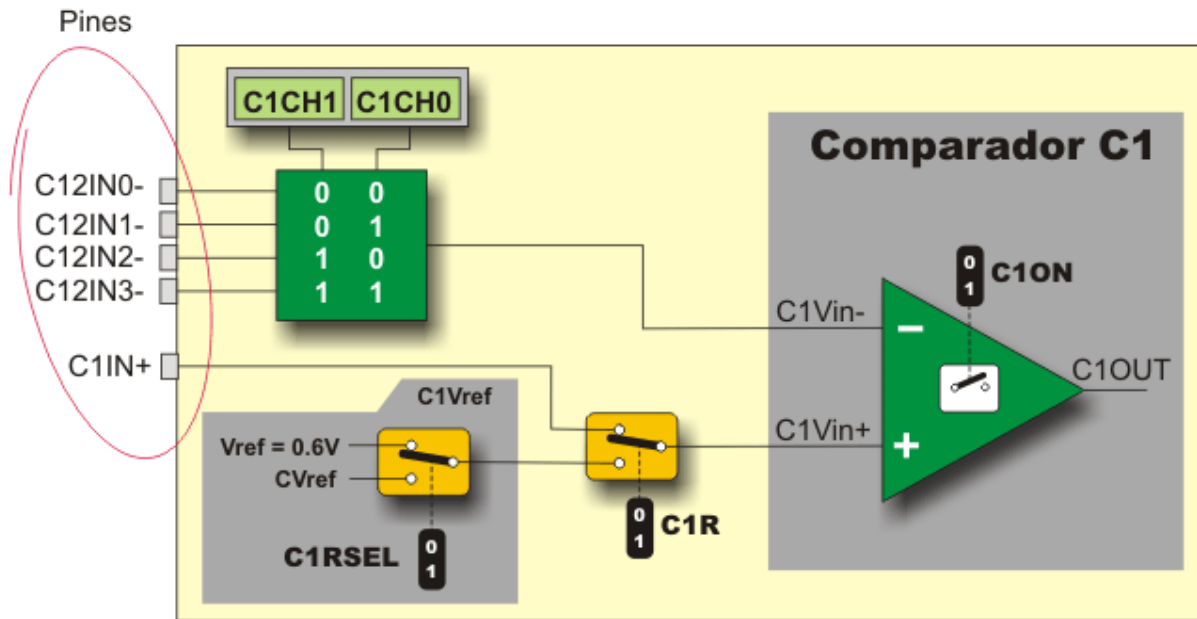
Registro CM1CON0

	R/W (0)	R (0)	R/W (0)	R/W (0)		R/W (0)	R/W (0)	R/W (0)	Características
CM1CON0	C1ON	C1OUT	C1OE	C1POL	-	C1R	C1CH1	C1CH0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit se pone a cero

Los bits de este registro están en control del comparador C1. Eso afecta principalmente a la configuración de las entradas. Para explicarlo con más claridad, vea la siguiente figura en la que se muestran sólo los componentes directamente afectados por los bits de este registro.



C1ON - Comparator C1 Enable bit (bit de habilitación del comparador C1) habilita al comparador C1.

- 1 - Comparador C1 está habilitado.
- 0 - Comparador C1 está deshabilitado.

C1OUT - Comparator C1 Output bit (bit de salida del comparador C1) es la salida del comparador C1.

Si C1POL = 1 (salida del comparador está invertida)

- 1 - Voltaje de entrada C1Vin+ es más bajo que el voltaje de entrada C1Vin-.
- 0 - Voltaje de entrada C1Vin+ es más alto que el voltaje de entrada C1Vin-.

If C1POL = 0 (salida del comparador no está invertida)

- 1 - Voltaje de entrada C1Vin+ es más alto que el voltaje de entrada C1Vin-.
- 0 - Voltaje de entrada C1Vin+ es más bajo que el voltaje de entrada C1Vin-.

C1OE Comparator C1 Output Enable bit (bit de habilitación de salida del comparador C1)

- 1 - Salida del comparador C1OUT está conectada al pin C1OUT *.
- 0 - Salida del comparador se utiliza internamente.

* Para habilitar que el bit C1OUT aparezca en el pin, se deben cumplir dos condiciones: C1ON = 1 (el comparador debe estar activado) y el bit correspondiente TRIS = 0 (pin se debe configurar como salida).

C1POL - Comparator C1 Output Polarity Select bit (bit de selección de polaridad de salida del comparador C1) habilita la inversión del estado de la salida del comparador C1.

- 1 - Salida del comparador C1 está invertida.
- 0 - Salida del comparador C1 no está invertida.

C1R - Comparator C1 Reference Select bit (bit de selección de la fuente de voltaje de referencia del comparador C1)

- 1 - Entrada no invertida C1Vin+ está conectada a la fuente de voltaje de referencia C1Vref.
- 0 - Entrada no invertida C1Vin+ está conectada al pin C1IN+.

C1CH1, C1CH0 - Comparator C1 Channel Select bit (bit de selección de canal del comparador C1)

C1CH1	C1CH0	Entrada C1Vin- del comparador
0	0	Entrada C1Vin- está conectada al pin C12IN0-
0	1	Entrada C1Vin- está conectada al pin C12IN1-
1	0	Entrada C1Vin- está conectada al pin C12IN2-
1	1	Entrada C1Vin- está conectada al pin C12IN3-

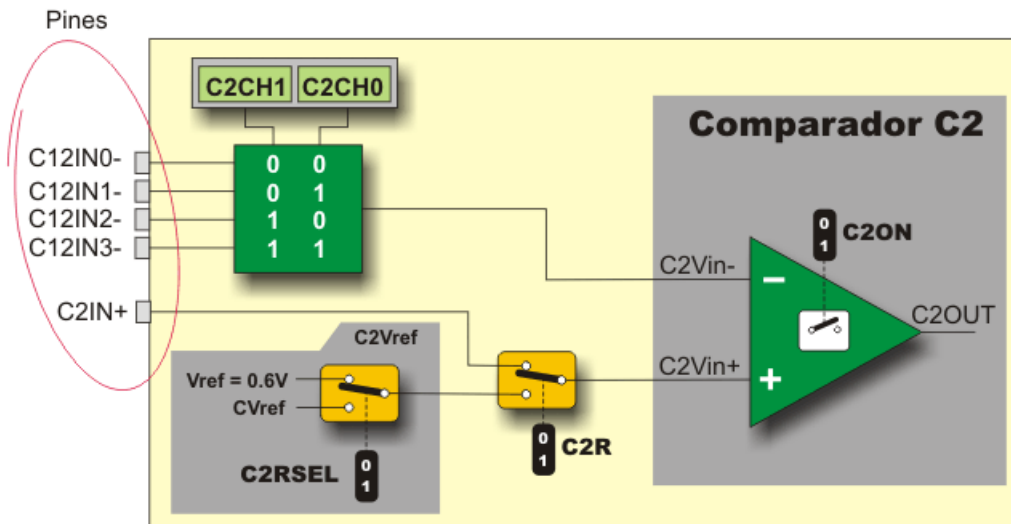
Registro CM2CON0

CM2CON0	R/W (0)	R (0)	R/W (0)	R/W (0)		R/W (0)	R/W (0)	R/W (0)	Características
	C2ON	C2OUT	C2OE	C2POL	-	C2R	C2CH1	C2CH0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

- Bit no implementado
- R/W Bit de lectura/escritura
- R Bit de lectura
- (0) Después del reinicio, el bit se pone a cero

Los bits de este registro están en control del comparador C2. Similar al caso anterior, la siguiente figura muestra un esquema simplificado del circuito afectado por los bits de este registro.



C2ON - Comparator C2 Enable bit (bit de habilitación del comparador C2) habilita el comparador C2.

- 1 - Comparador C2 está habilitado.
- 0 - Comparador C2 está deshabilitado.

C2OUT - Comparator C2 Output bit (bit de salida del comparador C2) es la salida del comparador C2.

If C2POL = 1 (salida del comparador está invertida)

- 1 - Voltaje de entrada C2Vin+ es más bajo que el voltaje de entrada C2Vin-.
- 0 - Voltaje de entrada C2Vin+ es más alto que el voltaje de entrada C2Vin-.

If C2POL = 0 (salida del comparador no está invertida)

- 1 - Voltaje de entrada C2Vin+ es más alto que el voltaje de entrada C2Vin-.
- 0 - Voltaje de entrada C2Vin+ es más bajo que el voltaje de entrada C2Vin-.

C2OE - Comparator C2Output Enable bit (bit de habilitación de salida del comparador C2)

- 1 - Salida del comparador C2OUT está conectada al pin C2OUT*.
- 0 - Salida del comparador se utiliza internamente.

* Para habilitar que el bit C2OUT aparezca en el pin, se deben cumplir dos condiciones: C2ON = 1 (el comparador debe estar activado) y el bit correspondiente TRIS = 0 (pin se debe configurar como salida).

C2POL - Comparator C2 Output Polarity Select bit (bit de selección de polaridad de salida del comparador C2) habilita la inversión del estado de la salida del comparador C2.

- 1 - Salida del comparador C2 está invertida.
- 0 - Salida del comparador C2 no está invertida.

C2R - Comparator C2 Reference Select bit (bit de selección de la fuente de voltaje de referencia del comparador C2)

- 1 - Entrada no invertida C2Vin+ está conectada a la fuente de voltaje de referencia C2Vref.
- 0 - Entrada no invertida C2Vin+ está conectada al pin C2IN+.

C2CH1, C2CH0 Comparator C2 Channel Select bit (bit de selección de canal del comparador C2)

C2CH1 C2CH0 Entrada C2Vin- del comparador

0	0	Entrada C2Vin- está conectada al pin C12IN0-
0	1	Entrada C2Vin- está conectada al pin C12IN1-
1	0	Entrada C2Vin- está conectada al pin C12IN2-
1	1	Entrada C2Vin- está conectada al pin C12IN3-

Registro CM2CON1

	R (0)	R (0)	R/W (0)	R/W (0)			R/W (1)	R/W (0)	Características
CM2CON1	MC1OUT	MC2OUT	C1RSEL	C2RSEL	-	-	T1GSS	C2SYNC	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit se pone a cero
(1)	Después del reinicio, el bit se pone a uno

MC1OUT Mirror Copy of C1OUT bit es una copia del bit C1OUT

MC2OUT Mirror Copy of C2OUT bit es una copia del bit C2OUT

C1RSEL Comparator C1 Reference Select bit (bit de selección de la fuente de voltaje de referencia del comparador C1)

- 1 - Voltaje seleccionable CVref se utiliza en la fuente de voltaje de referencia C1Vref.
- 0 - Voltaje de referencia fijo de 0,6V se utiliza en la fuente de voltaje de referencia C1Vref.

C2RSEL - Comparator C2 Reference Select bit (bit de selección de la fuente de voltaje de referencia del comparador C2)

- 1 - Voltaje seleccionable CVref se utiliza en la fuente de voltaje de referencia C2Vref.
- 0 - Voltaje de referencia fijo de 0,6V se utiliza en la fuente de voltaje de referencia C2Vref.

T1GSS - Timer1 Gate Source Select bit (bit de selección de la fuente de la compuerta del temporizador Timer1)

- 1 - Compuerta del temporizador Timer1 utiliza señal del pin T1G.
- 0 - Compuerta del temporizador Timer1 utiliza señal SYNCC2OUT.

C2SYNC - Comparator C2 Output Synchronization bit (bit de sincronización de salida del comparador C2)

- 1 - Salida del comparador C2 está sincronizada con un flanco ascendente de señal de reloj del temporizador Timer1
- 0 - Salida del comparador es una señal asíncrona.

Registro VRCON

VRCON	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
	VREN	VROE	VRR	VRSS	VR3	VR2	VR1	VR0	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Legenda

R/W (0)	Bit de lectura/escritura Después del reinicio, el bit se pone a cero
---------	---

VREN Comparator C1 Voltage Reference Enable bit (bit de habilitación de la fuente de voltaje de referencia del comparador C1)

- 1 - Fuente de voltaje de referencia CVref está encendido.
- 0 - Fuente de voltaje de referencia CVref está apagado.

VROE Comparator C2 Voltage Reference Enable bit (bit de habilitación de la fuente de voltaje de referencia del comparador C2)

- 1 - Fuente de voltaje de referencia CVref está conectada al pin.
- 0 - Fuente de voltaje de referencia CVref no está conectada al pin.

VRR - CVref Range Selection bit (bit de selección de gama de voltaje de referencia Vref)

- 1 - Fuente de voltaje de referencia se ajusta a producir baja gama de voltaje.
- 0 - Fuente de voltaje de referencia se ajusta a producir alta gama de voltaje.

VRSS - Comparator Vref Range selection bit (bit de selección de gama de voltaje de referencia Vref del comparador)

- 1 - Voltaje de referencia está en la gama de Vref+ a Vref-.
- 0 - Voltaje de referencia está en la gama de Vdd a Vss. (voltaje de alimentación).

VR3 - VR0 CVref Value Selection (selección de valor de voltaje de referencia)

If VRR = 1 (gama baja)

El voltaje de referencia se calcula por medio de la fórmula: $CV_{ref} = ([VR3:VR0]/24)V_{dd}$.

If VRR = 0 (gama alta)

El voltaje de referencia se calcula por medio de la fórmula $CV_{ref} = V_{dd}/4 + ([VR3:VR0]/32)V_{dd}$.

Pasos a seguir para utilizar apropiadamente los comparadores integrados:

Paso 1 - Configuración del módulo:

- Para seleccionar el modo apropiado, se deben configurar los estados de los bits de los registros CM1CON0 y CM2CON0. La interrupción debe estar deshabilitada durante el cambio de modo.

Paso 2 - Configurar la fuente de voltaje de referencia Vref interna (sólo si se utiliza). En el registro VRCON es necesario realizar lo siguiente:

- Seleccionar una de dos gamas de voltaje por medio del bit VRR.
- Configurar el voltaje de referencia Vref necesario por medio de los bits VR3 - VR0.
- Poner a uno el bit VROE si es necesario.
- Habilitar la fuente de voltaje de referencia Vref al poner a uno el bit VREN.

Fórmula utilizada para calcular el voltaje de referencia

VRR = 1 (gama baja)

$$CV_{ref} = ([VR3:VR0]/24)VLADDER$$

VRR = 0 (gama alta)

$$CV_{ref} = (VLADDER/4) + ([VR3:VR0]VLADDER/32)$$

$$V_{ladder} = V_{dd} \text{ or } ([V_{ref+}] - [V_{ref-}]) \text{ or } V_{ref+}$$

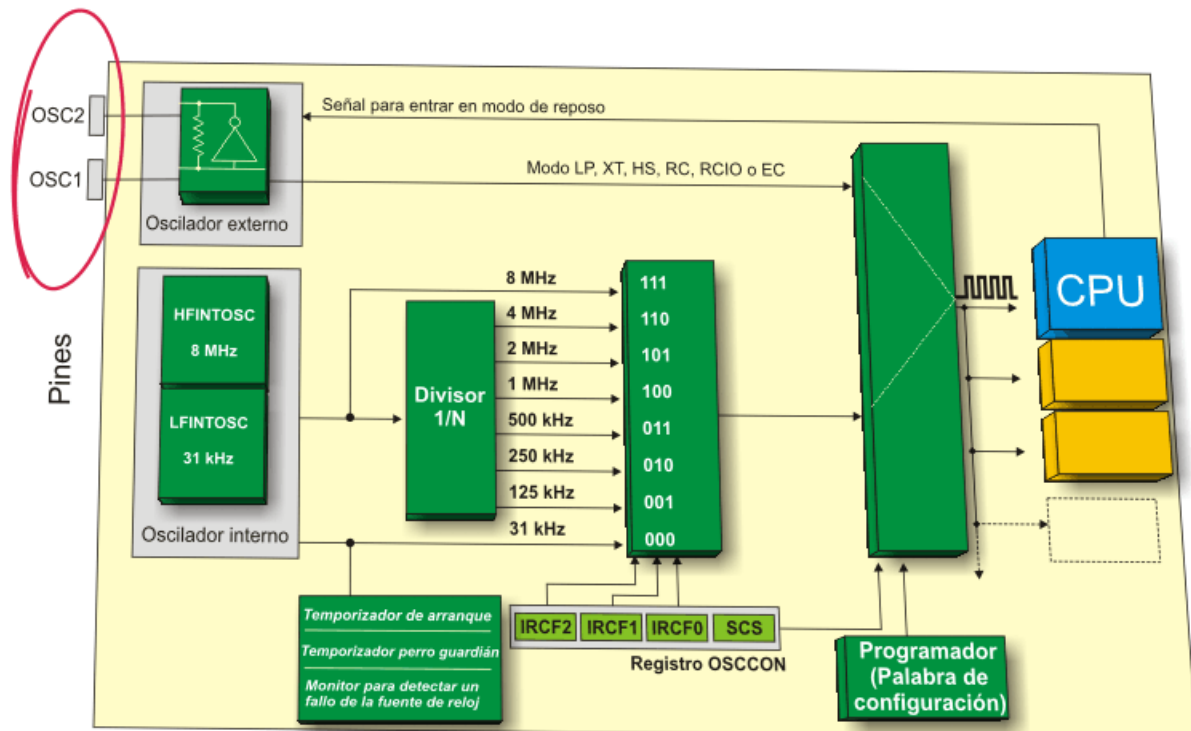
Paso 3 - Inicio del funcionamiento:

- Habilitar una interrupción al poner a uno los bits CMIE (registro PIE), PEIE y GIE (registro INTCON).
- Leer los bits C1OUT y C2OUT del registro CMCON.
- Leer la bandera de bit CMIF del registro PIR. Después de haber sido puesto a uno, este bit se pone a cero por software.

Para sincronizar todos los procesos que se llevan a cabo dentro del microcontrolador, se debe utilizar una señal de reloj, mientras que para generar una señal de reloj, se debe utilizar un oscilador. Así de simple. El microcontrolador dispone de varios osciladores capaces de funcionar en modos diferentes. Y aquí es donde viene lo interesante...

OSCILADOR DE RELOJ

Como se muestra en la siguiente figura, la señal de reloj se genera por uno de los dos osciladores integrados.



Un **oscilador externo** está instalado fuera del microcontrolador y conectado a los pines OSC1 y OSC2. Es denominado 'externo' porque utiliza componentes externos para generar una señal de reloj y estabilizar la frecuencia. Estos son: cristal de cuarzo, resonador cerámico o circuito resistor - capacitor. El modo de funcionamiento del oscilador se selecciona por los bits, que se envían durante la programación, denominados Palabra de Configuración.

El **oscilador interno** consiste en dos osciladores internos separados:

El HFINTOSC es un oscilador interno de alta frecuencia calibrado a 8MHz. El microcontrolador puede utilizar una señal de reloj generada a esta frecuencia o después de haber sido dividida en el pre-escalador.

El LFINTOSC es un oscilador interno de baja frecuencia calibrado a 31 kHz. Sus pulsos de reloj se utilizan para funcionamiento de los temporizadores de encendido y perro guardián, asimismo puede utilizarse como fuente de señal de reloj para el funcionamiento de todo el microcontrolador.

El bit *System Clock Select* (bit de selección del reloj del sistema - SCS) del registro OSCCON determina si una fuente de señal de reloj del microcontrolador será interna o externa.

Registro OSCCON

El registro OSCCON gobierna el microcontrolador y las opciones de selección de frecuencia. Contiene los siguientes bits: bits de selección de frecuencia (IRCF2, IRCF1, IRCF0), bits de estado de frecuencia (HTS, LTS), bits de control de reloj del sistema (OSTA, SCS).

OSCCON	-	IRCF2	IRCF1	IRCF0	OSTS	HTS	LTS	SCS	Características
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Nombre de bit
		R/W (1)	R/W (1)	R/W (0)	R (1)	R (0)	R (0)	R/W (0)	

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit se pone a cero
(1)	Después del reinicio, el bit se pone a uno

IRCF2-0 - Internal Oscillator Frequency Select bits. (bits de selección de frecuencia del oscilador interno). El valor del divisor de frecuencias depende de la combinación de estos tres bits. La frecuencia de reloj del oscilador interno se determina de la misma manera.

IRCF2	IRCF1	IRCF0	Frecuencia	OSC.
1	1	1	8 MHz	HFINTOSC
1	1	0	4 MHz	HFINTOSC
1	0	1	2 MHz	HFINTOSC
1	0	0	1 MHz	HFINTOSC
0	1	1	500 kHz	HFINTOSC
0	1	0	250 kHz	HFINTOSC
0	0	1	125 kHz	HFINTOSC
0	0	0	31 kHz	LFINTOSC

OSTS - Oscillator Start-up Time-out Status bit (bit de estado del temporizador de encendido) indica cuál fuente de reloj está actualmente en uso. Es un bit de sólo lectura.

- 1 - Se utiliza el oscilador de reloj externo.
- 0 - Se utiliza uno de los osciladores de reloj interno (HFINTOSC o LFINTOSC).

HTS - HFINTOSC Status bit (8 MHz - 125 kHz) (bit de estado del HFINTOSC) indica si el oscilador interno de alta frecuencia funciona en modo estable.

- 1 - HFINTOSC está estable.
- 0 - HFINTOSC no está estable.

LTS - LFINTOSC Stable bit (31 kHz) (bit de estado del LFINTOSC) indica si el oscilador de baja frecuencia funciona en modo estable.

- 1 - LFINTOSC está estable.
- 0 - LFINTOSC no está estable.

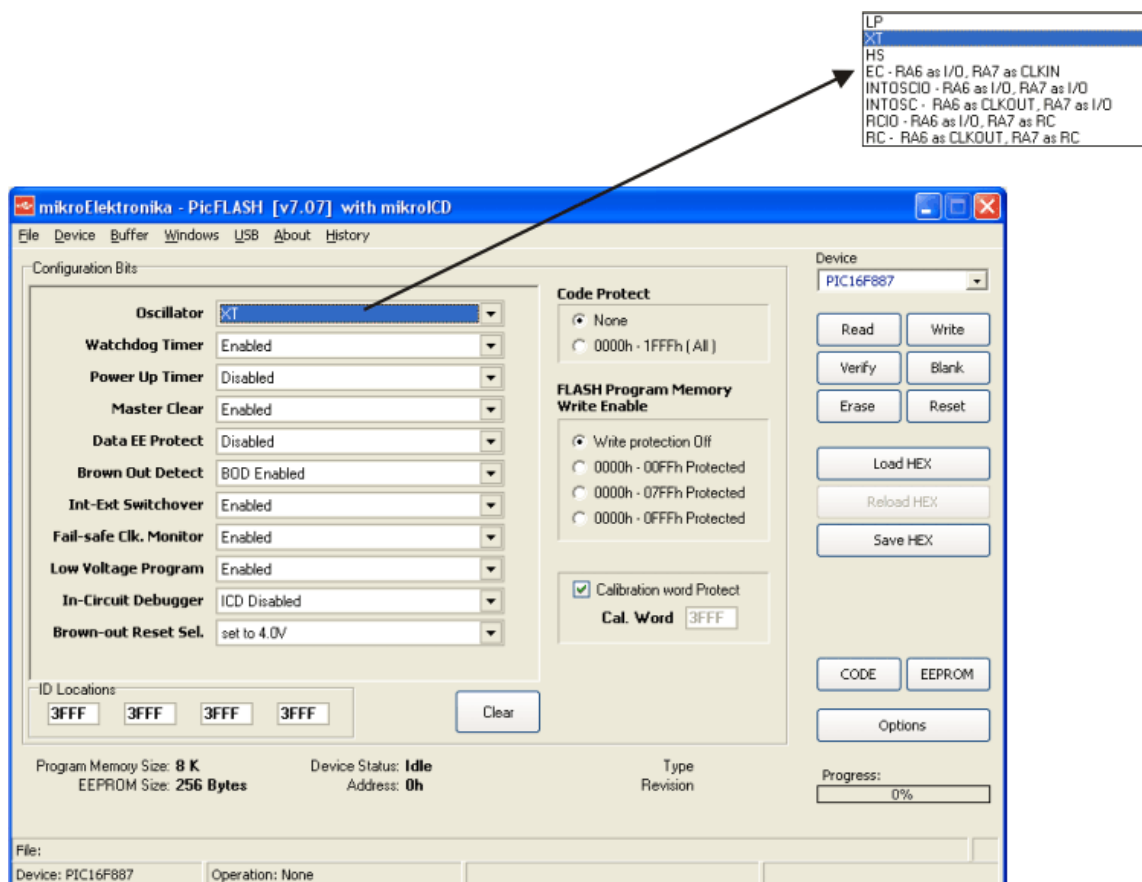
SCS - System Clock Select bit (bit de selección del reloj del sistema) determina cuál oscilador se utilizará como una fuente de reloj.

- 1 - Oscilador interno se utiliza como reloj del sistema.
 - 0 - Oscilador externo se utiliza como reloj del sistema.
- El modo del oscilador se configura por medio de los bits, denominados *Palabra de Configuración*, que se escribe en la memoria del microcontrolador durante el proceso de la programación.

MODOS DE RELOJ EXTERNO

El oscilador externo se puede configurar para funcionar en uno de varios modos, lo que habilita que funcione a diferentes velocidades y utilice diferentes componentes para estabilizar la frecuencia. El modo de funcionamiento se selecciona durante el proceso de escribir un programa en el microcontrolador. Antes que nada, es necesario activar el programa en una PC que se utilizará para programar el microcontrolador. En este caso, es el programa PICflash. Pulse sobre la casilla del oscilador y seleccione uno de la lista desplegable. Los bits apropiados se pondrán a uno automáticamente, formando parte de varios bytes, denominados Palabra de Configuración.

Durante el proceso de la programación del microcontrolador, los bytes de la Palabra de Configuración se escriben en la memoria ROM del microcontrolador y se almacenan en los registros especiales no disponibles al usuario. A base de estos bits, el microcontrolador “sabe” qué hacer, aunque eso no se indica explícitamente en el programa.



Modo de funcionamiento se selecciona después de escribir y compilar un programa

OSCILADOR EXTERNO EN MODO EC

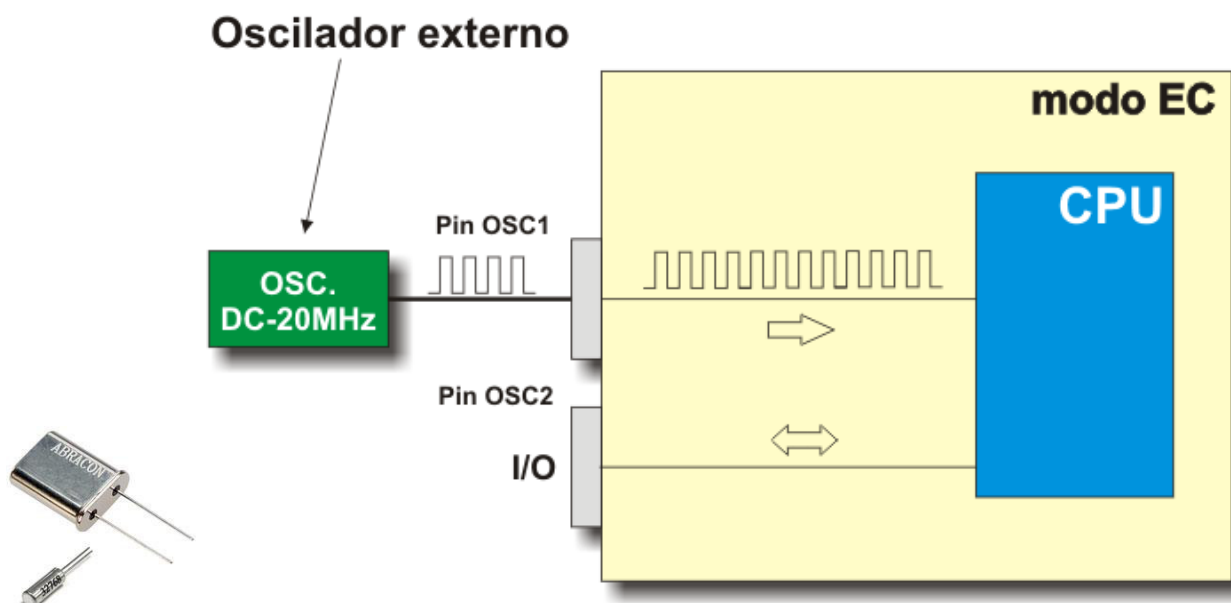
El modo de reloj externo (EC - external clock) utiliza un oscilador externo como una fuente de señal de reloj. La máxima frecuencia de señal de reloj está limitada a 20 MHz.



Las ventajas del funcionamiento del oscilador externo en modo EC son las siguientes:

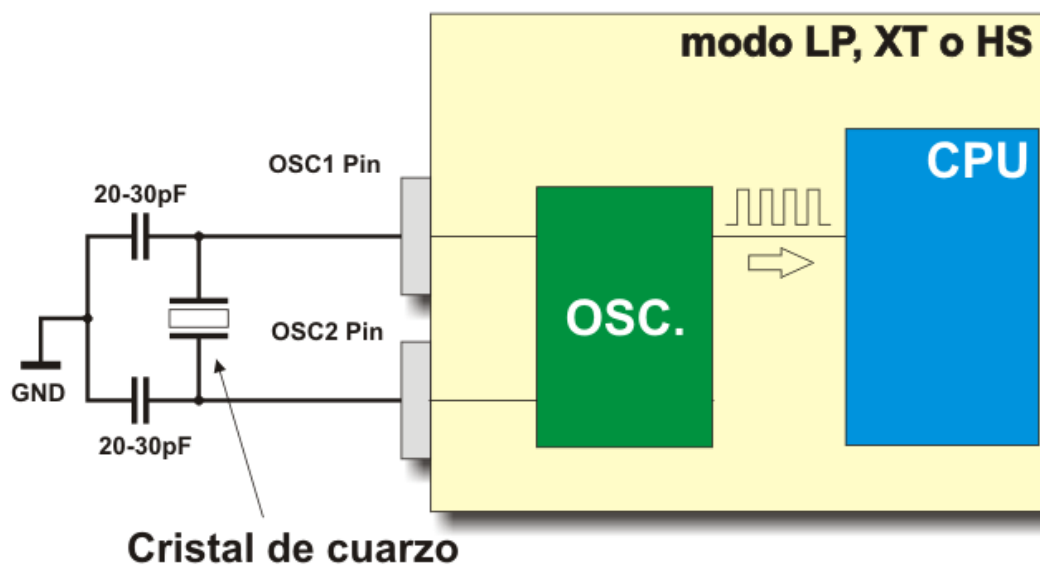
- La fuente de reloj externa independiente está conectada al pin de entrada OSC1. El pin OSC2 está disponible como pin de E/S de propósito general;
- Es posible sincronizar el funcionamiento del microcontrolador con los demás componentes incorporados en el dispositivo;
- En este modo el microcontrolador se pone a funcionar inmediatamente después de encenderlo. No se requiere esperar para estabilizar la frecuencia.
- Al deshabilitar temporalmente la fuente de reloj externa, se detiene el funcionamiento del dispositivo, dejando todos los datos intactos. Después de reiniciar el reloj externo, el dispositivo sigue funcionando como si no hubiera pasado nada.

OSCILADOR EXTERNO EN MODO LP, XT O HS



Los modos LP, XT y HS utilizan un oscilador externo como una fuente de reloj cuya frecuencia está determinada por un cristal de cuarzo o por resonadores cerámicos conectados a los pines OSC1 y OSC2. Dependiendo de las características de los componentes utilizados, seleccione uno de los siguientes modos:

- **Modo LP** - (Baja potencia) se utiliza sólo para cristal de cuarzo de baja frecuencia. Este modo está destinado para trabajar con cristales de 32.768 KHz normalmente embebidos en los relojes de cristal. Es fácil de reconocerlos por sus dimensiones pequeñas y una forma cilíndrica. Al utilizar este modo el consumo de corriente será menor que en los demás modos.
- **Modo XT** se utiliza para cristales de cuarzo de frecuencias intermedias hasta 8 MHz. El consumo de corriente es media en comparación con los demás modos.
- **Modo HS** - (Alta velocidad) se utiliza para cristales de reloj de frecuencia más alta de 8 MHz. Al utilizar este modo el consumo de corriente será mayor que en los demás modos.



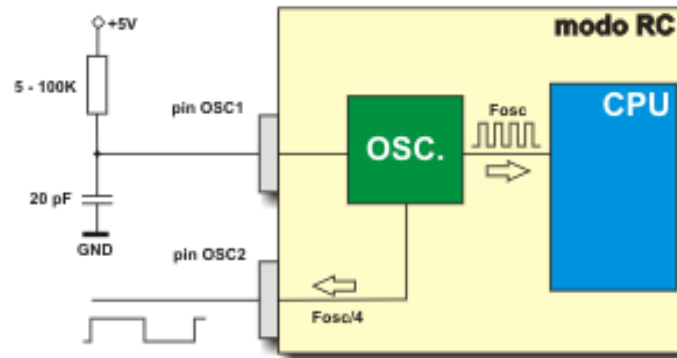
RESONADORES CERÁMICOS EN MODO XT O HS

Los resonadores cerámicos son similares a los cristales de cuarzo según sus características, por lo que se conectan de la misma manera. A diferencia de los cristales de cuarzo, son más baratos y los osciladores que hacen uso de ellos son de calidad más baja. Se utilizan para las frecuencias de reloj entre 100 kHz y 20 MHz.

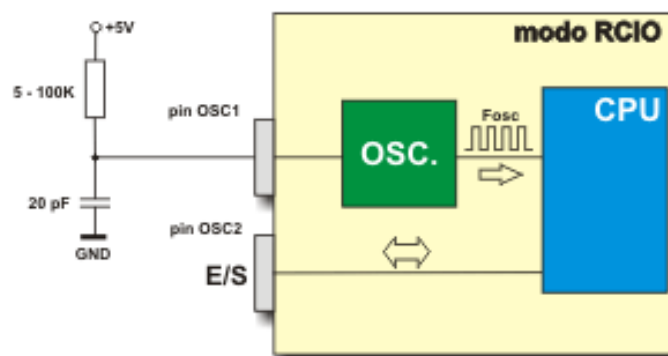


OSCILADOR EXTERNO EN MODOS RC Y RCIO

El uso de los elementos para estabilizar la frecuencia sin duda alguna tiene muchas ventajas, pero a veces realmente no es necesario. En la mayoría de casos el oscilador puede funcionar a frecuencias que no son precisamente definidas, así que sería una pérdida de dinero embeber tales elementos. La solución más simple y más barata es estas situaciones es utilizar una resistencia y un capacitor para el funcionamiento del oscilador. Hay dos modos:



Modo RC. Cuando el oscilador externo se configura a funcionar en modo RC, el pin OSC1 debe estar conectado al circuito RC como se muestra en la figura a la derecha. La señal de frecuencia del oscilador RC dividida por 4 está disponible en el pin OSC2. Esta señal se puede utilizar para la calibración, sincronización o para otros propósitos.



Modo RCIO. De manera similar, el circuito RC está conectado al pin OSC1. Esta vez, el pin OSC2 está disponible para ser utilizado como pin de E/S de propósito general.

En ambos casos se le recomienda utilizar los componentes como se muestra en la figura.

La frecuencia de este oscilador se calcula por medio de la fórmula $f = 1/T$ según la que:

- f = frecuencia [Hz];
- $T = R * C$ = constante de tiempo [s];
- R = resistencia eléctrica [Ω]; y
- C = capacitancia del condensador [F].

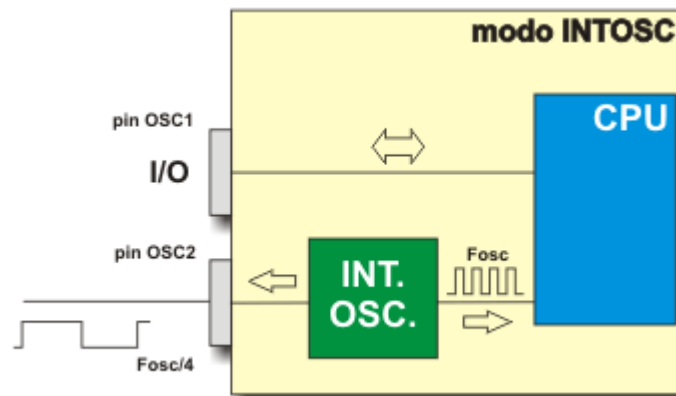
MODOS DE RELOJ INTERNO

El circuito del oscilador interno consiste en dos osciladores separados que se pueden seleccionar como la fuente del reloj del microcontrolador:

El oscilador **HFINTOSC** está calibrado de fábrica y funciona a 8Mhz. La frecuencia de este oscilador se puede configurar por el usuario por medio de software utilizando los bits del registro OSCTUNE.

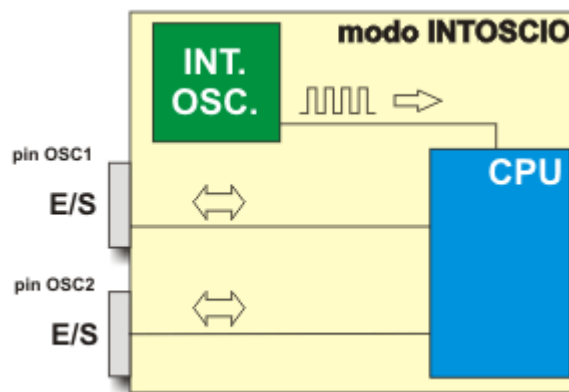
El oscilador **LFINTOSC** no está calibrado de fábrica y funciona a 31kHz.

Similar al oscilador externo, el interno también puede funcionar en varios modos. El modo de funcionamiento se selecciona de la misma manera que en el oscilador externo - por medio de los bits que forman *Palabra de configuración*. En otras palabras, todo se lleva a cabo dentro del software de PC antes de escribir un programa en el microcontrolador.



OSCILADOR INTERNO EN MODO INTOSCIO

En este modo, el pin OSC1 está disponible para ser utilizado como pin de E/S de propósito general. La señal de frecuencia del oscilador interno dividida por 4 está disponible en el pin OSC2.



OSCILADOR INTERNO EN MODO INTOSCIO

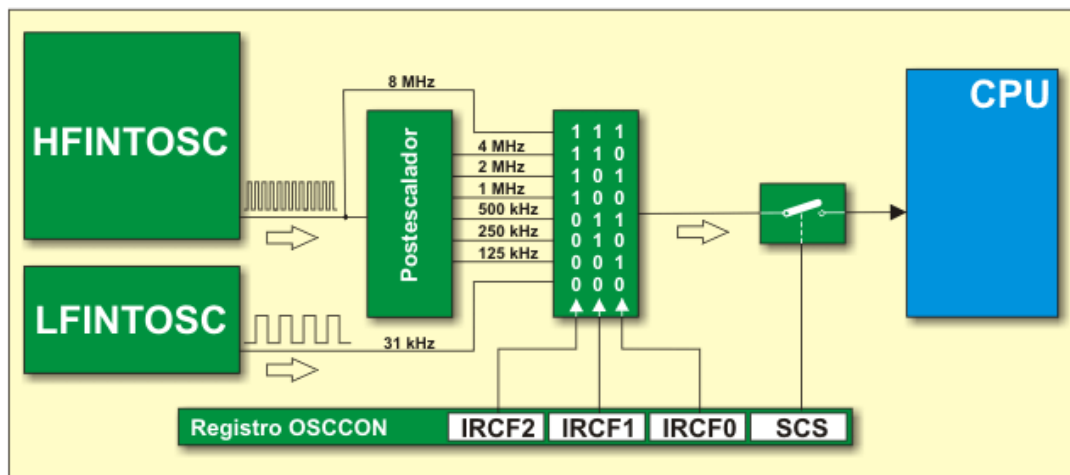
En este modo, los dos pines están disponibles como pines de E/S de propósito general.

CONFIGURACIÓN DEL OSCILADOR INTERNO

El oscilador interno consiste en dos circuitos separados:

1. El oscilador interno de alta frecuencia HFINTOSC está conectado al post-escalador (divisor de frecuencias). Está calibrado de fábrica y funciona a 8 Mhz. Al utilizar el post-escalador, este oscilador puede producir una señal de reloj a una de siete frecuencias. La selección de frecuencia se realiza dentro del software utilizando los pines IRCF2, IRCF1 y IRCF0 del registro OSCCON.

El HFINTOSC está habilitado al seleccionar una de siete frecuencias (entre 8 Mhz y 125 kHz) y poner a uno el bit de la fuente de reloj del sistema (SCS) del registro OSCCON. Como se muestra en la siguiente figura , todo el procedimiento se realiza por medio de los bits del registro OSCCON.



2. El oscilador de baja frecuencia LFINTOSC no está calibrado de fábrica y funciona a 31 kHz. Está habilitado al seleccionar la frecuencia (bits del registro OSCCON) y poner a uno el bit SCS del mismo registro.

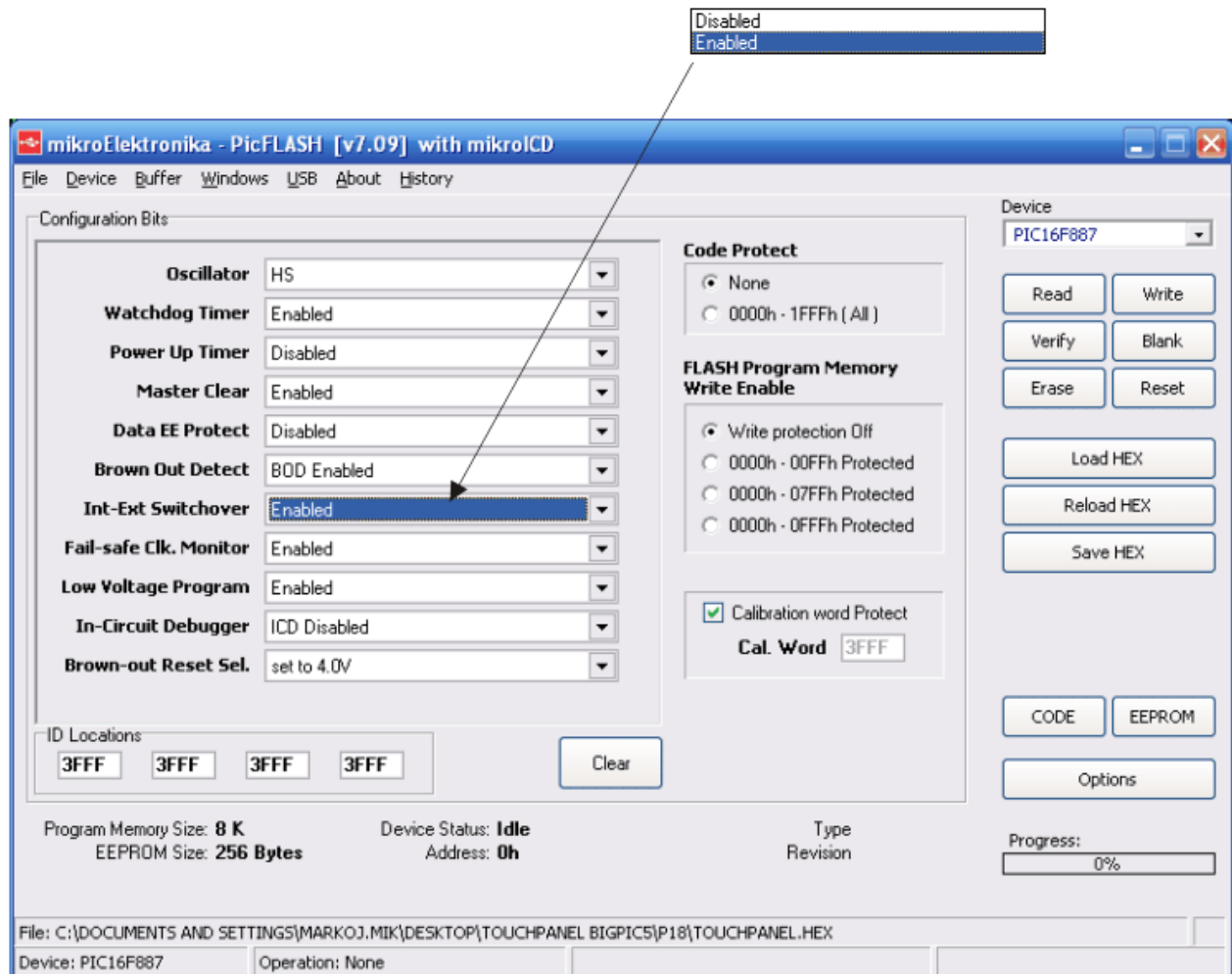
MODO DE CAMBIO AUTOMÁTICO DE VELOCIDAD DE RELOJ (TWO-SPEED CLOCK START-UP MODE)

El modo de cambio automático de velocidad de reloj se utiliza para reducir el consumo de corriente cuando el microcontrolador funciona en modo de reposo. ¿De qué se trata todo esto?

Cuando se configura en modo LP, XT o HS, el oscilador externo se desactiva al pasar a modo de reposo para reducir el consumo de corriente total del dispositivo. Cuando se cumplen las condiciones de "despertamiento", el microcontrolador no se pone a funcionar inmediatamente puesto que tiene que esperar a que se establezca la frecuencia de señal de reloj. Este tiempo muerto dura exactamente 1024 pulsos, después de que el microcontrolador continúa con la ejecución del programa. El caso es que se ejecutan

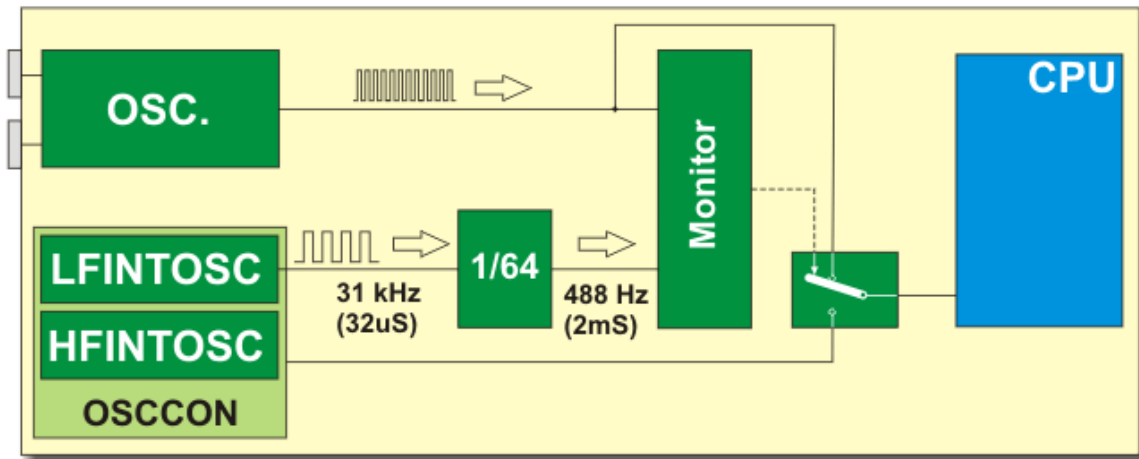
sólo unas pocas instrucciones antes de que el microcontrolador vuelva al modo de reposo.

Eso significa que la mayoría de tiempo así como la mayoría de corriente de baterías se ha perdido en vano. El caso se soluciona utilizando el oscilador interno para ejecutar el programa durante la duración de 1024 pulsos. Tan pronto como se establezca la frecuencia del oscilador externo, él retoma automáticamente "el papel principal". Todo el procedimiento se habilita al poner a uno el bit de palabra de configuración. Para programar el microcontrolador, es necesario seleccionar la opción Int-Ext Switchover (conmutación interna/externa) por software.



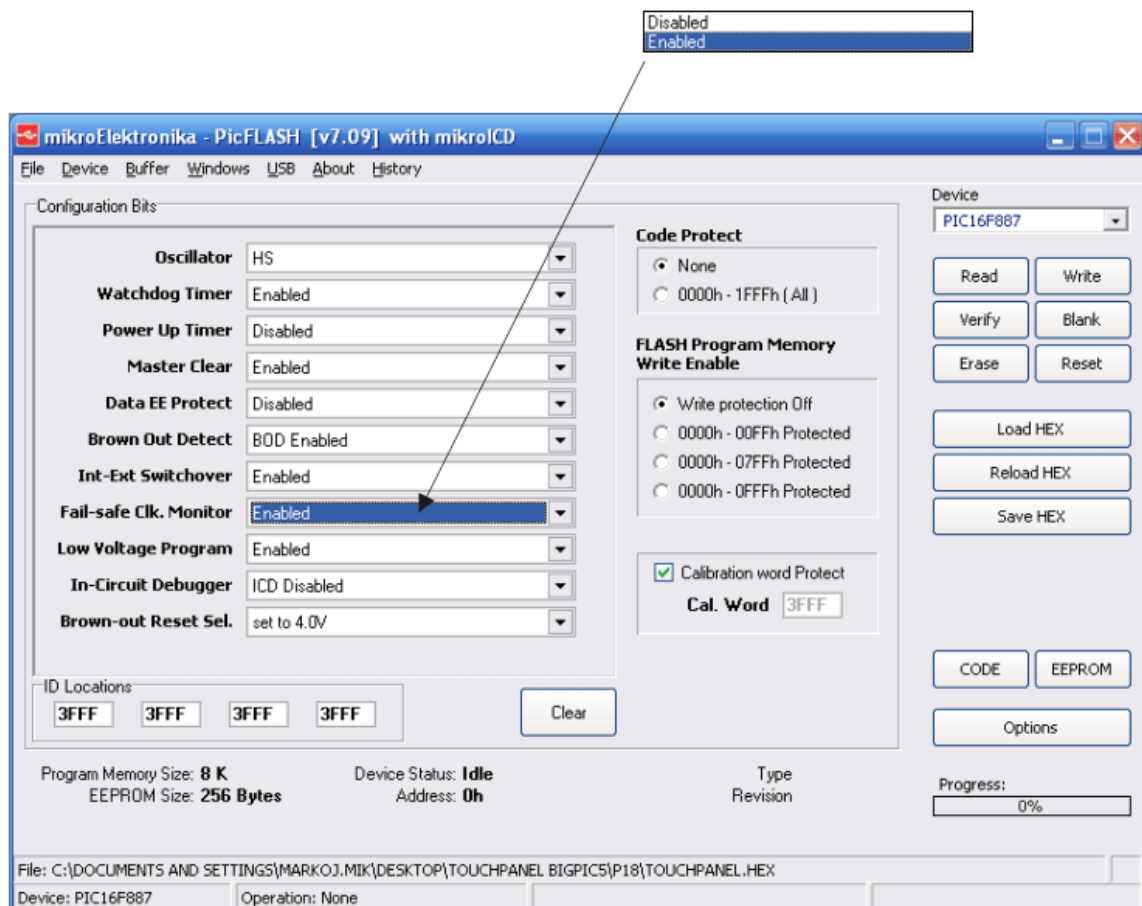
MONITOR PARA DETECTAR UN FALLO DE LA FUENTE DE RELOJ (FAIL-SAFE CLOCK MONITOR)

Como indica su nombre, el monitor para detectar un fallo de la fuente de reloj (Fail-Safe Clock Monitor - FSCM) monitorea el funcionamiento externo y permite al microcontrolador continuar con la ejecución de programa en caso de que el oscilador falle por alguna razón. En tal caso, el oscilador interno toma su función.



El monitor detecta un fallo al comparar las fuentes de reloj interno y externo. Si los pulsos del oscilador externo tardan más de 2mS en llegar, la fuente de reloj será automáticamente cambiada por la interna. Así, el oscilador interno sigue funcionando controlado por los bits del registro OSCCON. Si el bit OSFIE del registro PIE2 está a uno, se producirá una interrupción.

El reloj interno sigue siendo la fuente del reloj del sistema hasta que el dispositivo reinicie con éxito el oscilador externo que vuelve a ser la fuente de reloj del sistema. De manera similar a casos anteriores, este módulo está habilitado al cambiar la palabra de configuración justamente antes de que se inicie el proceso de programar el chip. Esta vez, esto se realiza al seleccionar la opción Fail-Safe Clock Monitor.



Registro OSCTUNE

Los cambios del registro OSCTUNE afectan a la frecuencia del oscilador HFINTOSC, pero no a la frecuencia del LFINTOSC. No hay ninguna indicación de que haya ocurrido desplazamiento de frecuencia durante el funcionamiento del microcontrolador.

OSCTUNE	R/W (0)					R/W (0)				Características
	-	-	-	TUN4	TUN3	TUN2	TUN1	TUN0	Nombre de bit	
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
(0)	Después del reinicio, el bit se pone a cero

TUN4 - TUN0 Frequency Tuning bits. (bits de calibrar la frecuencia). Al combinar estos cinco bits, la frecuencia del oscilador de 8Mhz se reduce o se aumenta. De este modo, las frecuencias obtenidas por la división en el post-escalador cambian también.

TUN4	TUN3	TUN2	TUN1	TUN0	Frecuencia
0	1	1	1	1	Máxima
0	1	1	1	0	
0	1	1	0	1	
0	0	0	0	1	
0	0	0	0	0	Calibrada
1	1	1	1	1	
1	0	0	1	0	
1	0	0	0	1	
1	0	0	0	0	Mínima

La EEPROM es un segmento de memoria separado, que no pertenece a la memoria de programa (ROM), tampoco a la memoria de datos (RAM). Aunque a estas localidades de memoria no se les puede acceder fácil y rápidamente, su propósito es insustituible. Los datos almacenados en la EEPROM están permanentemente guardados incluso al apagar la fuente de alimentación, y pueden ser cambiados en cualquier momento. Por estas características excepcionales cada byte de la EEPROM se considera valioso.

MEMORIA EEPROM

El microcontrolador PIC16F887 dispone de 256 localidades de memoria EEPROM controlados por los bits de los siguientes registros:

- EECON1 (registro de control);
- EECON2 (registro de control);
- EEDAT (almacena los datos listos para escritura y lectura); y
- EEADR (almacena la dirección de la EEPROM a la que se accede).

Además, el registro EECON2 no es un registro verdadero, no existe físicamente en el chip. Se utiliza sólo durante la escritura de los datos en la memoria.

Los registros EEDATH y EEADRH se utilizan durante la escritura y lectura de la EEPROM. Los dos se utilizan también durante la escritura y lectura de la memoria de programa (FLASH).

Por considerar esto una zona de riesgo (por supuesto usted no quiere que el microcontrolador borre su propio programa por casualidad), no vamos a discutirlo aquí, no obstante le avisamos que tenga cuidado.

Registro EECON1

	R/W (x)				R/W (x)	R/W (0)	R/S (0)	R/S (0)	Características
EECON1	EEPGD	-	-	-	WRERR	WREN	WR	RD	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
R	Bit de lectura
S	Bit se puede poner sólo a uno
(0)	Después del reinicio, el bit se pone a cero
(x)	Después del reinicio, el estado de bit es desconocido

EEPGD - Program/Data EEPROM Select bit (bit de selección de memorias)

- 1 - Acceso a la memoria Flash de programa.
- 0 - Acceso a la memoria de datos EEPROM.

WRERR - EEPROM Error Flag bit (bit de error de escritura)

- 1 - Se produce un error de escritura de forma prematura y ha ocurrido un error.
- 0 - Se ha completado la operación de escritura.

WREN - EEPROM Write Enable bit (bit de habilitación de escritura)

- 1 - Escritura de datos en la EEPROM habilitada.
- 0 - Escritura de datos en la EEPROM deshabilitada.

WR - Write Control bit (bit de control de escritura)

- 1 - Se ha iniciado una operación de escritura de datos en la EEPROM.
- 0 - Se ha completado una operación de escritura de datos en la EEPROM.

RD - Read Control bit (bit de control de lectura)

- 1 - Inicia una lectura de la memoria EEPROM.
- 0 - Lectura de la memoria EEPROM deshabilitada.

LECTURA DE LA MEMORIA EEPROM

Para leer los datos de la memoria EEPROM, siga los siguientes pasos:

- **Paso 1:** Escribir la dirección (00h - FFh) en el registro EEADR.
- **Paso 2:** Seleccionar el bloque de memoria EEPROM al poner a cero el bit EEPGD del registro EECON1.
- **Paso 3:** Poner a uno el bit RD del mismo registro para leer el contenido de la localidad.
- **Paso 4:** El dato se almacena en el registro EEDAT y está listo para su uso.

El siguiente ejemplo muestra el procedimiento anteriormente descrito al escribir un programa en lenguaje ensamblador:

```
BSF STATUS,RP1 ;
BCF STATUS,RP0 ; Acceder al banco 2
MOVF ADDRESS,W ; Mover la dirección al registro W
MOVWF EEADR ; Escribir la dirección
BSF STATUS,RP0 ; Acceder al banco 3
BCF EECON1,EEPGD ; Seleccionar la EEPROM
BSF EECON1,RD ; Leer los datos
BCF STATUS,RP0 ; Acceder al banco 2
MOVF EEDATA,W ; Dato se almacena en el registro W
```

La misma secuencia de programa escrita en C se parece a lo siguiente:

```
W = EEPROM_Read(ADDRESS);
```

Las ventajas del uso del lenguaje C se han hecho más obvias, no lo cree?

ESCRITURA EN LA MEMORIA EEPROM

Antes de escribir los datos en la memoria EEPROM es necesario escribir la dirección en el registro EESADR y los datos en el registro EESAT. Sólo ha quedado seguir a una secuencia especial para iniciar la escritura para cada byte. Durante el proceso de escritura las interrupciones deben estar deshabilitadas.

El ejemplo que sigue muestra el procedimiento anteriormente descrito al escribir un programa en lenguaje ensamblador:

```
BSF STATUS,RP1
BSF STATUS,RP0
```

```

BTFSC EECON,WR1 ; Esperar a que se complete la escritura anterior
GOTO $-1 ;
BCF STATUS,RP0 ; Banco 2
MOVF ADDRESS,W ; Mover la dirección a W
MOVWF EEADR ; Escribir la dirección
MOVF DATA,W ; Mover los datos a W

```

```

MOVWF EEDATA ; Escribir los datos
BSF STATUS,RP0 ; Banco 3
BCF EECON1,EEPGD ; Seleccionar la EEPROM
BSF EECON1,WREN ; Escritura a la EEPROM habilitada
BCF INCON,GIE ; Todas las interrupciones deshabilitadas

```

```

MOVLW 55h
MOVWF EECON2
MOVLW AAh
MOVWF EECON2
BSF EECON1,WR

```

```

BSF INTCON,GIE ; Interrupciones habilitadas
BCF EECON1,WREN ; Escritura a la EEPROM deshabilitada

```

La misma secuencia de programa escrita en C se parece a lo siguiente:

```
W = EEPROM_Write(ADDRESS, W);
```

No hace falta comentar nada.

Vamos a hacerlo en mikroC...

// El ejemplo muestra cómo utilizar la librería EEPROM en el compilador mikroC PRO for PIC.

```
char ii; // La variable ii utilizada en el bucle
```

```

void main(){
    ANSEL = 0;           // Configuración de los pines AN como E/S digitales
    ANSELH = 0;
    PORTB = 0;
    PORTC = 0;
    PORTD = 0;
    TRISB = 0;
    TRISC = 0;
    TRISD = 0;
    for(ii = 0; ii < 32; ii++) // Llenar el búfer con los datos
        EEPROM_Write(0x80+ii, ii); // Escribir los datos en la dirección 0x80+ii

    EEPROM_Write(0x02,0xAA); // Escribir un dato en la dirección 2 de la EEPROM
    EEPROM_Write(0x50,0x55); // Escribir un dato en la dirección 0x50

    // de la EEPROM
    Delay_ms(1000); // Diodos en los puertos PORTB y PORTC
    PORTB = 0xFF; // para indicar el comienzo de la lectura
    PORTC = 0xFF;
    Delay_ms(1000);
    PORTB = 0x00;
    PORTC = 0x00;
    Delay_ms(1000);
    PORTB = EEPROM_Read(0x02); // Leer los datos de la dirección 2 de la EEPROM y

    // visualizarla en el puerto PORB
    PORTC = EEPROM_Read(0x50); // Leer los datos de la dirección 0x50 de la EEPROM y

```

```
// visualizarla en el puerto PORC
Delay_ms(1000);

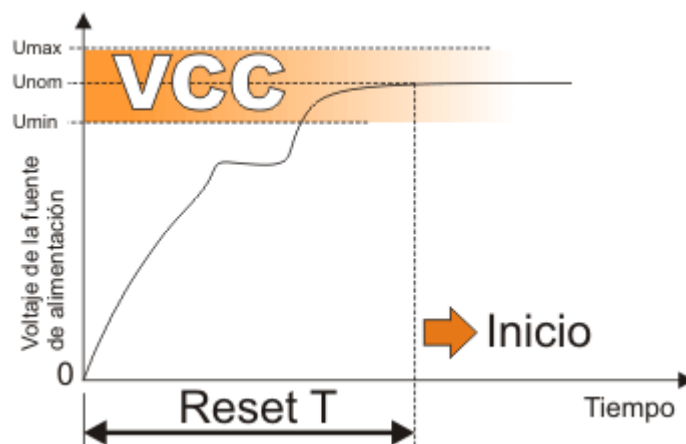
for(ii = 0; ii < 32; ii++) { // Leer el bloque de 32 bytes de la dirección
  PORTD = EEPROM_Read(0x80+ii); // 0x80 y visualizarla en el puerto PORTD

  Delay_ms(250);
}
}
```

A primera vista, basta con encender una fuente de alimentación para hacer funcionar un microcontrolador. A primera vista, basta con apagar una fuente de alimentación para detenerlo. Sólo a primera vista. En realidad, el arranque y el final del funcionamiento son las fases críticas de las que se encarga una señal especial denominada RESET.

¡REINICIO! ¿BLACK-OUT, BROWN-OUT O RUIDOS?

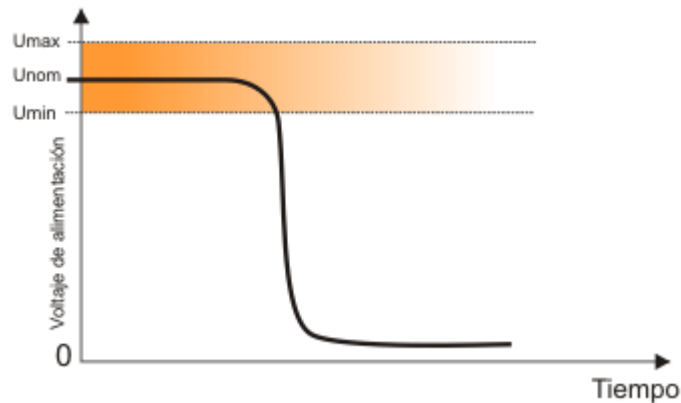
Al producirse un reinicio el microcontrolador detiene su funcionamiento inmediatamente y borra sus registros. Una señal de reinicio se puede generar externamente en cualquier momento (nivel lógico bajo en el pin MCLR). Si se necesita, una señal también puede ser generada por la lógica de control interna. Al encender una fuente de alimentación siempre se produce un reinicio. Por muchos eventos de transición que ocurren al encender una fuente de alimentación (centelleos y fogonazos de contactos eléctricos en interruptores, subida de voltaje lenta, estabilización de la frecuencia de señal de reloj graduada etc.) es necesario proporcionar un cierto tiempo muerto antes de que el microcontrolador se ponga a funcionar. Dos temporizadores internos PWRT y OST se encargan de eso. El PWRT puede estar habilitado/ deshabilitado durante el proceso de escribir un programa. Vamos a ver cómo funciona todo.



Cuando el voltaje de la fuente de alimentación alcanza entre 1.2 y 1.7V, un circuito denominado temporizador de arranque (Power-up timer) mantiene al microcontrolador reiniciado durante unos 72mS. Tan pronto como transcurra el tiempo, otro temporizador denominado temporizador de encendido del oscilador (Oscillator start-up timer) genera otra señal de reinicio durante la duración de 1024 períodos del oscilador de cuarzo. Al expirar el tiempo muerto (marcado con Reset T en la Figura) y al poner a alto el pin MCLR, todas las condiciones se han cumplido y el microcontrolador se pone a ejecutar la primera instrucción en el programa.

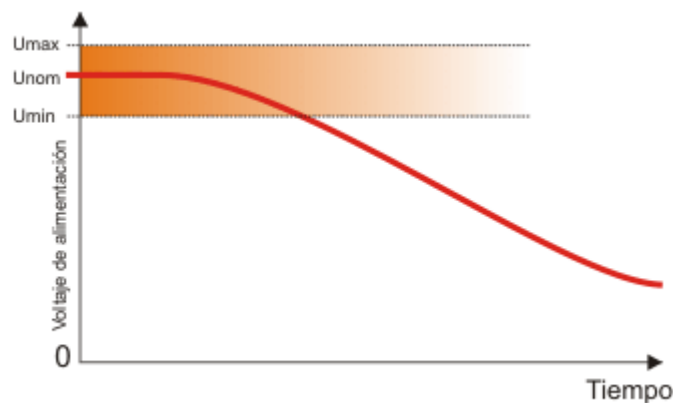
Aparte de este reinicio "controlado" que ocurre al encender una fuente de alimentación, hay dos tipos de reinicio denominados Black-out y Brown-out que pueden producirse durante el funcionamiento del microcontrolador así como al apagar una fuente de alimentación.

REINICIO BLACK-OUT



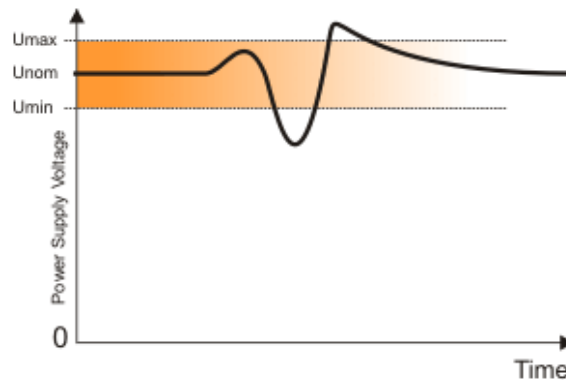
El reinicio black out ocurre al apagar una fuente de alimentación correctamente. El microcontrolador no tiene tiempo para hacer nada imprevisible puesto que el voltaje cae muy rápidamente por debajo de su valor mínimo. En otras palabras, ¡se apaga la luz, las cortinas bajan y el espectáculo ha terminado!

REINICIO BROWN-OUT



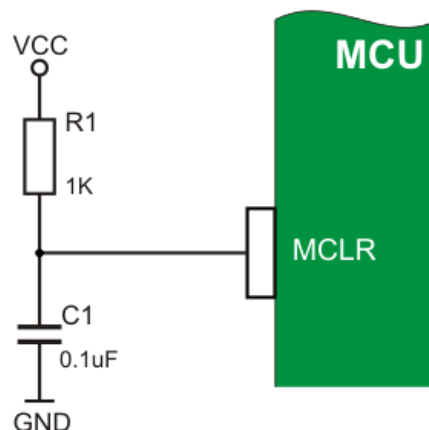
Cuando el voltaje de la fuente de alimentación cae lentamente (un ejemplo típico es descarga de baterías, aunque el microcontrolador experimentaría unas caídas mucho más rápidas como un proceso lento) los componentes internos detienen su funcionamiento gradualmente y ocurre el así llamado reinicio Brownout. En tal caso, antes de que el microcontrolador detenga su funcionamiento completamente, hay un peligro real de que los circuitos que funcionan a frecuencias altas se pongan a funcionar de forma imprevisible. El reinicio brown-out puede causar cambios fatales en el programa ya que se almacena en la memoria flash incorporada en el chip.

RUIDO ELÉCTRICO



Es un tipo especial del reinicio Brownout que ocurre en un ambiente industrial cuando voltaje de alimentación “parpadea” por un momento y cae por debajo del valor mínimo. Aunque es corto, este ruido producido en una línea de conducción eléctrica puede afectar desfavorablemente al funcionamiento del dispositivo.

PIN MCLR



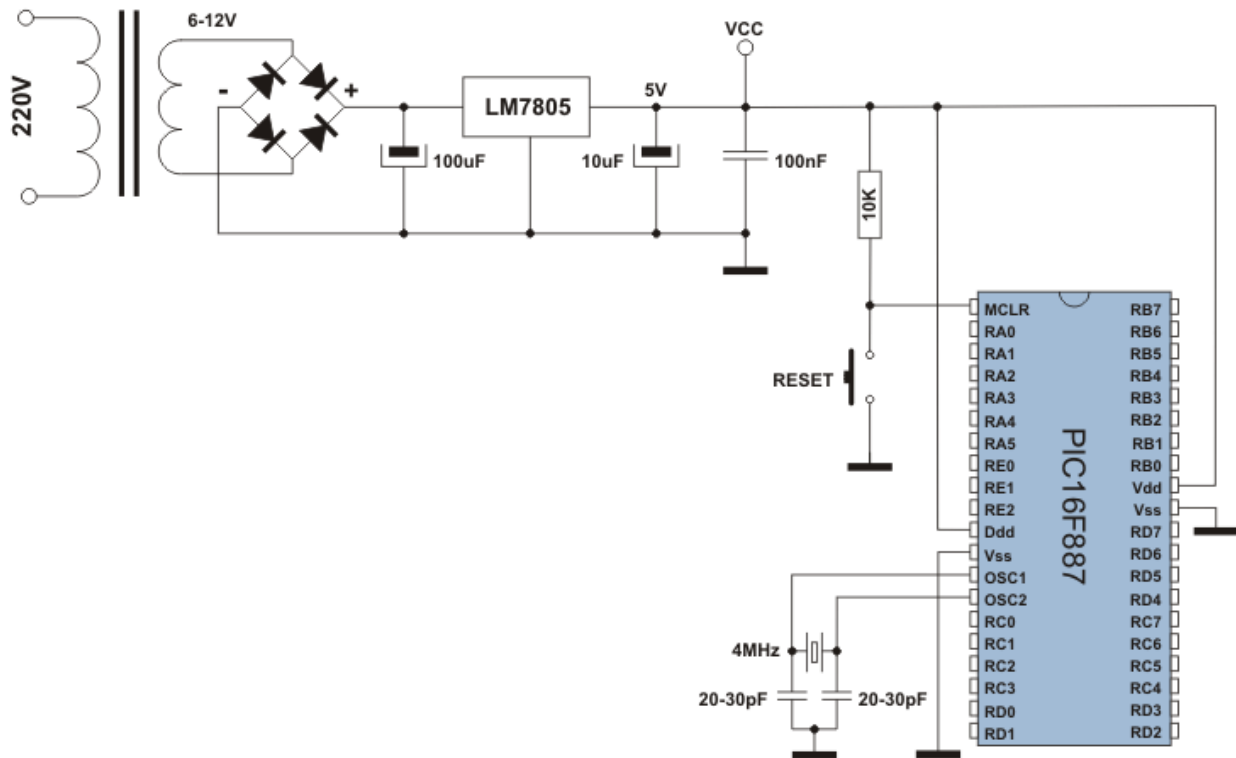
Un cero lógico (0) al pin MCLR causa un reinicio inmediato y regular. Es recomendable conectarlo de la forma mostrada en la Figura a la derecha. La función de los componentes adicionales es de mantener un uno lógico "puro" durante el funcionamiento normal. Si sus valores se seleccionan de manera que proporcionen un nivel lógico alto en el pin después de que haya transcurrido el tiempo muerto reset T , el microcontrolador se pondrá a funcionar inmediatamente. Esto puede ser muy útil cuando se necesita sincronizar el funcionamiento del microcontrolador con los componentes adicionales o con el funcionamiento de varios microcontroladores.

Para evitar posibles errores al producirse el reinicio Brown-out, el PIC 16F887 tiene un “mecanismo de protección” incorporado. Es un circuito simple, pero eficaz que reacciona cada vez que el voltaje de alimentación cae por debajo de 4V (si se utiliza un voltaje de alimentación de 5V) y mantiene este nivel de voltaje por más de 100 microsegundos. Este circuito genera una señal después de que todo el microcontrolador funcionará como si hubiera sido encendido por primera vez.

CONEXIÓN BÁSICA

Para que un microcontrolador funcione apropiadamente es necesario proporcionar lo siguiente:

- Alimentación;
- Señal de reinicio; y
- Señal de reloj.



Como se muestra en la figura anterior, se trata de circuitos simples, pero no tiene que ser siempre así. Si el dispositivo destino se utiliza para controlar las máquinas caras o para mantener funciones vitales, todo se vuelve mucho más complicado.

ALIMENTACIÓN

Aunque el PIC16F887 es capaz de funcionar a diferentes voltajes de alimentación, no es recomendable probar la ley de Murphy. Lo más adecuado es proporcionar un voltaje de alimentación de 5V DC. Este circuito, mostrado en la página anterior, utiliza un regulador de voltaje positivo de tres terminales LM7805. Es un regulador integrado y barato que proporciona una estabilidad de voltaje de alta calidad y suficiente corriente para habilitar el funcionamiento apropiado del controlador y de los periféricos (aquí suficiente significa una corriente de 1A).

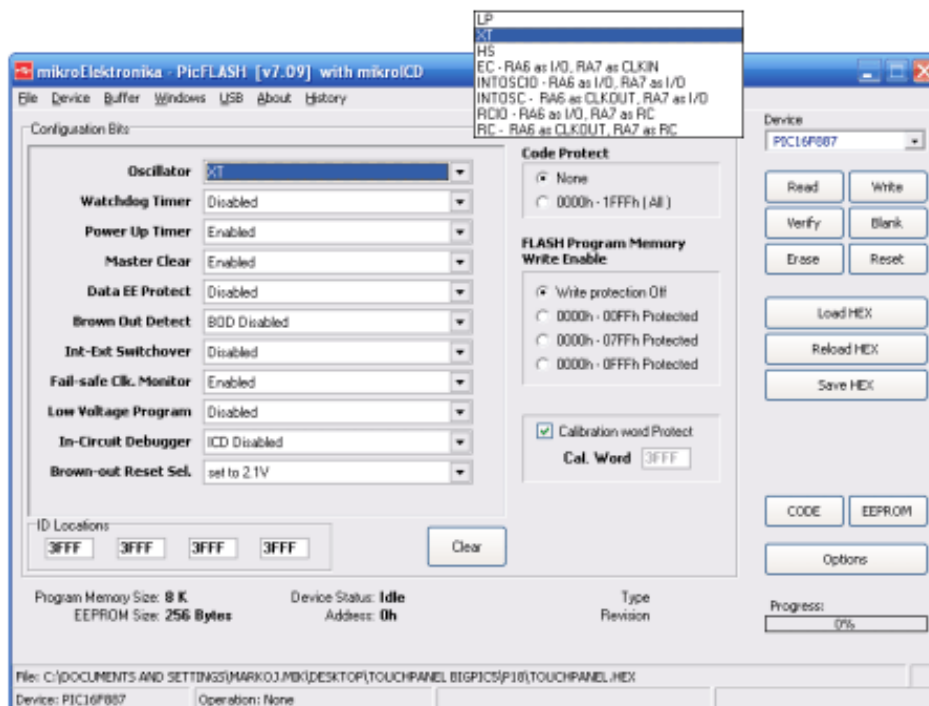
SEÑAL DE REINICIO

Para que un microcontrolador pueda funcionar apropiadamente, un uno lógico (VCC) se debe colocar en el pin de reinicio. El botón de presión que conecta el pin MCLR a GND no es necesario. Sin embargo, este botón casi siempre está proporcionado ya que habilita al microcontrolador volver al modo normal de funcionamiento en caso de que algo salga mal. Al pulsar sobre el botón RESET, el pin MCLR se lleva un voltaje de 0V, el microcontrolador se reinicia y la ejecución de programa comienza desde el principio. Una resistencia de 10k se utiliza para impedir un corto circuito a tierra al presionar este botón.

SEÑAL DE RELOJ

A pesar de tener un oscilador incorporado, el microcontrolador no puede funcionar sin componentes externos que estabilizan su funcionamiento y determinan su frecuencia (velocidad de operación del microcontrolador). Dependiendo de los elementos utilizados así como de las frecuencias el oscilador puede funcionar en cuatro modos diferentes:

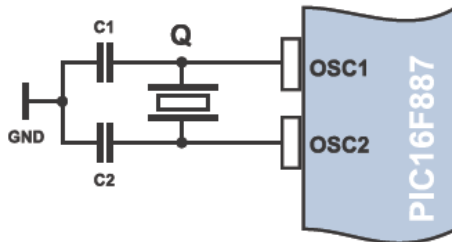
- LP - Cristal de bajo consumo;
- XT - Cristal / Resonador;
- HS - Cristal/Resonador de alta velocidad; y
- RC - Resistencia / Condensador.



¿Por qué son estos modos importantes? Como es casi imposible construir un oscilador estable que funcione a un amplio rango de frecuencias, el microcontrolador tiene que “saber” a qué cristal está conectado, para poder ajustar el funcionamiento de sus componentes internos. Ésta es la razón por la que todos los programas utilizados para escribir un programa en el chip contienen una opción para seleccionar el modo de oscilador. Vea la figura de la izquierda.

Cristal de cuarzo

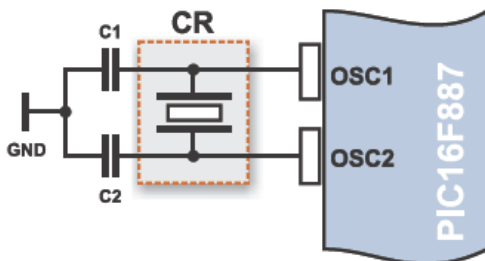
Al utilizar el cristal de cuarzo para estabilizar la frecuencia, un oscilador incorporado funciona a una frecuencia determinada, y no es afectada por los cambios de temperatura y de voltaje de alimentación. Esta frecuencia se etiqueta normalmente en el encapsulado del cristal. Aparte del cristal, los condensadores C1 y C2 deben estar conectados como se muestra en el siguiente esquema. Su capacitancia no es de gran importancia. Por eso, los valores proporcionados en la siguiente tabla se deben tomar como recomendación y no como regla estricta.



Modo	Frecuencia	C1, C2
LP	32 KHz	33pF
	200 KHz	15pF
XT	200 KHz	47-68 pF
	1 MHz	15 pF
	4 MHz	15 pF
HS	4 MHz	15 pF
	8 MHz	15-33 pF
	20 MHz	15-33 pF

Resonador cerámico

Un resonador cerámico es más barato y muy similar a un cuarzo por la función y el modo de funcionamiento. Por esto, los esquemas que muestran su conexión al microcontrolador son idénticos. No obstante, los valores de los condensadores difieren un poco debido a las diferentes características eléctricas. Refiérase a la tabla que está a continuación.



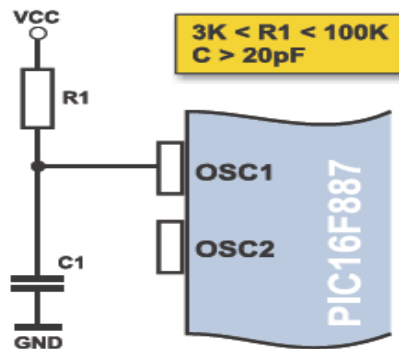
Modo	Frecuencia	C1, C2
XT	455 KHz	68-100 pF
	2 MHz	15-68 pF
	4 MHz	15-68 pF
HS	8 MHz	10-68 pF
	16 MHz	10-22 pF

Estos resonadores se conectan normalmente a los osciladores en caso de que no sea necesario proporcionar una frecuencia extremadamente precisa.

Oscilador RC

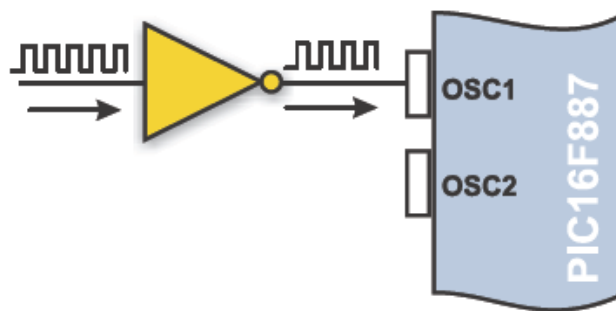
Si la frecuencia de operación no es de importancia, entonces no es necesario utilizar los componentes caros y adicionales para la estabilización. En vez de eso, basta con utilizar una simple red RC, mostrada en la siguiente figura. Como aquí es utilizada sólo la entrada del oscilador local, la señal de reloj con la frecuencia $F_{osc}/4$ aparecerá en el pin OSC2.

Ésta es la frecuencia de operación del microcontrolador, o sea la velocidad de ejecución de instrucciones.



Oscilador externo

Si se requiere sincronizar el funcionamiento de varios microcontroladores o si por alguna razón no es posible utilizar ninguno de los esquemas anteriores, una señal de reloj se puede generar por un oscilador externo. Refiérase a la siguiente figura.



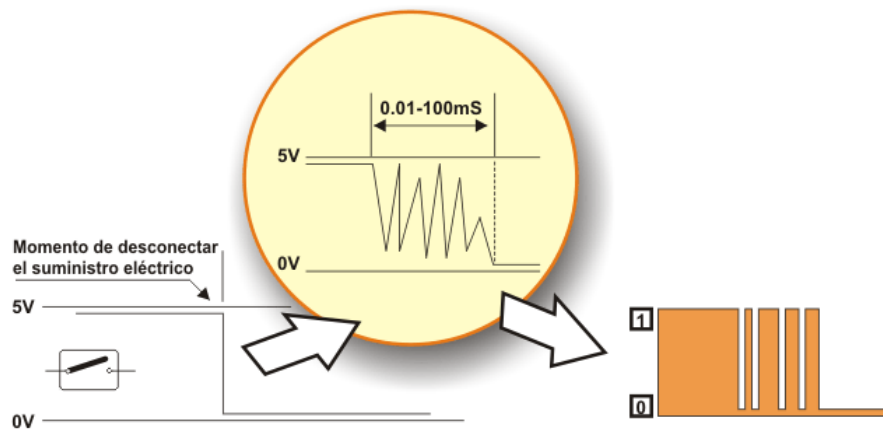
Apesar del hecho de que el microcontrolador es un producto de la tecnología moderna, no es tan útil sin estar conectado a los componentes adicionales. Dicho de otra manera, el voltaje llevado a los pines del microcontrolador no sirve para nada si no se utiliza para llevar a cabo ciertas operaciones como son encender/apagar, desplazar, visualizar etc.

COMPONENTES ADICIONALES

Esta parte trata los componentes adicionales utilizados con más frecuencia en la práctica, tales como resistencias, transistores, diodos LED, visualizadores LED, visualizadores LCD y los circuitos de comunicación RS-232.

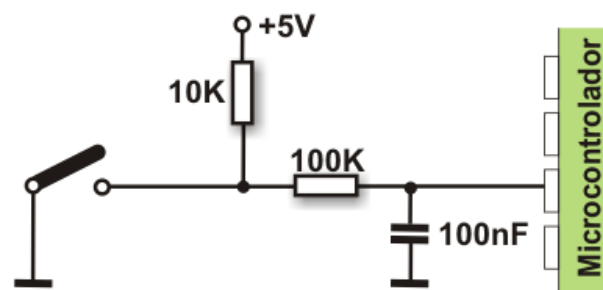
INTERRUPTORES Y BOTONES DE PRESIÓN

Los interruptores y los botones de presión son los dispositivos simples para proporcionar la forma más simple de detectar la aparición de voltaje en un pin de entrada del microcontrolador. No obstante, no es tan simple como parece... Es por un rebote de contacto. El rebote de contacto es un problema común en los interruptores mecánicos.

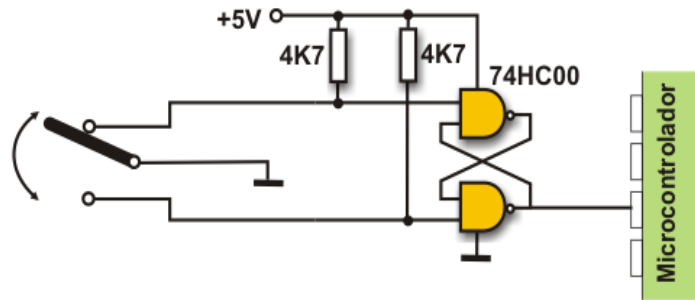


Al tocarse los contactos, se produce un rebote por su inercia y elasticidad. Por consiguiente, la corriente eléctrica es rápidamente pulsada en lugar de tener una clara transición de cero a la corriente máxima. Por lo general, esto ocurre debido a las vibraciones, los desniveles suaves y la suciedad entre los contactos. Este efecto no se percibe normalmente al utilizar estos componentes en la vida cotidiana porque el rebote ocurre demasiado rápido para afectar a la mayoría de los dispositivos eléctricos. Sin embargo, pueden surgir problemas en algunos circuitos lógicos que responden lo suficientemente rápido de manera que malinterpreten los pulsos producidos al tocarse los contactos como un flujo de datos. De todos modos, el proceso entero no dura mucho (unos pocos micro - o milisegundos), pero dura lo suficiente para que lo detecte el microcontrolador. Al utilizar sólo un botón de presión como una fuente de señal de contador, en casi 100% de los casos ocurren los errores.

El problema se puede resolver con facilidad al conectar un simple circuito RC para suprimir rápidos cambios de voltaje. Como el período del rebote no está definido, los valores de los componentes no están precisamente determinados. En la mayoría de los casos es recomendable utilizar los valores que se muestran en la siguiente figura.



Si se necesita una estabilidad completa, entonces hay que tomar medidas radicales. La salida del circuito, mostrado en la siguiente figura (biestable RS, también llamado flip flop RS), cambiará de estado lógico después de detectar el primer pulso producido por un rebote de contacto. Esta solución es más cara (interruptor SPDT), pero el problema es resuelto.



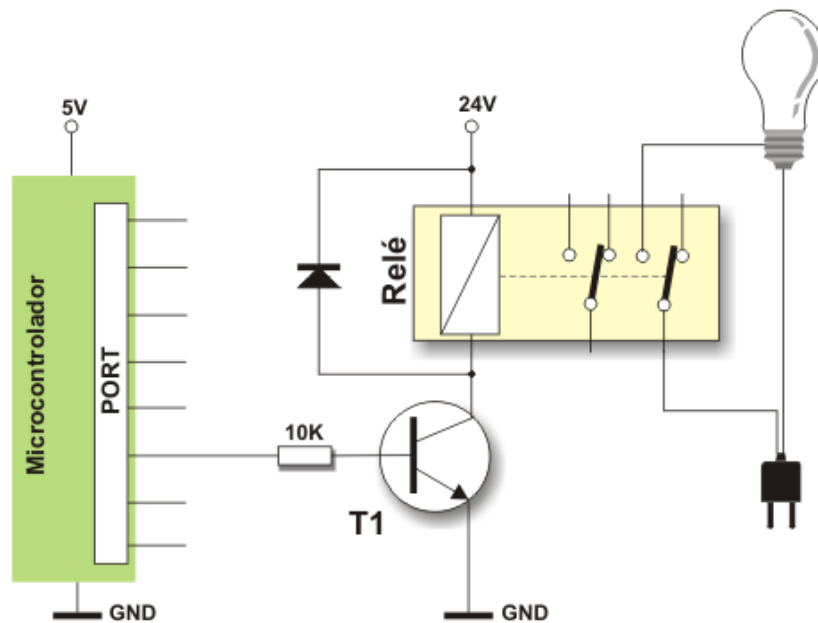
Aparte de estas soluciones de hardware, hay también una simple solución de software. Mientras el programa prueba el estado de circuito lógico de un pin de entrada, si detecta un cambio, hay que probarlo una vez más después de un cierto tiempo de retardo. Si el programa confirma el cambio, esto significa que un interruptor/botón de presión ha cambiado de posición. Las ventajas de esta solución son obvias: es gratuita, se borran los efectos del rebote de contacto y se puede aplicar a los contactos de una calidad más baja también.

RELÉ

Un relé es un interruptor eléctrico que se abre y se cierra bajo el control de otro circuito electrónico. Por eso está conectado a los pines de salida del microcontrolador y utilizado para encender/apagar los dispositivos de alto consumo tales como: motores, transformadores, calefactores, bombillas etc. Estos dispositivos se colocan casi siempre lejos de los componentes sensibles de la placa. Hay varios tipos de relés, pero todos funcionan de la misma manera. Al fluir la corriente por la bobina, el relé funciona por medio de un electromagneto, abriendo y cerrando uno o más conjunto de contactos. Similar a los optoacopladores no hay conexión galvánica (contacto eléctrico) entre los circuitos de entrada y salida. Los relés requieren con frecuencia tanto un voltaje más alto y una corriente más alta para empezar a funcionar. También hay relés miniatura que se pueden poner en marcha por una corriente baja obtenida directamente de un pin del microcontrolador.



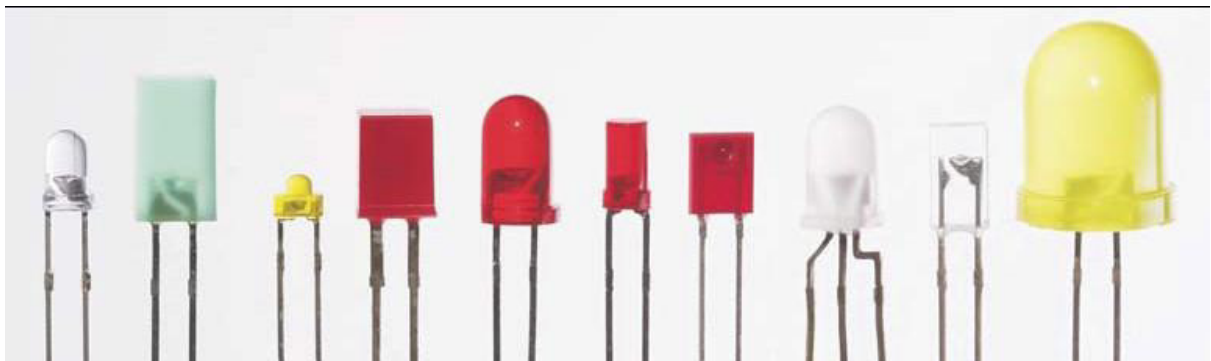
La figura que sigue muestra la solución utilizada con más frecuencia.



Para prevenir la aparición de un alto voltaje de autoinducción, causada por una parada repentina del flujo de corriente por la bobina, un diodo polarizado invertido se conecta en paralelo con la bobina. El propósito de este diodo es de “cortar” este pico de voltaje.

DIODOS LED

Probablemente sepa todo lo que necesita saber sobre los diodos LED, pero también debe pensar en los jóvenes... A ver, ¿cómo destruir un LED? Bueno...muy fácil.



Quemar con rapidez

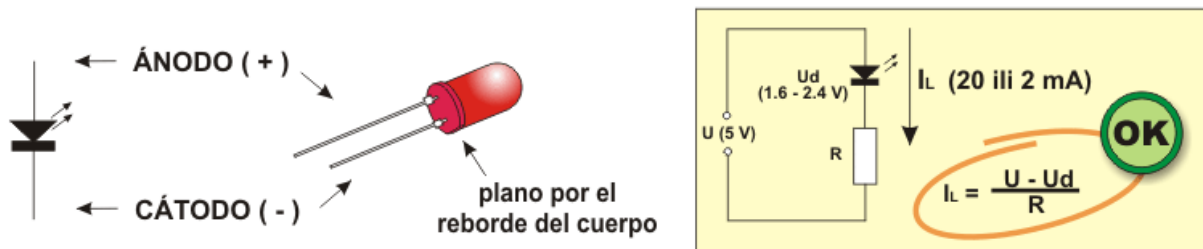
Como cualquier otro diodo, los LEDs tienen dos puntas - un ánodo y un cátodo. Conecte un diodo apropiadamente a la fuente de alimentación y va a emitir luz sin ningún problema. Ponga al diodo al revés y conéctelo a la misma fuente de alimentación (aunque sea por un momento). No emitirá luz - ¡nunca más!

Quemar lentamente

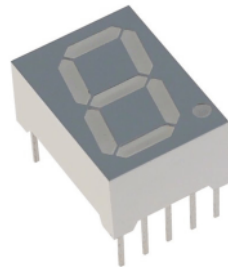
Hay un límite de corriente nominal, o sea, límite de corriente máxima especificada para cada LED que no se deberá exceder. Si eso sucede, el diodo emitirá luz más intensiva, pero sólo por un período corto de tiempo.

Algo para recordar

De manera similar, todo lo que tiene que hacer es elegir una resistencia para limitar la corriente mostrada a continuación. Dependiendo de voltaje de alimentación, los efectos pueden ser espectaculares.



VISUALIZADOR LED



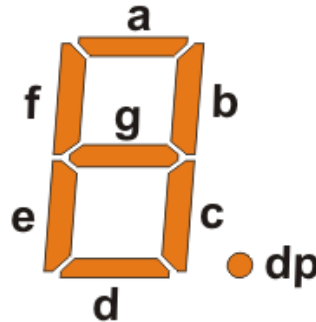
Básicamente, un visualizador LED no es nada más que varios diodos LED moldeados en la misma caja plástica. Hay varios tipos de los visualizadores y algunos de ellos están compuestos por varias docenas de diodos incorporados que pueden visualizar diferentes símbolos. No obstante, el visualizador utilizado con más frecuencia es el visualizador de 7 segmentos. Está compuesto por 8 LEDs. Los siete segmentos de un dígito están organizados en forma de un rectángulo para visualizar los símbolos, mientras que el segmento adicional se utiliza para el propósito de visualizar los puntos decimales. Para simplificar la conexión, los ánodos y los cátodos de todos los diodos se conectan al pin común así que tenemos visualizadores de ánodo común y visualizadores de cátodo común, respectivamente. Los segmentos están etiquetados con letras de a a g y dp, como se muestra en la siguiente figura. Al conectarlos, cada diodo LED se trata por separado, lo que significa que cada uno dispone de su propia resistencia para limitar la corriente.

Aquí le presentamos unas cosas importantes a las que debe prestar atención al comprar un visualizador LED:

- Como hemos mencionado, dependiendo de si ánodos o cátodos están conectados al pin común, tenemos visualizadores de ánodo común y visualizadores de cátodo común. Visto de afuera, parece que no hay ninguna diferencia entre estos visualizadores, pues se le recomienda comprobar cuál se va a utilizar antes de instalarlo.
- Cada pin del microcontrolador tiene un límite de corriente máxima que puede recibir o dar. Por eso, si varios visualizadores están conectados al

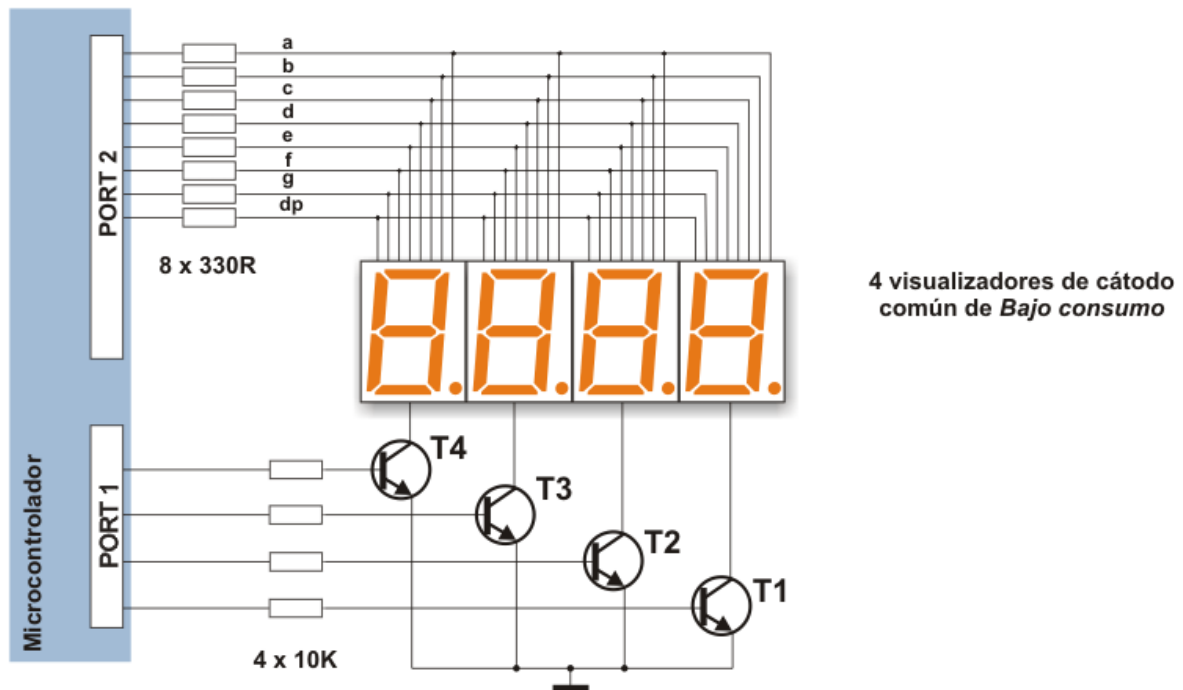
microcontrolador, es recomendable utilizar así llamados LEDs de Bajo consumo que utilizan solamente 2mA para su funcionamiento.

- Los segmentos del visualizador están normalmente etiquetados con letras de a a g, pero no hay ninguna regla estricta a cuáles pines del visualizador estarán conectados. Por eso es muy importante comprobarlo antes de empezar a escribir un programa o diseñar un dispositivo.



Los visualizadores conectados al microcontrolador normalmente ocupan un gran número de los pines de E/S valiosos, lo que puede ser un problema sobre todo cuando se necesita visualizar los números compuestos por varios dígitos. El problema se vuelve más obvio si, por ejemplo, se necesita visualizar dos números de seis dígitos (un simple cálculo muestra que en este caso se necesitan 96 pines de salida). La solución de este problema es denominada multiplexión.

Aquí es cómo se ha hecho una ilusión óptica basada en el mismo principio de funcionamiento como una cámara de película. Un sólo dígito está activo a la vez, pero se tiene la impresión de que todos los dígitos de un número están simultáneamente activos por cambiar tan rápidamente de las condiciones de encendido/apagado.

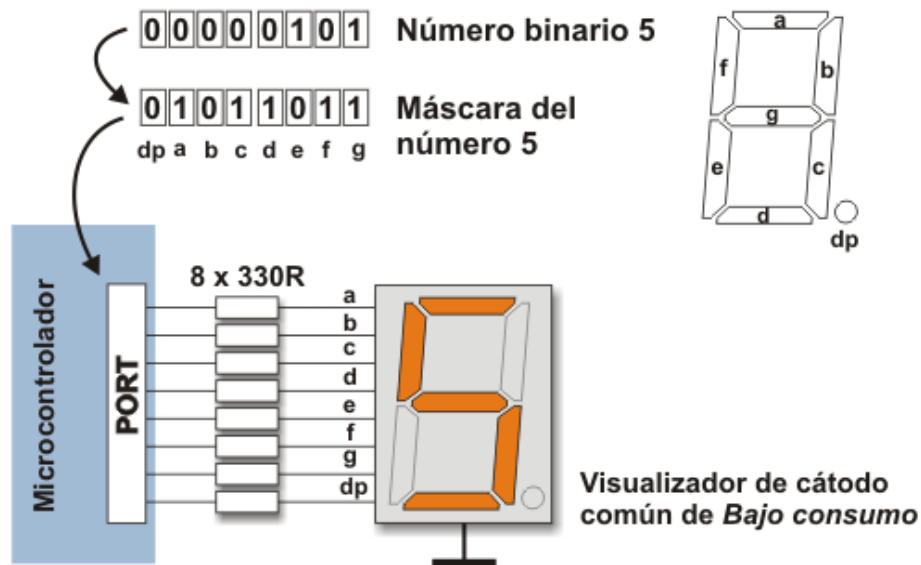


Veamos la figura anterior. Primero se aplica un byte que representa unidades al puerto PORT2 del microcontrolador y se activa el transistor T1 a la vez. Después de poco tiempo, el transistor T1 se apaga, un byte que representa decenas se aplica al puerto PORT2 y el transistor T2 se activa. Este proceso se está repitiendo cíclicamente a alta velocidad en todos los dígitos y transistores correspondientes.

Lo decepcionante es que el microcontrolador es sólo un tipo de computadora miniatura diseñada para interpretar el lenguaje de ceros y unos, lo que se pone de manifiesto al visualizar cualquier dígito. Concretamente, el microcontrolador no conoce cómo son unidades, decenas, centenas, ni diez dígitos a los que estamos acostumbrados. Por esta razón, cada número a visualizar debe pasar por el siguiente procedimiento:

Antes que nada, un número de varios dígitos debe ser dividido en unidades, centenas etc. en una subrutina específica. Luego, cada de estos dígitos se debe almacenar en los bytes particulares. Los dígitos se hacen reconocibles al realizar "enmascaramiento". En otras palabras, el formato binario de cada dígito se sustituye por una combinación diferente de los bits por medio de una subrutina simple. Por ejemplo, el dígito 8 (0000 1000) se sustituye por el número binario 0111 1111 para activar todos los LEDs que visualizan el número 8. El único diodo que queda inactivo aquí está reservado para el punto decimal.

Si un puerto del microcontrolador está conectado al visualizador de tal manera que el bit 0 active el segmento 'a', el bit 1 active el segmento 'b', el bit 2 active el segmento 'c' etc, entonces la tabla que sigue muestra la "máscara" para cada dígito.



Dígitos a visualizar	Segmentos del visualizador							
	dp	a	b	C	d	e	f	g
0	0	1	1	1	1	1	1	0
1	0	0	1	1	0	0	0	0
2	0	1	1	0	1	1	0	1
3	0	1	1	1	1	0	0	1
4	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1
6	0	1	0	1	1	1	1	1
7	0	1	1	1	0	0	0	0
8	0	1	1	1	1	1	1	1
9	0	1	1	1	1	0	1	1

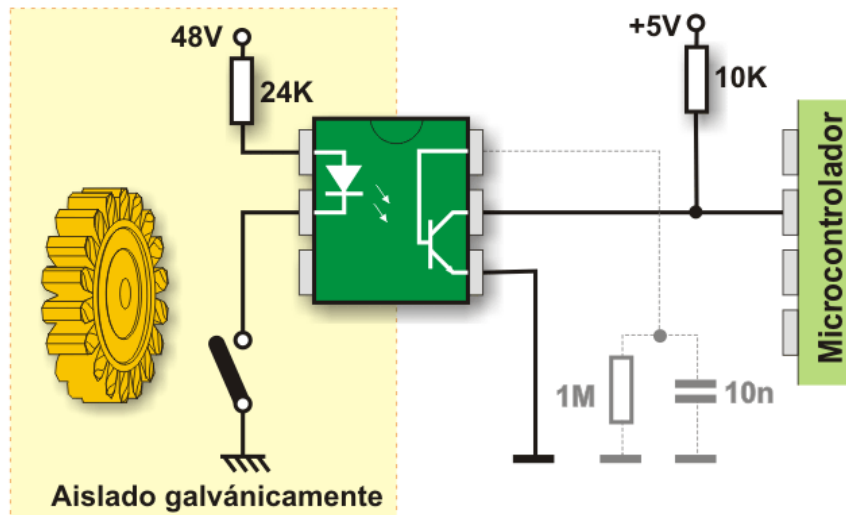
Además de los dígitos de 0 a 9, hay algunas letras -A, C, E, J, F, U, H, L, b, c, d, o, r, t - que se pueden visualizar al enmascarar.

En caso de que se utilicen los visualizadores de ánodo común, todos los unos contenidos en la tabla anterior se deben sustituir por ceros y viceversa. Además, los transistores PNP se deben utilizar como controladores.

OPTOACOPLADORES

Un optoacoplador es un dispositivo frecuentemente utilizado para aislar galvánicamente el microcontrolador de corriente o voltaje potencialmente peligroso de su entorno. Los optoacopladores normalmente disponen de una, dos o cuatro fuentes de luz (diodos LED) en su entrada mientras que en su salida, frente a los diodos, se encuentra el mismo número de los elementos sensibles a la luz (foto-transistores, foto-tiristores, foto-triacs). El punto es que un optoacoplador utiliza una corta ruta de transmisión óptica para transmitir una señal entre los elementos de circuito, que están aislados eléctricamente. Este aislamiento tiene sentido sólo si los diodos y los elementos foto-sensitivos se alimentan por separado.

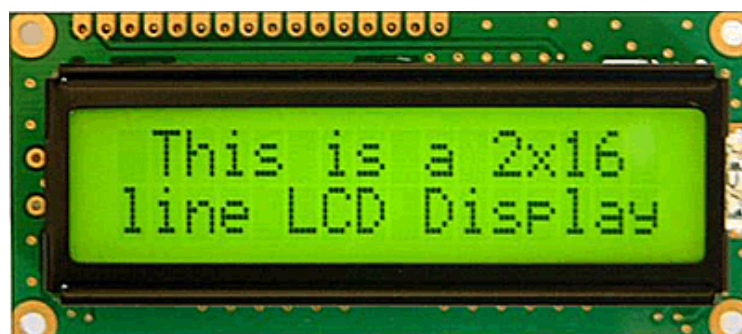
Así, el microcontrolador y los componentes adicionales y caros están completamente protegidos de alto voltaje y ruidos que son la causa más frecuente de destrucción, daño y funcionamiento inestable de los dispositivos electrónicos en la práctica. Los optoacopladores utilizados con más frecuencia son aquéllos con foto-transistores en sus salidas. En los optoacopladores con la base conectada al pin 6 interno (también hay optoacopladores sin ella), la base puede quedarse desconectada.



La red R/C representada por una línea quebrada en la figura anterior indica una conexión opcional de la base de transistores dentro del optoacoplador, que reduce los efectos de ruidos al eliminar los pulsos muy cortos.

VISUALIZADOR LCD

Este componente está específicamente fabricado para ser utilizado con los microcontroladores, lo que significa que no se puede activar por los circuitos integrados estándar. Se utiliza para visualizar los diferentes mensajes en un visualizador de cristal líquido miniatura. El modelo descrito aquí es el más utilizado en la práctica por su bajo precio y grandes capacidades. Está basado en el microcontrolador HD44780 (Hitachi) integrado y puede visualizar mensajes en dos líneas con 16 caracteres cada una. Puede visualizar todas las letras de alfabeto, letras de alfabeto griego, signos de puntuación, símbolos matemáticos etc. También es posible visualizar símbolos creados por el usuario. Entre otras características útiles es el desplazamiento automático de mensajes (a la izquierda y a la derecha), aparición del cursor, retroiluminación LED etc.



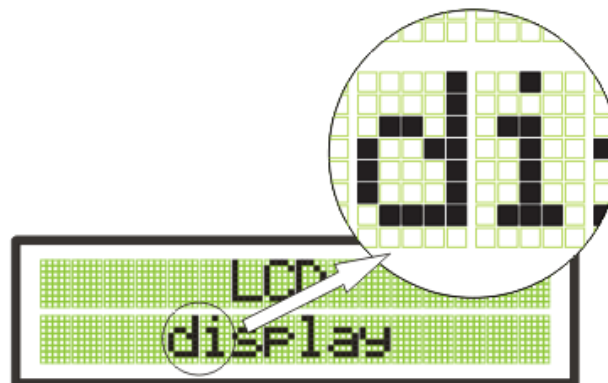
Pines del visualizador LCD

A lo largo de un lado de una placa impresa pequeña del visualizador LCD se encuentran los pines que le permiten estar conectado al microcontrolador. Hay 14 pines en total marcados con números (16 si hay retroiluminación). Su función se muestra en la tabla que sigue:

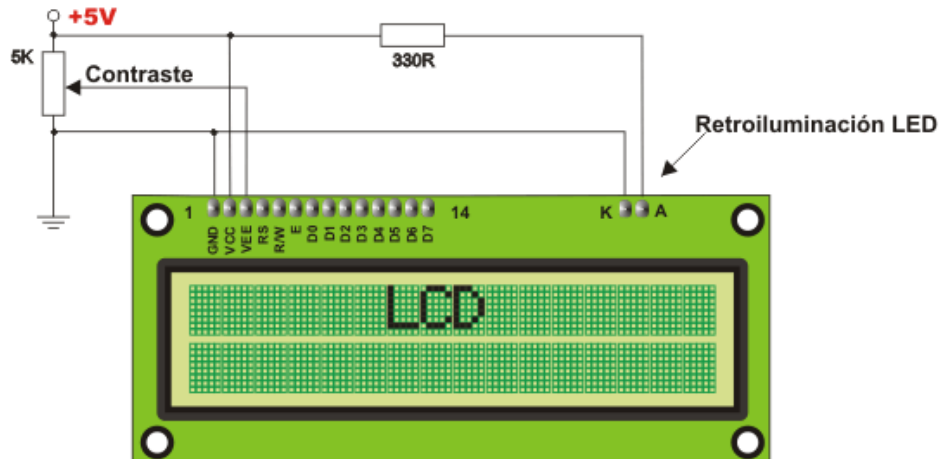
Función	Número	Nombre	Estado lógico	Descripción
Tierra	1	Vss	-	0V
Alimentación	2	Vdd	-	+5V
Contraste	3	Vee	-	0 - Vdd
	4	RS	0	D0 – D7 considerados como comandos
			1	D0 – D7 considerados como datos
Control de funcionamiento	5	R/W	0	Escribir los datos (del microcontrolador al LCD)
			1	Leer los datos (del LCD al microcontrolador)
Datos / comandos	6	E	0	Acceso al visualizador LCD deshabilitado
			1	Funcionamiento normal
				Datos/comandos se están transmitiendo al LCD
	7	D0	0/1	Bit 0 LSB
	8	D1	0/1	Bit 1
	9	D2	0/1	Bit 2
	10	D3	0/1	Bit 3
	11	D4	0/1	Bit 4
12	D5	0/1	Bit 5	
13	D6	0/1	Bit 6	
14	D7	0/1	Bit 7 MSB	

Pantalla LCD

Una pantalla LCD puede visualizar dos líneas con 16 caracteres cada una. Cada carácter consiste en 5x8 o 5x11 píxeles. Este libro cubre un visualizador de 5x8 píxeles que es utilizado con más frecuencia.



El contraste del visualizador depende del voltaje de alimentación y de si los mensajes se visualizan en una o dos líneas. Por esta razón, el voltaje variable 0-V_{dd} se aplica al pin marcado como V_{ee}. Un potenciómetro trimer se utiliza con frecuencia para este propósito. Algunos de los visualizadores LCD tienen retroiluminación incorporada (diodos LED azules o verdes). Al utilizarlo durante el funcionamiento, se debe de conectar una resistencia en serie a uno de los pines para limitar la corriente (similar a diodos LED).



Si no hay caracteres visualizados o si todos los caracteres están oscurecidos al encender el visualizador, lo primero que se debe hacer es comprobar el potenciómetro para ajustar el contraste. ¿Está ajustado apropiadamente? Lo mismo se aplica si el modo de funcionamiento ha sido cambiado (escribir en una o en dos líneas).

Memoria LCD

El visualizador LCD dispone de tres bloques de memoria:

- DDRAM Display Data RAM (RAM de datos de visualización);
- CGRAM Character Generator RAM (generador de caracteres RAM); y
- CGROM Character Generator ROM (generador de caracteres ROM)

Memoria DDRAM

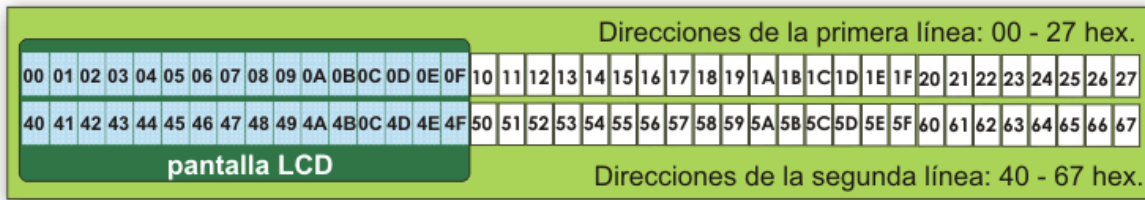
La memoria DDRAM se utiliza para almacenar los caracteres a visualizar. Tiene una capacidad de almacenar 80 caracteres. Algunas localidades de memoria están directamente conectadas a los caracteres en el visualizador.

Todo funciona muy simple: basta con configurar el visualizador para incrementar direcciones automáticamente (desplazamiento a la derecha) y establecer la dirección inicial para el mensaje que se va a visualizar (por ejemplo 00 hex).

Luego, todos los caracteres enviados por las líneas D0-D7 se van a visualizar en el formato de mensaje al que nos hemos acostumbrado - de la izquierda a la derecha. En este caso, la visualización empieza por el primer campo de la primera línea ya que la dirección inicial es 00hex. Si se envía más de 16 caracteres, todos se memorizarán, pero sólo los primeros 16 serán visibles. Para visualizar los demás, se debe utilizar el comando shift. Virtualmente, parece como si el visualizador LCD fuera una ventana, desplazándose de la izquierda a la derecha sobre las localidades de memoria con

diferentes caracteres. En realidad, así es cómo se creó el efecto de desplazar los mensajes sobre la pantalla.

Memoria DDRAM



Si se habilita ver el cursor, aparecerá en la localidad actualmente direccionada. En otras palabras, si un carácter aparece en la posición del cursor, se va a mover automáticamente a la siguiente localidad direccionada.

Esto es un tipo de memoria RAM así que los datos se pueden escribir en ella y leer de ella, pero su contenido se pierde irrecuperablemente al apagar la fuente de alimentación.

Memoria CGROM

La memoria CGROM contiene un mapa estándar de todos los caracteres que se pueden visualizar en la pantalla. A cada carácter se le asigna una localidad de memoria:

		4 bits más altos de la dirección															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
4 bits más bajos de la dirección	XXXX 0000	CG RAM (1)		P													
	XXXX 0001	CG RAM (2)		Q													
	XXXX 0010	CG RAM (3)		R													
	XXXX 0011	CG RAM (4)		S													
	XXXX 0100	CG RAM (5)		T													
	XXXX 0101	CG RAM (6)		U													
	XXXX 0110	CG RAM (7)		V													
	XXXX 0111	CG RAM (8)		W													
	XXXX 1000	CG RAM (1)		X													
	XXXX 1001	CG RAM (2)		Y													
	XXXX 1010	CG RAM (3)		Z													
	XXXX 1011	CG RAM (4)		[
	XXXX 1100	CG RAM (5)		1													
	XXXX 1101	CG RAM (6)		2													
	XXXX 1110	CG RAM (7)		3													
	XXXX 1111	CG RAM (8)		4													

Las direcciones de las localidades de memoria CGROM corresponden a los caracteres ASCII. Si el programa que se está actualmente ejecutando llega al comando 'enviar el carácter P al puerto', el valor binario 0101 0000 aparecerá en el puerto. Este valor es el equivalente ASCII del carácter P. Al escribir este valor en un LCD, se visualizará el símbolo de la localidad 0101 0000 de la CGROM. En otras palabras, se visualizará el carácter P. Esto se aplica a todas las letras del alfabeto (minúsculas y mayúsculas), pero no se aplica a los números.

Como se muestra en el mapa anterior, las direcciones de todos los dígitos se desplazan por 48 en relación con sus valores (dirección del dígito 0 es 48, dirección del dígito 1 es 49, dirección del dígito 2 es 50 etc.). Por consiguiente, para visualizar los dígitos correctamente es necesario añadir el número decimal 48 a cada uno antes de enviarlos a un LCD.

ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol		
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	\$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(56	38	8
9	9	TAB	25	19	EM	41	29)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?
ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol			ASCII Hex Symbol		
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	
77	4D	M	93	5D]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

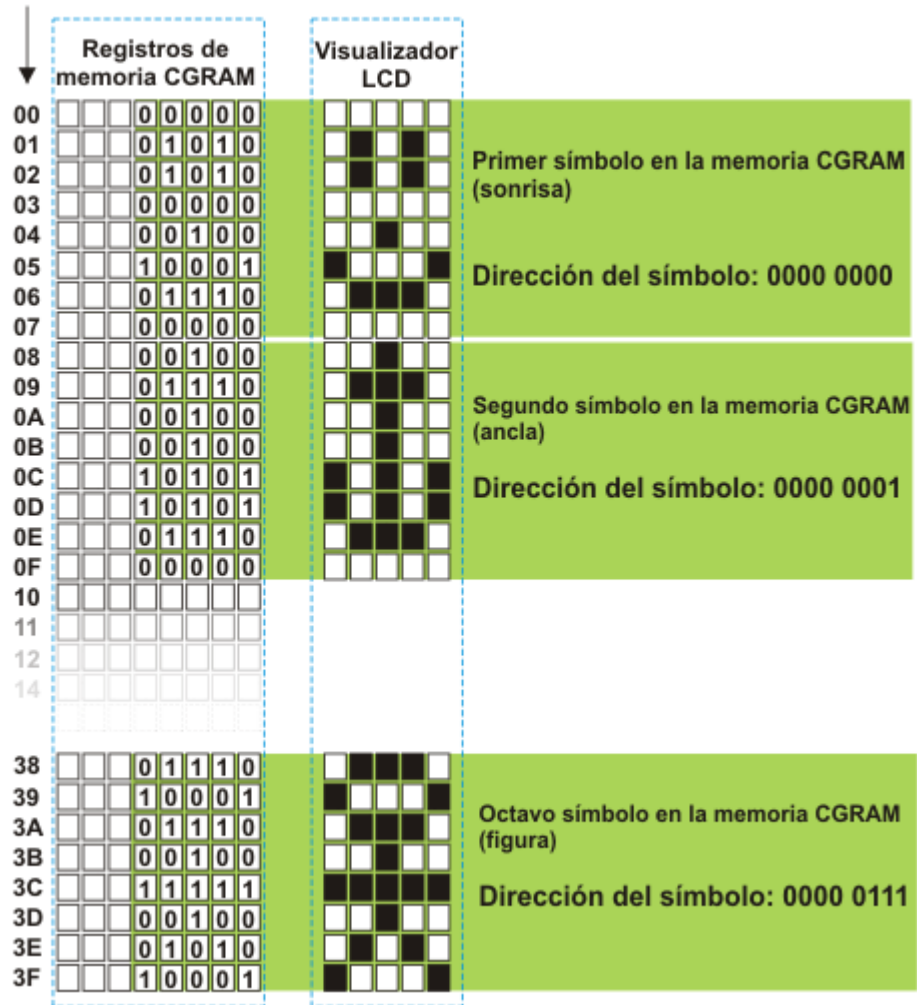
¿Qué es un código ASCII? Desde su aparición hasta hoy en día, las computadoras han sido capaces de reconocer solamente números, y no las letras. Esto significa que todos los datos que una computadora intercambia con un periférico, reconocidos como letras por los humanos, en realidad están en el formato binario (el teclado es un buen ejemplo). En otras palabras, a cada carácter le corresponde la combinación única de ceros y unos. El código ASCII representa una codificación de caracteres basada en el alfabeto inglés. El ASCII especifica una correspondencia entre los símbolos de caracteres estándar y sus equivalentes numéricos.

Memoria CGRAM

Además de los caracteres estándar, el visualizador LCD puede visualizar símbolos definidos por el usuario. Esto puede ser cualquier símbolo de 5x8 píxeles. La memoria RAM denominada CGRAM de 64 bytes lo habilita.

Los registros de memoria son de 8 bits de anchura, pero sólo se utilizan 5 bits más bajos. Un uno lógico (1) en cada registro representa un punto oscurecido, mientras que 8 localidades agrupados representan un carácter. Esto se muestra en la siguiente figura:

Direcciones hex. de los registros



Los símbolos están normalmente definidos al principio del programa por una simple escritura de ceros y unos de la memoria CGRAM así que crean las formas deseadas. Para visualizarlos basta con especificar su dirección. Preste atención a la primera columna en el mapa de caracteres CGROM. No contiene direcciones de la memoria RAM, sino los símbolos de los que se está hablando aquí. En este ejemplo 'visualizar 0' significa visualizar 'sonrisa', 'visualizar 1' significa - visualizar 'ancla' etc.

Comandos básicos del visualizador LCD

Todos los datos transmitidos a un visualizador LCD por las salidas D0-D7 serán interpretados como un comando o un dato, lo que depende del estado lógico en el pin RS:

- **RS = 1** - Los bits D0 - D7 son direcciones de los caracteres a visualizar. El procesador LCD direcciona un carácter del mapa de caracteres y lo visualiza. La dirección DDRAM especifica la localidad en la que se va a visualizar el carácter. Esta dirección se define antes de transmitir el carácter o la dirección del carácter anteriormente transmitido será aumentada automáticamente.
- **RS = 0** - Los bits D0 - D7 son los comandos para ajustar el modo del visualizador.

En la siguiente tabla se muestra una lista de comandos reconocidos por el LCD:

Comando	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Tiempo de ejecución
Borrar el visualizador	0	0	0	0	0	0	0	0	0	1	1.64mS
Poner el cursor al inicio	0	0	0	0	0	0	0	0	1	x	1.64mS
Modo de entrada	0	0	0	0	0	0	0	1	I/D	S	40uS
Activar/desactivar el visualizador	0	0	0	0	0	0	1	D	U	B	40uS
Desplazar el cursor/visualizador	0	0	0	0	0	1	D/C	R/L	x	x	40uS
Modo de funcionamiento	0	0	0	0	1	DL	N	F	x	x	40uS
Establecer la dirección CGRAM	0	0	0	1	Dirección CGRAM					40uS	
Establecer la dirección DDRAM	0	0	1	Dirección CGRAM					40uS		
Leer la bandera "BUSY"(ocupado) (BF)	0	1	BF Dirección CGRAM							-	
Escribir en la CGRAM o en la DDRAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	40uS
Leer la CGRAM o la DDRAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	40uS

I/D 1 = Incremento (por 1)
0 = Decremento (por 1)

R/L 1 = Desplazamiento a la derecha
0 = Desplazamiento a la izquierda

S 1 = Desplazamiento del visualizador activado
0 = Desplazamiento del visualizador desactivado

DL 1 = Bus de datos de 8 bits
0 = Bus de datos de 4 bits

D 1 = Visualizador encendido
0 = Visualizador apagado

N 1 = Visualizador de dos líneas
0 = Visualizador en una línea

U 1 = Cursor activado
0 = Cursor desactivado

F 1 = Carácter de 5x10 puntos
0 = Carácter de 5x7 puntos

B 1 = Parpadeo del cursor encendido
0 = Parpadeo del cursor apagado

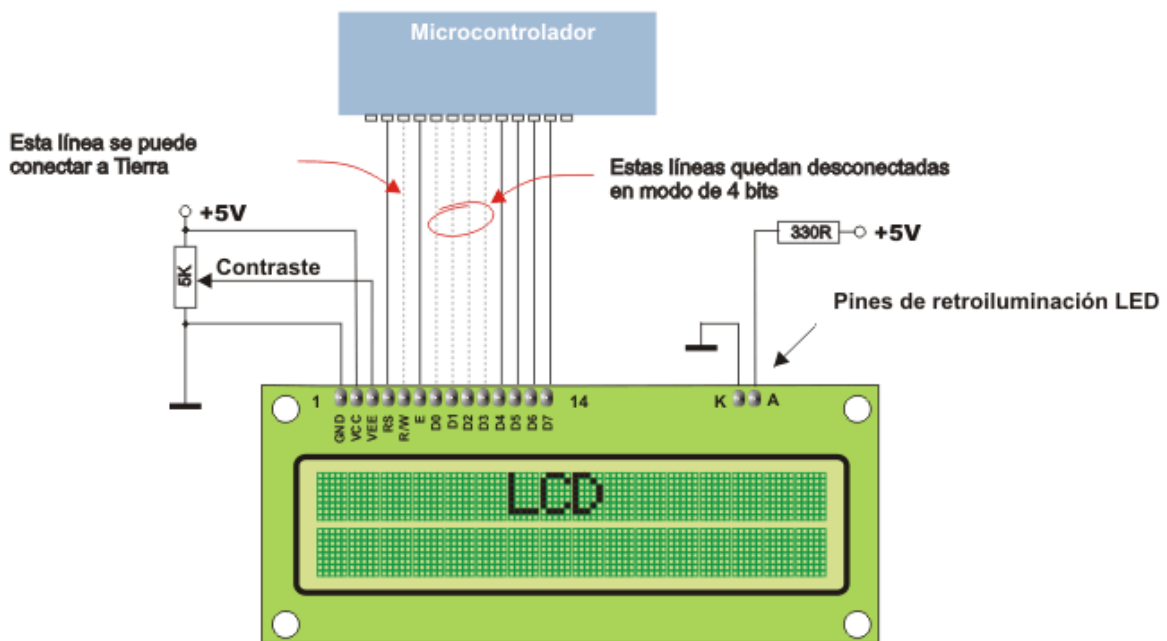
D/C 1 = Desplazamiento del visualizador
0 = Desplazamiento del cursor

¿QUÉ ES UNA BANDERA DE OCUPADO (BUSY FLAG)?

En comparación al microcontrolador, el LCD es un componente extremadamente lento. Por esta razón, era necesario proporcionar una señal que, al ejecutar un comando, indicaría que el visualizador estaba listo para recibir el siguiente dato. Esta señal denominada bandera de ocupado (busy flag) se puede leer de la línea D7. El visualizador está listo para recibir un nuevo dato cuando el voltaje en esta línea es de 0V (BF=0).

Conectar al visualizador LCD

Dependiendo de cuántas líneas se utilizan para conectar un LCD al microcontrolador, hay dos modos de LCD, el de 8 bits y el de 4 bits. El modo apropiado se selecciona en el inicio del funcionamiento en el proceso denominado ‘inicialización’. El modo de LCD de 8 bits utiliza los pines D0-D7 para transmitir los datos, como hemos explicado en la página anterior. El propósito principal del modo de LCD de 4 bits es de ahorrar los valiosos pines de E/S del microcontrolador. Sólo los 4 bits más altos (D4-D7) se utilizan para la comunicación, mientras que los demás pueden quedarse desconectados. Cada dato se envía al LCD en dos pasos - primero se envían 4 bits más altos (normalmente por las líneas D4- D7), y luego los 4 bits más bajos. La inicialización habilita que el LCD conecte e interprete los bits recibidos correctamente.



Pocas veces se leen los datos del LCD (por lo general se transmiten del microcontrolador al LCD) así que, con frecuencia, es posible guardar un pin de E/S de sobra. Es simple, basta con conectar el pin L/E a Tierra. Este “ahorro” del pin tiene su precio. Los mensajes se visualizarán normalmente, pero no será posible leer la bandera de ocupado ya que tampoco es posible leer los datos del visualizador. Afortunadamente, hay una solución simple. Después de enviar un carácter o un comando es importante dar al LCD suficiente tiempo para hacer su tarea. Debido al hecho de que la ejecución de un comando puede durar aproximadamente 1.64mS, el LCD tarda como máximo 2mS en realizar su tarea.

Inicializar al visualizador LCD

Al encender la fuente de alimentación, el LCD se reinicia automáticamente. Esto dura aproximadamente 15mS. Después de eso, el LCD está listo para funcionar. Asimismo, el modo de funcionamiento está configurado por defecto de la siguiente manera:

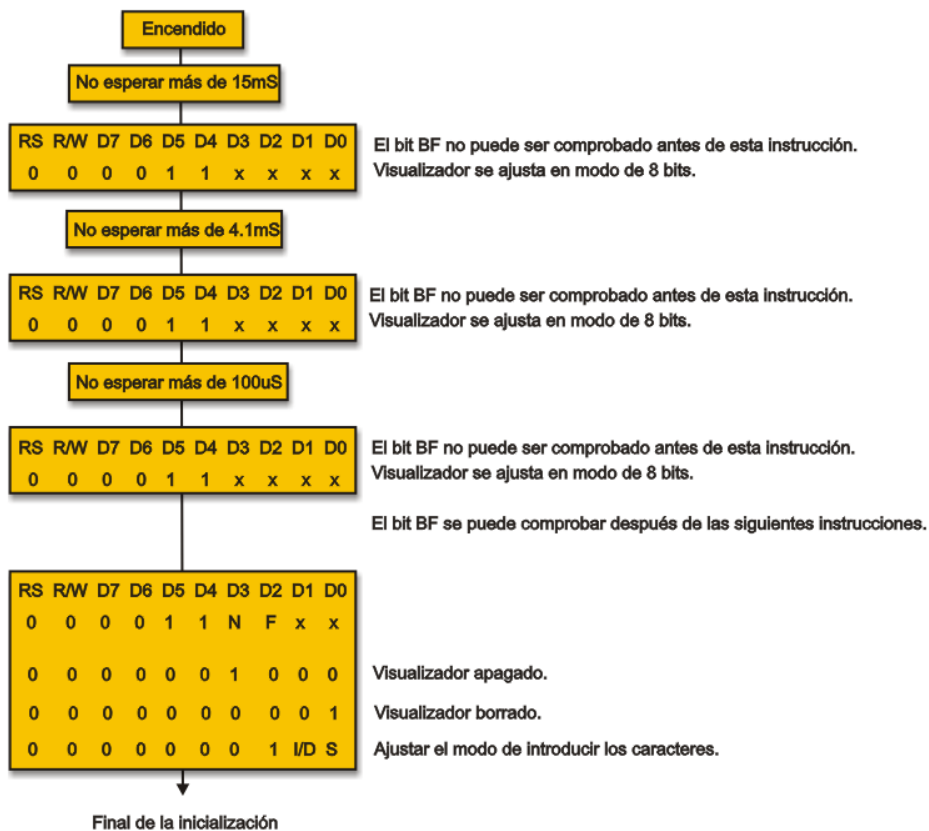
1. Visualizador está borrado.
2. Modo
DL = 1 - Bus de datos de 8 bits

- N = 0 - LCD de una línea
- F = 0 - Carácter de 5 x 8 puntos
- 3. Visualizador/Cursor encendido/apagado
 - D = 0 - Visualizador apagado
 - U = 0 - Cursor apagado
 - B = 0 - Parpadeo del cursor apagado
- 4. Introducción de caracteres
 - ID = 1 Direcciones visualizadas se incrementan automáticamente en 1
 - S = Desplazamiento del visualizador desactivado

Por lo general, el reinicio automático se lleva a cabo sin problemas. ¡En la mayoría de los casos, pero no siempre! Si por cualquier razón, el voltaje de alimentación no llega a su máximo valor en 10mS, el visualizador se pone a funcionar de manera completamente imprevisible. Si la unidad de voltaje no es capaz de cumplir con las condiciones o si es necesario proporcionar un funcionamiento completamente seguro, se aplicará el proceso de inicialización. La inicialización, entre otras cosas, reinicia de nuevo al LCD, al habilitarle un funcionamiento normal.

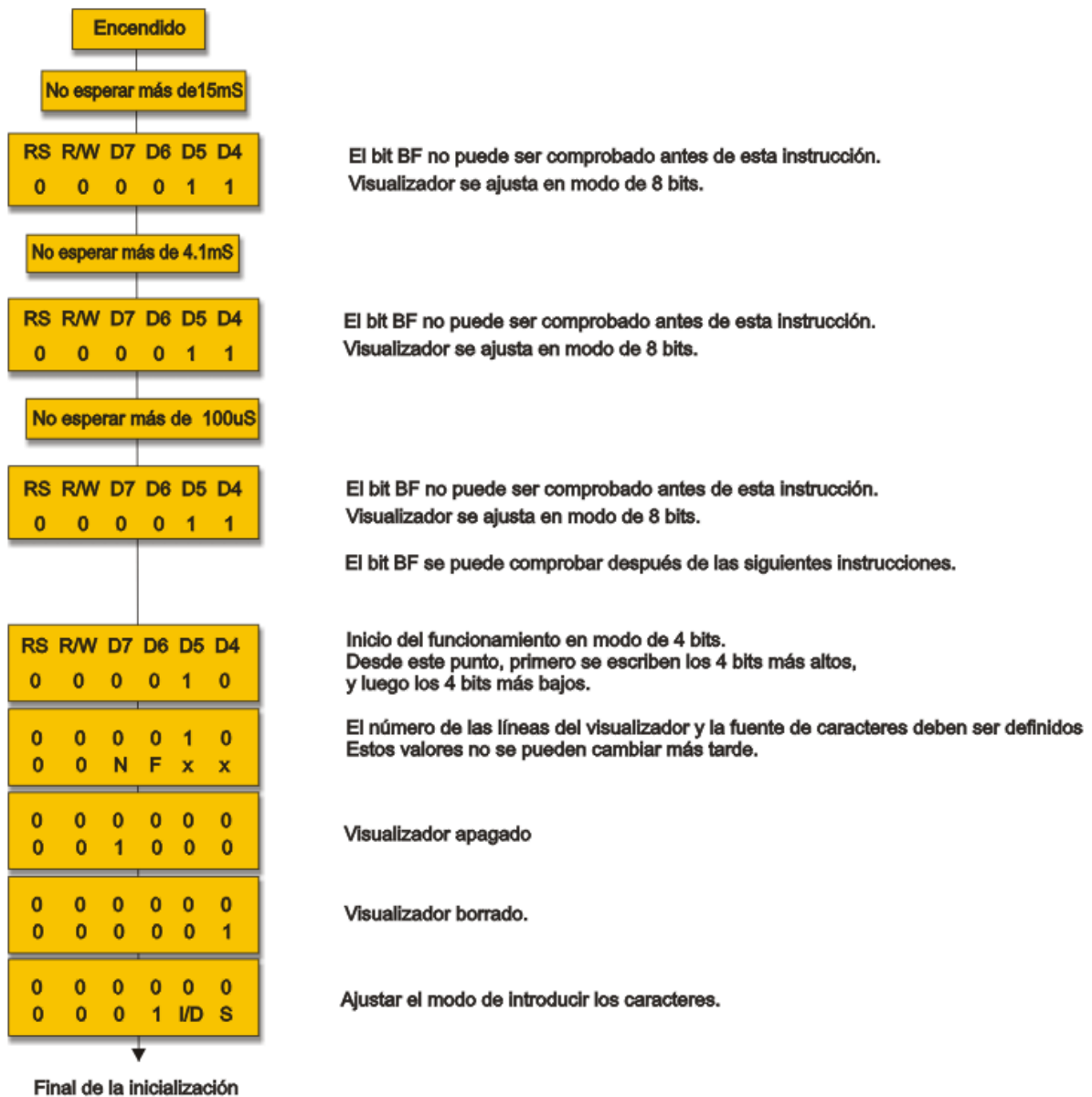
Hay dos algoritmos de inicialización. Cuál se utilizará depende de si la conexión al microcontrolador se realiza por el bus de datos de 4 o 8 bits. En ambos casos, después de inicialización sólo queda especificar los comandos básicos y, por supuesto, visualizar los mensajes.

Refiérase a la Figura que sigue para el procedimiento de inicialización por el bus de datos de 8 bits:



¡Esto no es un error! En este algoritmo, el mismo valor se transmite tres veces en fila.

El procedimiento de inicialización por el bus de datos de 4 bits:



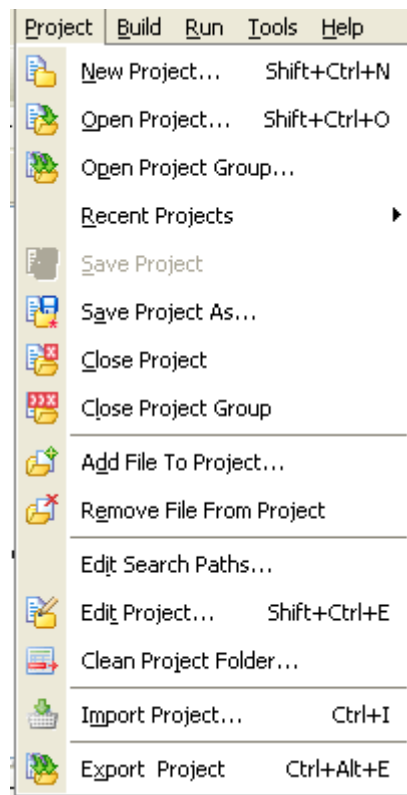
Vamos a hacerlo en mikroC...

/ En mikroC for PIC, basta con escribir sólo una función para realizar todo el proceso de la inicialización del LCD. Antes de llamar esta función es necesario declarar los bits LCD_D4-LCD_D7, LCD_RS y LCD_EN. */*

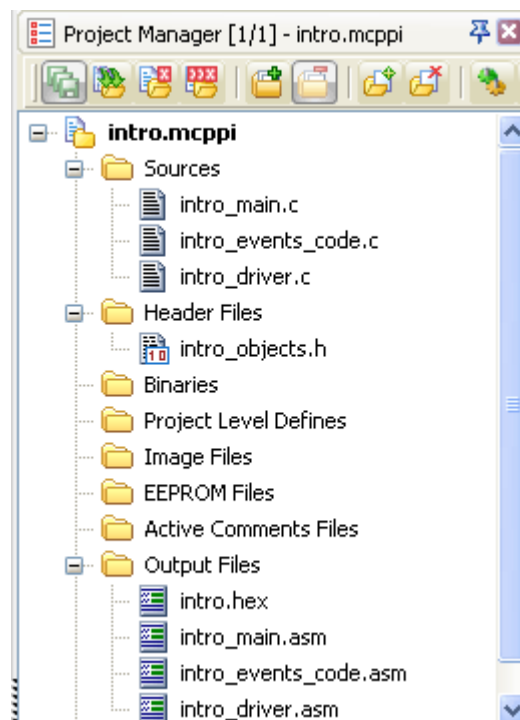
```
...
Lcd_Init(); // Inicializar el LCD
...
```

EJEMPLOS PRÁCTICOS

El proceso de crear un proyecto nuevo es muy simple. Seleccione la opción **New Project** del menú **Project** como se muestra en la Figura de la derecha.



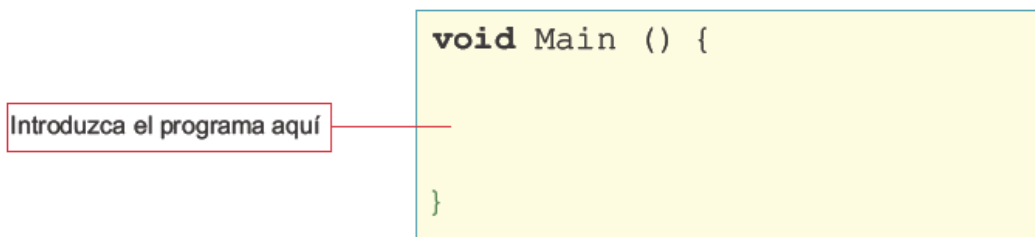
Aparecerá una ventana que le guiará a través del proceso de creación de un proyecto nuevo. La ventana de entrada de este programa contiene una lista de acciones a realizar para crear un proyecto nuevo. Pulse el botón **Next**.



El proceso de creación de un proyecto nuevo consiste en cinco pasos:

1. Seleccione el tipo de microcontrolador a programar. En este caso se trata del PIC16F887.
2. Seleccione la frecuencia de reloj del microcontrolador. En este caso el valor seleccionado es 8 MHz.
3. Seleccione el nombre y la ruta del proyecto. En este caso, el nombre del proyecto es First_Project. Está guardado en la carpeta C:\My projects. Al nombre del proyecto se le asigna automáticamente la extensión .mcppi. Se creará en el proyecto el archivo fuente con el mismo nombre (First_Project .c.h).
4. Si el nuevo proyecto consiste de varios archivos fuente, se necesita especificarlos y pulse sobre el botón Add para incluirlos. En este ejemplo no hay archivos fuente adicionales.
5. Por último, se necesita confirmar todas las opciones seleccionadas. Pulse sobre Finish.

Después de crear el proyecto, aparecerá una ventana blanca en la que debe escribir el programa. Vea la siguiente figura:



Una vez creado el programa, es necesario compilarlo en un código .hex. Seleccione una de las opciones para compilar del menú **Project**:

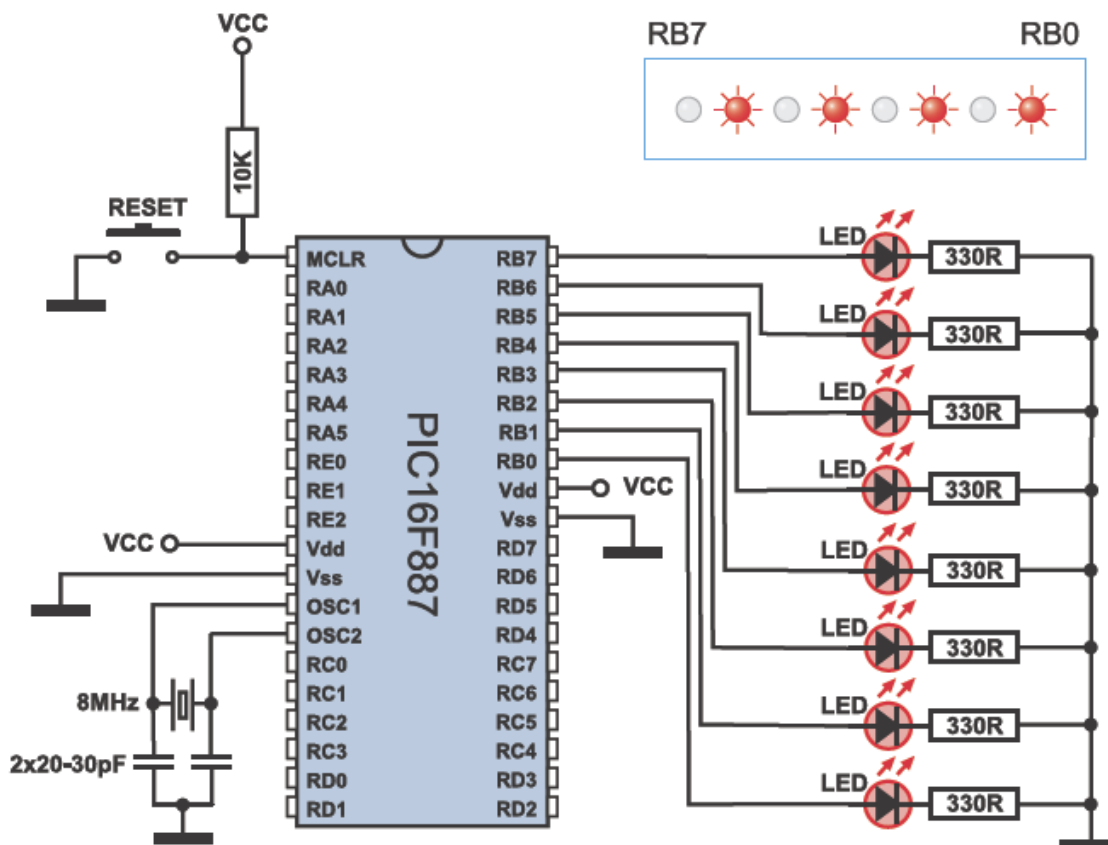
- Para crear un archivo .hex, seleccione la opción Build (Ctrl+F9) del menú Project o pulse sobre el icono Build de la barra de herramientas Project.
- Por medio de la opción Build All Projects (Shift+F9) se compilan todos los archivos del proyecto, librerías (si el código fuente contiene alguna de ellas) y los archivos def para el microcontrolador utilizado.
- La opción Build + Program (Ctrl+F11) es importante ya que permite al compilador mikroC PRO for PIC cargar automáticamente el programa en el microcontrolador después de la compilación. El proceso de la programación se realiza por medio del programador PICFlash.

Todos los errores encontrados durante la compilación aparecerán en la ventana Message. Si no hay errores en el programa, el compilador mikroC PRO for PIC generará los correspondientes archivos de salida.

EJEMPLO 1

Escribir cabecera, configurar pines de E/S, utilizar la función Delay y el operador Switch

El único propósito de este programa es de encender varios diodos LED en el puerto B. Utilice este ejemplo para examinar cómo es un programa real. La siguiente figura muestra el esquema de conexión, mientras que el programa se encuentra en la siguiente página.



Al encender la fuente de alimentación, cada segundo, el diodo LED en el puerto B emite luz, lo que indica que el microcontrolador está conectado correctamente y que funciona normalmente.

En este ejemplo se muestra cómo escribir una cabecera correctamente. Lo mismo se aplica a todos los programas descritos en este libro. Para no repetir, en los siguientes ejemplos no vamos a escribir la cabecera. Se considera estar en el principio de cada programa, marcada como "Cabecera".

Ejemplo 1

Cabecera

```

/*
 * Nombre de programa:
   Ejemplo 1
 * Derecho de autor:
   (c) MikroElektronika, 2005-2010
 * Descripción:
   Esto es un simple programa utilizado para demostrar el funcionamiento del
   microcontrolador. Cada segundo varios LED en el puerto PORTB estarán encendidos.

 * Configuración:
   Microcontrolador: PIC16F887
   Dispositivo:      EasyPIC6
   Oscilador:       HS, 08.0000 MHz
   SW:              mikroC PRO v8.0
 * Notas: -
 */

```

La cabecera se coloca en el principio del programa y proporciona informaciones básicas en forma de comentarios (nombre de programa, fecha de lanzamiento, etc.) No se haga ilusiones que en varios meses sabrá qué es lo que ha hecho el programa y por qué lo ha guardado. De eso se encargan los comentarios.

Ejecución de programa

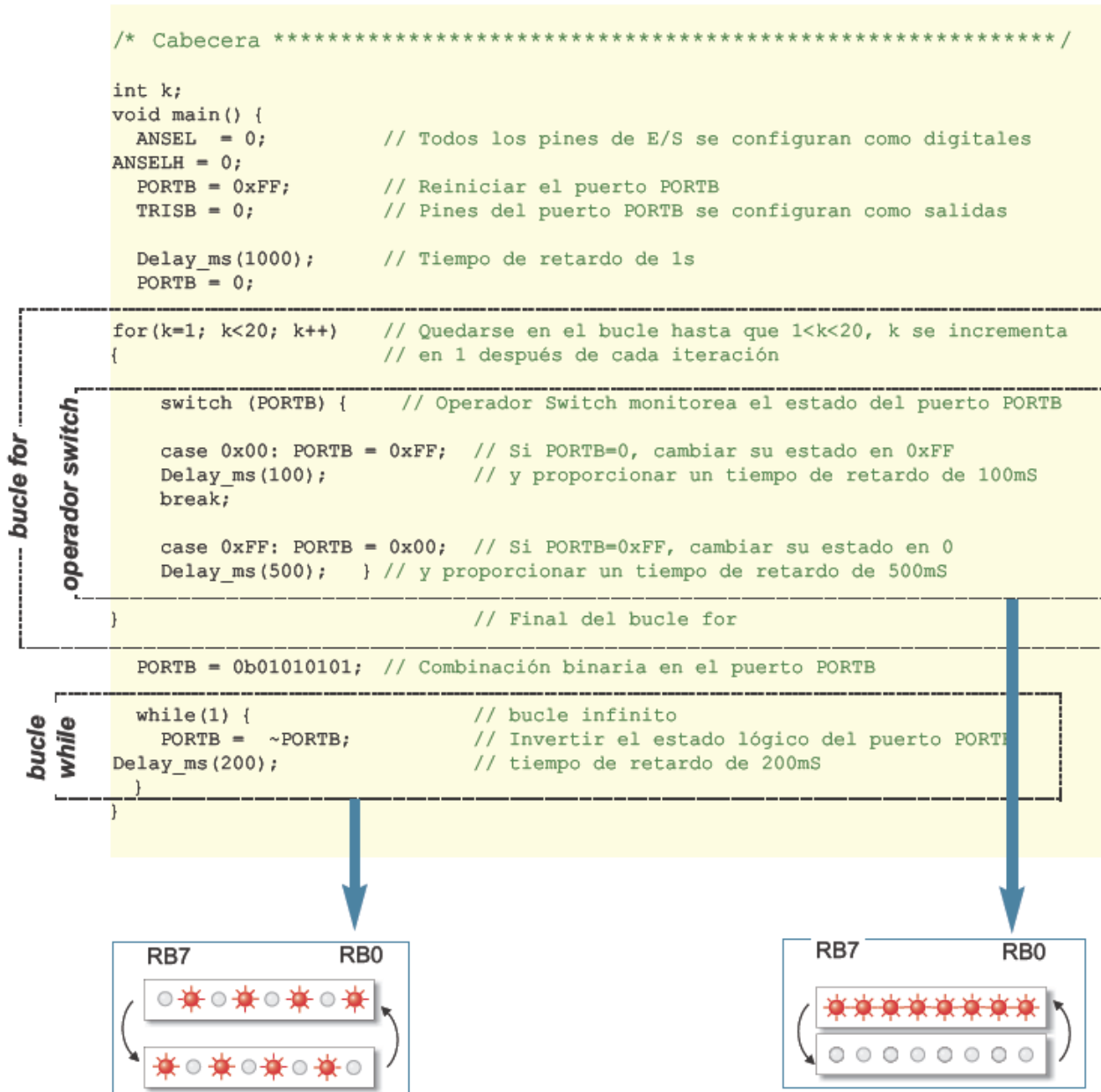
```

void main() {
  ANSEL = 0;           // Todos los pines de E/S se configuran como digitales
  ANSELH = 0;
  PORTB = 0b01010101; // Combinación binaria en el puerto PORTB
  TRISB = 0;          // Pines del puerto PORTB se configuran como salidas
}

```

Para hacer este ejemplo más interesante, vamos a habilitar que los LEDs conectados al puerto PORTB parpadeen. Hay varios modos de hacerlo:

1. Tan pronto como se encienda el microcontrolador, todos los LEDs emitirán la luz por un segundo. La función Delay se encarga de eso en el programa. Sólo se necesita ajustar la duración del tiempo de retardo en milisegundos.
2. Después de un segundo, el programa entra en el bucle for, y se queda allí hasta que la variable k sea menor que 20. La variable se incrementa en 1 después de cada iteración. Dentro del bucle for, el operador switch monitorea el estado lógico en el puerto PORTB. Si PORTB=0xFF, su estado se invierte en 0x00 y viceversa. Cualquier cambio de estos estados lógicos hace todos los LEDs parpadear. El ciclo de trabajo es 5:1 (500mS:100mS).
3. Al salir del bucle for, el estado lógico del puerto POTRB cambia (0xb 01010101) y el programa entra en el bucle infinito while y se queda allí hasta que 1=1. El estado lógico del puerto PORTB se invierte cada 200mS.



EJEMPLO 2

Utilizar instrucciones en ensamblador y el oscilador interno LFINTOSC...

En realidad, esto es una continuación del ejemplo anterior, pero se ocupa de un problema un poco más complicado... El propósito era hacer los LEDs en el puerto PORTB parpadear lentamente. Se puede realizar al introducir un valor suficiente grande para el parámetro del tiempo de retardo en la función Delay. No obstante, hay otra manera más eficiente para ejecutar el programa lentamente. Acuértese de que este microcontrolador tiene un oscilador incorporado LFINTOSC que funciona a una frecuencia de 31kHz. Ahora llegó la hora de “darle una oportunidad”.

El programa se inicia con el bucle do-while y se queda allí por 20 ciclos. Después de cada iteración, llega el tiempo de retardo de 100ms, indicado por un parpadeo relativamente rápido de los LEDs en el puerto PORTB. Cuando el programa salga de este bucle, el microcontrolador se inicia al utilizar el oscilador LFINTOSC como una fuente de señal de reloj. Los LEDs parpadean más lentamente aunque el programa ejecuta el mismo bucle do-while con un tiempo de retardo 10 veces más corto.

Con el propósito de hacer evidentes algunas situaciones potencialmente peligrosas, se activan los bits de control por medio de las instrucciones en ensamblador. Dicho de manera sencilla, al entrar o salir una instrucción en ensamblador en el programa, el compilador no almacena los datos en un banco actualmente activo de la RAM, lo que significa que en esta sección de programa, la selección de bancos depende del registro SFR utilizado. Al volver a la sección de programa escrito en C, los bits de control RP0 y RP1 deben recuperar el estado que tenían antes de 'la aventura en ensamblador'. En este programa, el problema se resuelve al utilizar la variable auxiliar saveBank, lo que guarda el estado de estos dos bits.

```

/* Cabecera *****/

int k = 0;           // Variable k es de tipo int
char saveBank;     // Variable saveBank es de tipo char

void main() {
  ANSEL = 0;       // Todos los pines de E/S se configuran como digitales
  ANSELH = 0;
  PORTB = 0;      // Todos los pines del puerto PORTB se ponen a 0
  TRISB = 0;     // Pines del puerto PORTB se configuran como salidas

  do {
    PORTB = ~PORTB; // Invertir el estado lógico del puerto PORTB
    Delay_ms(100); // Tiempo de retardo de 100mS
    k++;           // Incrementar k en 1
  }
  while(k<20);    // Quedarse en bucle hasta que k<20

  k=0;           // Reiniciar variable k
  saveBank = STATUS & 0b01100000; // Guardar el estado de los bits RP0 y RP1

  // (bits 5 y 6 del registro STATUS)
  asm {
    bsf STATUS,RP0 // Seleccionar el banco de memoria que contiene el
    bcf STATUS,RP1 // registro OSCCON
    bcf OSCCON,6   // Seleccionar el oscilador interno LFINTOSC
    bcf OSCCON,5   // de frecuencia de 31KHz
    bcf OSCCON,4
    bsf OSCCON,0   // Microcontrolador utiliza oscilador interno
  } // Final de la secuencia en ensamblador

  STATUS &= 0b10011111; // Bits RP0 y RP1 recuperan el estado original
  STATUS |= saveBank;

  do {
    PORTB = ~PORTB; // Invertir el estado lógico del puerto PORTB
    Delay_ms(10);   // Tiempo de retardo de 10 mS
    k++;           // Incrementar k en 1
  }
  while(k<20);    // Quedarse en el bucle hasta que k<20
}

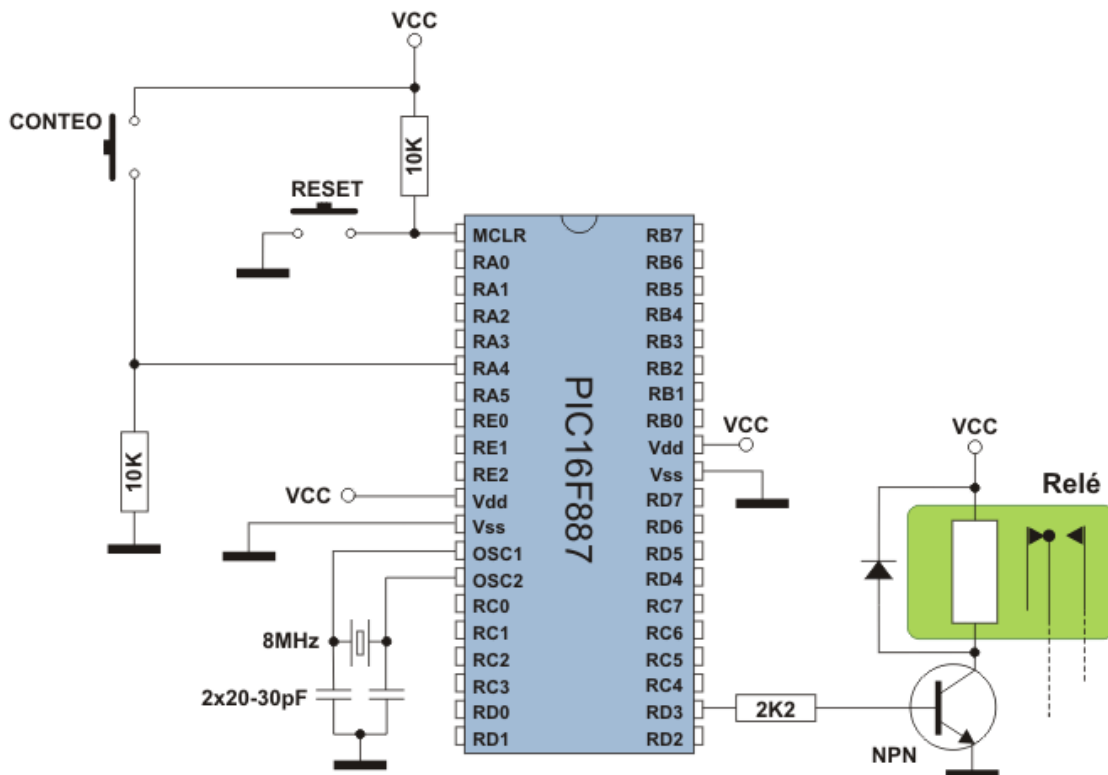
```

EJEMPLO 3

Timer0 como un contador, declarar variables nuevas, constantes de enumeración, utilizar relés...

En cuanto a los ejemplos anteriores, el microcontrolador ha ejecutado el programa sin haber sido afectado de ninguna forma por su entorno. En la práctica, hay pocos dispositivos que funcionen de esta manera (por ejemplo, un simple controlador de luz de neón). Los pines de entrada se utilizan también en este ejemplo. En la siguiente figura se muestra un esquema, mientras que el programa está en la siguiente página. Todo sigue siendo muy simple. El temporizador Timer0 se utiliza como un contador. La entrada del contador está conectada a un botón de presión, así que cada vez que se presiona el botón, el temporizador Timer0 cuenta un pulso.

Cuando el número de pulsos coincide con el número almacenado en el registro TEST, un uno lógico (5V) aparecerá en el pin PORTD.3. Este voltaje activa un relé electromecánico, y por eso este bit se le denomina 'RELÉ' en el programa.



```
/*Cabecera*****
```

```
void main() {
  char TEST = 5;      // Constante TEST = 5
  enum salidas {RELÉ = 3}; // Constante RELAY = 3

  ANSEL = 0;        // Todos los pines de E/S se configuran como digitales
  ANSELH = 0;
  PORTA = 0;        // Reiniciar el puerto PORTA
  TRISA = 0xFF;     // Todos los pines del puerto PORTA se configuran como entradas
  PORTD = 0;        // Reiniciar el puerto PORTD
  TRISD = 0b11110111; // Pin RD3 se configura como salida, mientras que los demás
                      // se configuran como entradas
}
```

```

OPTION_REG.F5 = 1;    // Contador TMR0 recibe los pulsos por el pin RA4
OPTION_REG.F3 = 1;    // Valor del pre-escalador 1:1

TMR0 = 0;            // Reiniciar el temporizador/contador TMR0

do {
  if (TMR0 == TEST)  // ¿Coincide el número en el temporizador con la
                    // constante TEST?
    (PORTD.RELAY = 1); // Números coinciden. Poner el bit RD3 a uno (salida RELÉ)
}
while (1);          // Quedarse en el bucle infinito
}

```

Sólo una constante de enumeración RELÉ se utiliza en este ejemplo. Se le asigna un valor mediante la declaración.

```
enum salidas {RELÉ = 3}; // Constante RELÉ = 3
```

Si varios pines del puerto PORTD están conectados a los relés, la expresión anterior se puede escribir de la siguiente manera también:

```
enum salidas {RELÉ=3, CALENTADOR, MOTOR=6, SURTIDOR};
```

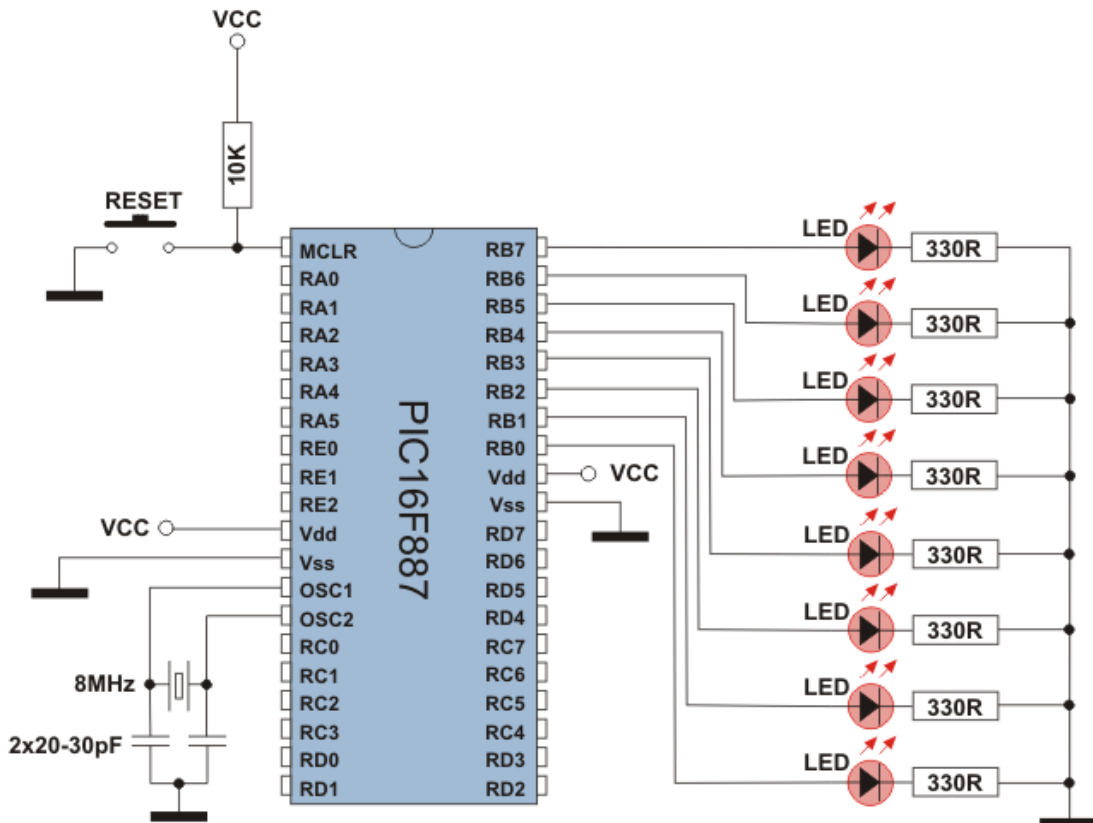
A todas las constantes, precedidas por las constantes con valores asignados (RELÉ=3 y MOTOR=6), se les asignan automáticamente los valores de las constantes precedentes, incrementados en 1. En este ejemplo, a las constantes CALENTADOR y SURTIDOR se les asignan los valores 4 y 7, es decir (CALENTADOR=4 y SURTIDOR=7), respectivamente.

EJEMPLO 4

Utilizar los temporizadores Timer0, Timer1 y Timer2. Utilizar interrupciones, declarar nuevas funciones...

Al analizar los ejemplos anteriores, es probable que se haya fijado en la desventaja de proporcionar tiempo de retardo por medio de la función Delay. En estos casos, el microcontrolador se queda 'estático' y no hace nada. Simplemente espera que transcurra una cierta cantidad de tiempo. Tal pérdida de tiempo es un lujo inaceptable, por lo que se deberá aplicar otro método.

¿Se acuerda usted del capítulo de los temporizadores? ¿Se acuerda de lo de interrupciones? Este ejemplo los conecta de una manera práctica. El esquema se queda inalterada, y el desafío sigue siendo presente. Es necesario proporcionar un tiempo de retardo suficiente largo para darse cuenta de los cambios en el puerto. Para este propósito se utiliza el temporizador Timer0 con el pre-escalador asignado. Siempre que se genere una interrupción con cada desbordamiento en el registro del temporizador, la variable cnt se aumenta automáticamente en 1 al ejecutarse cada rutina de interrupción. Cuando la variable llega al valor 400, el puerto PORTB se incrementa en 1. Todo el procedimiento se lleva a cabo "entre bastidores", lo que habilita al microcontrolador hacer otra tarea.



/*Cabecera*****

unsigned cnt; // Definir la variable cnt

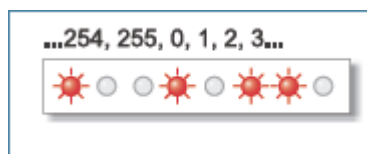
```
void interrupt() {
    cnt++; // Con una interrupción la cnt se incrementa en 1
    TMR0 = 96; // El valor inicial se devuelve en el temporizador TMR0
    INTCON = 0x20; // Bit T0IE se pone a 1, el bit T0IF se pone a 0
}
```

```
void main(){
    OPTION_REG = 0x84; // Pre-escalador se le asigna al temporizador TMR0
    ANSEL = 0; // Todos los pines de E/S se configuran como digitales
    ANSELH = 0;
    TRISB = 0; // Todos los pines de puerto PORTB se configuran
```

```
    // como salidas
    PORTB = 0x0; // Reiniciar el puerto PORTB
    TMR0 = 96; // Temporizador T0 cuenta de 96 a 255
    INTCON = 0xA0; // Habilitada interrupción TMR0
    cnt = 0; // A la variable cnt se le asigna un 0
```

```
do { // Bucle infinito
    if (cnt == 400) { // Incrementar el puerto PORTB después 400 interrupciones
        PORTB = PORTB++; // Incrementar número en el puerto PORTB en 1
        cnt = 0; // Reiniciar la variable cnt
    }
} while(1);
```

}



Siempre que se produzca un desbordamiento en el registro del temporizador TRM0, ocurre una interrupción.

```

/*Cabecera*****/

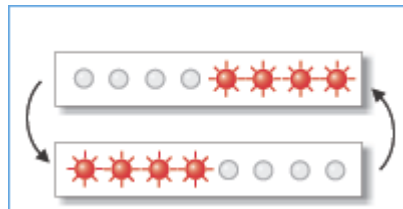
unsigned short cnt; // Definir la variable cnt

void interrupt() {
    cnt++; // Con una interrupción la cnt se incrementa en 1
    PIR1.TMR1IF = 0; // Reiniciar el bit TMR1IF
    TMR1H = 0x80; // El valor inicial se devuelve en los registros
    TMR1L = 0x00; // del temporizador TMR1H y TMR1L
}

void main() {
    ANSEL = 0; // Todos los pines de E/S se configuran como digitales
    ANSELH = 0;
    PORTB = 0xF0; // Valor inicial de los bits del puerto PORTB
    TRISB = 0; // Pines del puerto PORTB se configuran como salidas
    T1CON = 1; // Configurar el temporizador TMR1
    PIR1.TMR1IF = 0; // Reiniciar el bit TMR1IF
    TMR1H = 0x80; // Ajustar el valor inicial del temporizador TMR1
    TMR1L = 0x00;
    PIE1.TMR1IE = 1; // Habilitar la interrupción al producirse un desbordamiento
    cnt = 0; // Reiniciar la variable cnt
    INTCON = 0xC0; // Interrupción habilitada (bits GIE y PEIE)

    do { // Bucle infinito
        if (cnt == 76) { // Cambiar el estado del puerto PORTB después de 76 interrupciones
            PORTB = ~PORTB; // Número en el puerto PORTB está invertido
            cnt = 0; // Reiniciar la variable cnt
        }
    } while (1);
}

```



En este caso, una interrupción se habilita después de que se produzca un desbordamiento en el registro del temporizador TMR1 (TMR1H, TMR1L). Además, la combinación de los bits que varía en el puerto POTRB difiere de la en el ejemplo anterior.

```

/*Cabecera*****/

unsigned short cnt; // Definir la variable cnt

void Reemplazar() {
    PORTB = ~PORTB; // Definir nueva función 'Reemplazar'
} // Función invierte el estado del puerto

void interrupt() {
    if (PIR1.TMR2IF) { // Si el bit TMR2IF = 1,
        cnt++; // Incrementar variable la cnt en 1
        PIR1.TMR2IF = 0; // Reiniciar el bit y
        TMR2 = 0; // Reiniciar el registro TMR2
    }
}

```

```

}

// main
void main() {
    cnt = 0;           // Reiniciar la variable cnt
    ANSEL = 0;        // Todos los pines de E/S se configuran como digitales
    ANSELH = 0;
    PORTB = 0b10101010; // Estado lógico en los pines del puerto PORTB
    TRISB = 0;        // Todos los pines del puerto PORTB se configuran como salidas
    T2CON = 0xFF;     // Configurar el temporizador T2
    TMR2 = 0;         // Valor inicial del registro del temporizador TMR2
    PIE1.TMR2IE = 1; // Interrupción habilitada
    INTCON = 0xC0;    // Bits GIE y PEIE se ponen a 1

    while (1) {       // Bucle infinito
        if (cnt > 30) { // Cambiar el estado del puerto PORTB después de
                        // más de 30 interrupciones
            Reemplazar(); // Función Reemplazar invierte el estado del puerto PORTB
            cnt = 0;       // Reiniciar la variable cnt
        }
    }
}

```

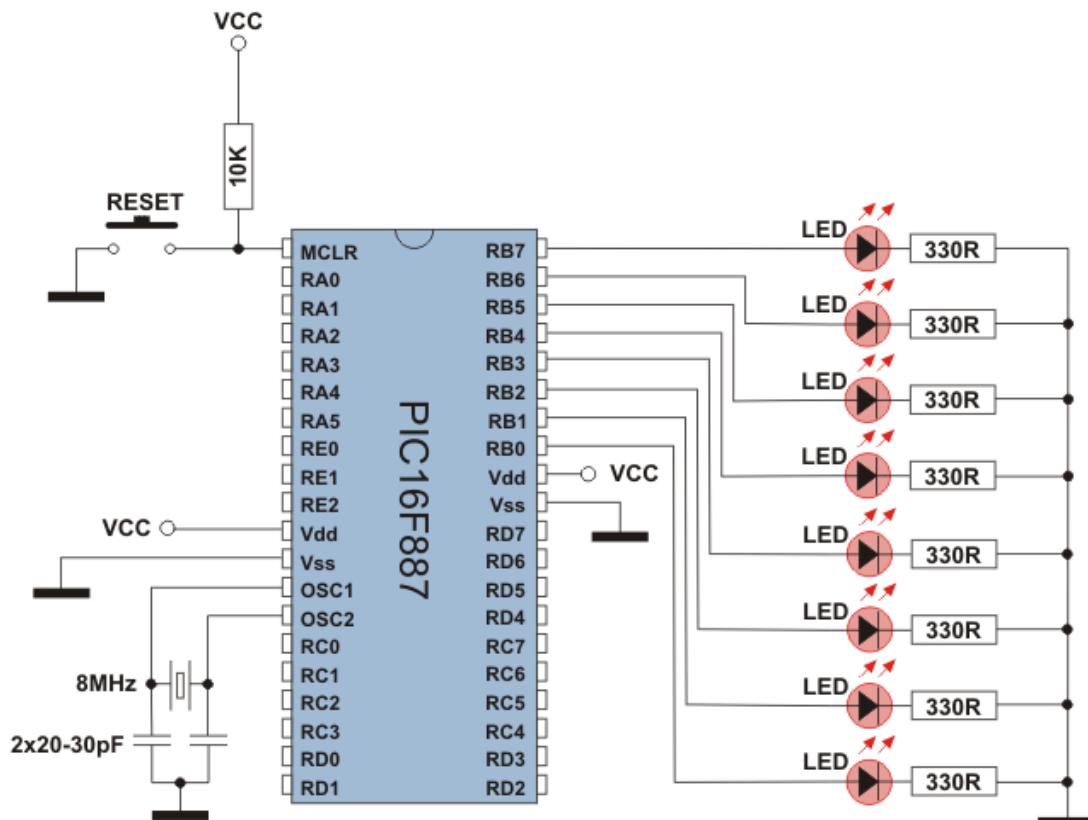


En este ejemplo, una interrupción ocurre después de que se produce un desbordamiento en el registro del temporizador TMR2. Para invertir el estado lógico de los pines del puerto se utiliza la función Reemplazar, que normalmente no pertenece al lenguaje C estándar.

EJEMPLO 5

Utilizar el temporizador perro - guardián

Este ejemplo muestra cómo NO se debe utilizar el temporizador perro-guardián. Un comando utilizado para reiniciar este temporizador se omite a propósito en el bucle del programa principal, lo que habilita al temporizador perro guardián “ganar la batalla del tiempo” y reiniciar al microcontrolador. Por consiguiente, el microcontrolador se va a reiniciar sin parar, lo que indicará el parpadeo de los LEDs del puerto PORTB.



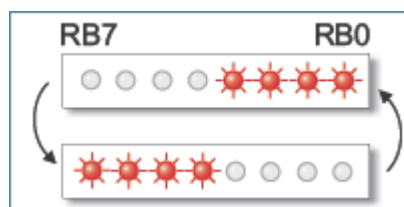
/*Cabecera*****

```

void main() {
    OPTION_REG = 0x0E; // Pre-escalador se le asigna al temporizador WDT (1:64)
    asm CLRWDT;      // Comando en ensamblador para reiniciar el temporizador WDT
    PORTB = 0x0F;    // Valor inicial del registro PORTB
    TRISB = 0;      // Todos los pines del puerto PORTB se configuran como salidas
    Delay_ms(300);  // Tiempo de retardo de 30mS
    PORTB = 0xF0;   // Valor del puerto PORTB diferente del inicial

    while (1);      // Bucle infinito. El programa se queda aquí hasta que el
                    // temporizador WDT reinicie al microcontrolador
}
    
```

Para que este ejemplo funcione apropiadamente, es necesario habilitar al temporizador perro-guardián al seleccionar la opción Tools/mE Programmer/Watchdog Timer - Enabled.



EJEMPLO 6

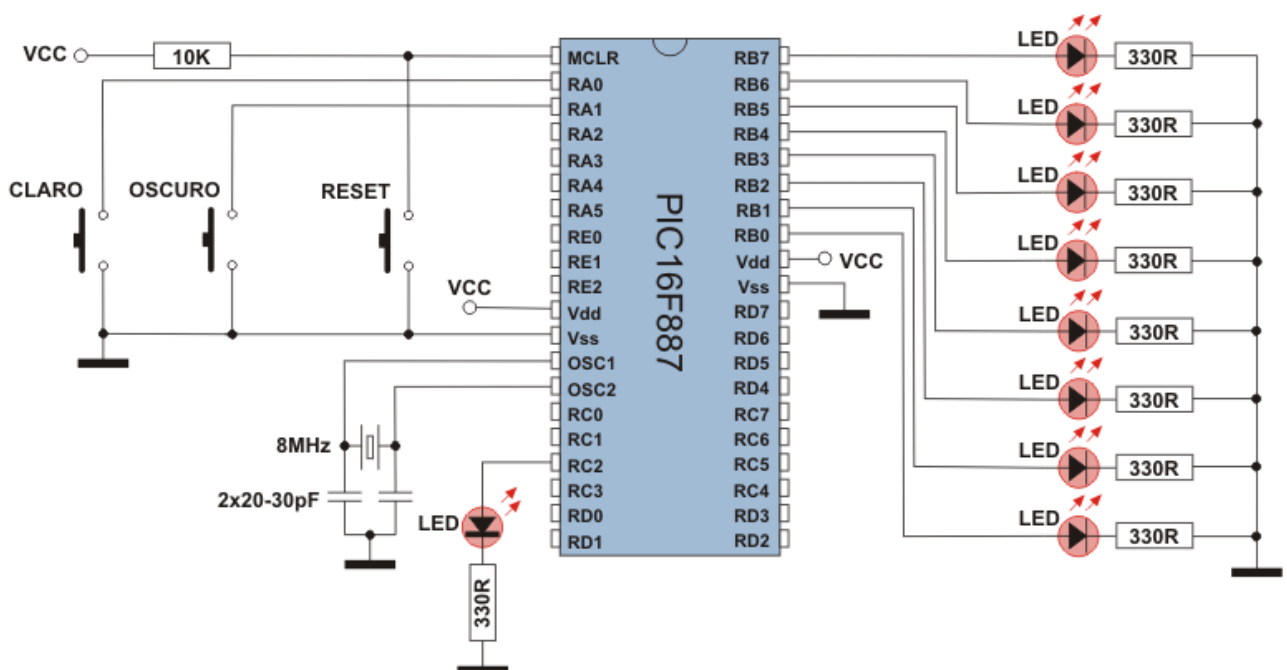
Módulo CCP1 como generador de señal PWM

Este ejemplo muestra el uso del módulo CCP1 en modo PWM. Para hacer las cosas más interesantes, la duración de los pulsos en la salida P1A (PORTC,2) se puede

cambiar por medio de los botones de presión simbólicamente denominados 'OSCURO' y 'CLARO'. La duración ajustada se visualiza como una combinación binaria en el puerto PORTB. El funcionamiento de este módulo está bajo el control de las funciones pertenecientes a la librería especializada PWM. Aquí se utilizan las tres de ellas:

1. **PWM1_init** tiene el prototipo: **void Pwm1_Init(long freq);**
El parámetro freq ajusta la frecuencia de la señal PWM expresada en hercios. En este ejemplo equivale a 5kHz.
2. **PWM1_Start** tiene el prototipo: **void Pwm1_Start(void);**
3. **PWM1_Set_Duty** tiene el prototipo: **void Pwm1_Set_Duty(unsigned short duty_ratio);**
El parámetro duty_ratio ajusta la duración de pulsos en una secuencia de pulsos.

La librería PWM también contiene la función PWM_Stop utilizada para deshabilitar este modo. Su prototipo es: **void Pwm1_Stop(void);**



/*Cabecera*****

// Definir las variables ciclo_de_trabajo_actual,
// ciclo_de_trabajo_anterior

unsigned short ciclo_de_trabajo_actual;
unsigned short ciclo_de_trabajo_anterior;

```
void initMain() {
    ANSEL = 0;    // Todos los pines de E/S se configuran como digitales
    ANSELH = 0;
    PORTA = 255; // Estado inicial del puerto PORTA
    TRISA = 255; // Todos los pines del puerto PORTA se configuran como entradas
    PORTB = 0;   // Estado inicial del puerto PORTB
    TRISB = 0;  // Todos los pines del puerto PORTB se configuran como salidas
    PORTC = 0;  // Estado inicial del puerto PORTC
    TRISC = 0;  // Todos los pines del puerto PORTC se configuran

    // como salidas
    PWM1_Init(5000); // Inicialización del módulo PWM (5KHz)
}
```



```

void main() {
  initMain();
  ciclo_de_trabajo_actual = 16; // Valor inicial de la variable ciclo_de_trabajo_actual
  ciclo_de_trabajo_anterior = 0; // Reiniciar la variable ciclo_de_trabajo_anterior
  PWM1_Start(); // Iniciar el módulo PWM1

  while (1) { // Bucle infinito
    if (Button(&PORTA, 0,1,1)) // Si se presiona el botón conectado al RA0
      ciclo_de_trabajo_actual++; // incrementar el valor de la variable current_duty
    if (Button(&PORTA, 1,1,1)) // Si se presiona el botón conectado al RA1
      ciclo_de_trabajo_actual--; // decrementar el valor de la variable current_duty

    if (old_duty != ciclo_de_trabajo_actual) { // Si ciclo_de_trabajo_actual y
      // ciclo_de_trabajo_anterior no son iguales
      PWM1_Set_Duty(ciclo_de_trabajo_actual); // ajustar un nuevo valor a PWM,
      ciclo_de_trabajo_anterior = ciclo_de_trabajo_actual; // Guardar el nuevo valor
      PORTB = ciclo_de_trabajo_anterior; // y visualizarlo en el puerto PORTB
    }
    Delay_ms(200); // Tiempo de retardo de 200mS
  }
}

```

Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana Library Manager antes de compilar el programa:

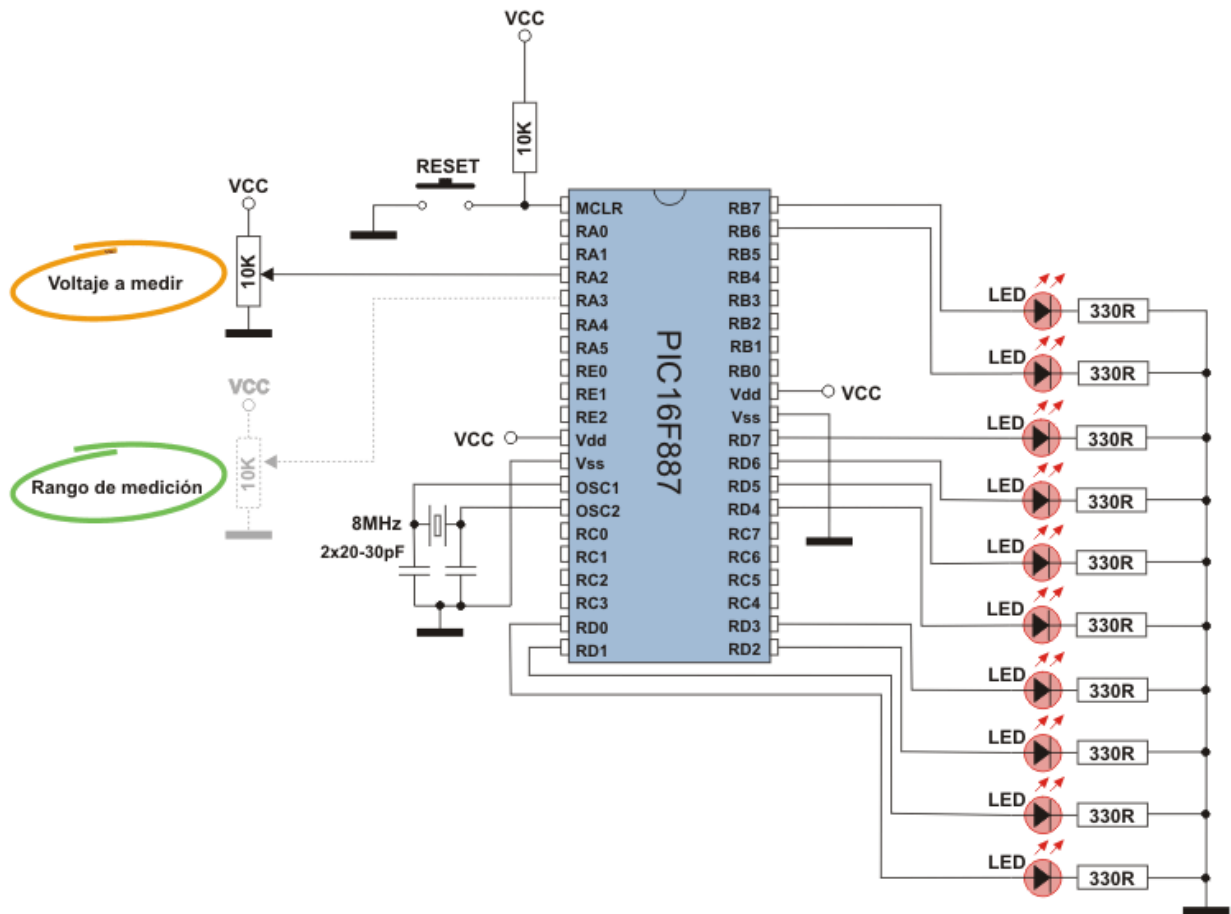
- PWM
- Button

EJEMPLO 7

Utilizar el convertidor A/D

El convertidor A/D del microcontrolador PIC16F887 se utiliza en este ejemplo. ¿Hace falta decir que todo es pan comido? Una señal analógica variable se aplica al pin AN2, mientras que el resultado de la conversión de 10 bits se muestra en los puertos POTRB y PORTD (8 bits menos significativos en el puerto PORTD y 2 bits más significativos en el puerto PORTB). La Tierra (GND) se utiliza como voltaje de referencia bajo Vref-, mientras que el voltaje de referencia alto se aplica al pin AN3. Esto habilita que la escala de medición se estire y encoja.

En otras palabras, el convertidor A/D siempre genera un resultado binario de 10 bits, lo que significa que reconoce 1024 niveles de voltaje en total ($2^{10}=1024$). La diferencia entre dos niveles de voltaje no es siempre la misma. Cuánto menor sea la diferencia entre Vref+ y Vref-, tanto menor será la diferencia entre dos de 1024 niveles. Como hemos visto, el convertidor A/D es capaz de detectar pequeños cambios de voltaje.



*/*Cabecera******

unsigned int temp_res;

void main() {

 ANSEL = 0x0C; *// Pines AN2 y AN3 se configuran como analógicos*
 TRISA = 0xFF; *// Todos los pines del puerto PORTA se configuran*

// como entradas

 ANSELH = 0; *// Los demás pines se configuran como digitales*
 TRISB = 0x3F; *// Pines del puerto PORTB, RB7 y RB6 se configuran*

// como salidas

 TRISD = 0; *// Todos los pines del PORTD se configuran como salidas*
 ADCON1.F4 = 1; *// Voltaje de referencia es llevado al pin RA3.*

do {

 temp_res = ADC_Read(2); *// Resultado de la conversión A/D es copiado a temp_res*
 PORTD = temp_res; *// 8 bits menos significativos se mueven al puerto PORTD*
 PORTB = temp_res >> 2; *// 2 bits más significativos se mueven a los bits RB6 y RB7*
} **while**(1); *// Bucle infinito*

}

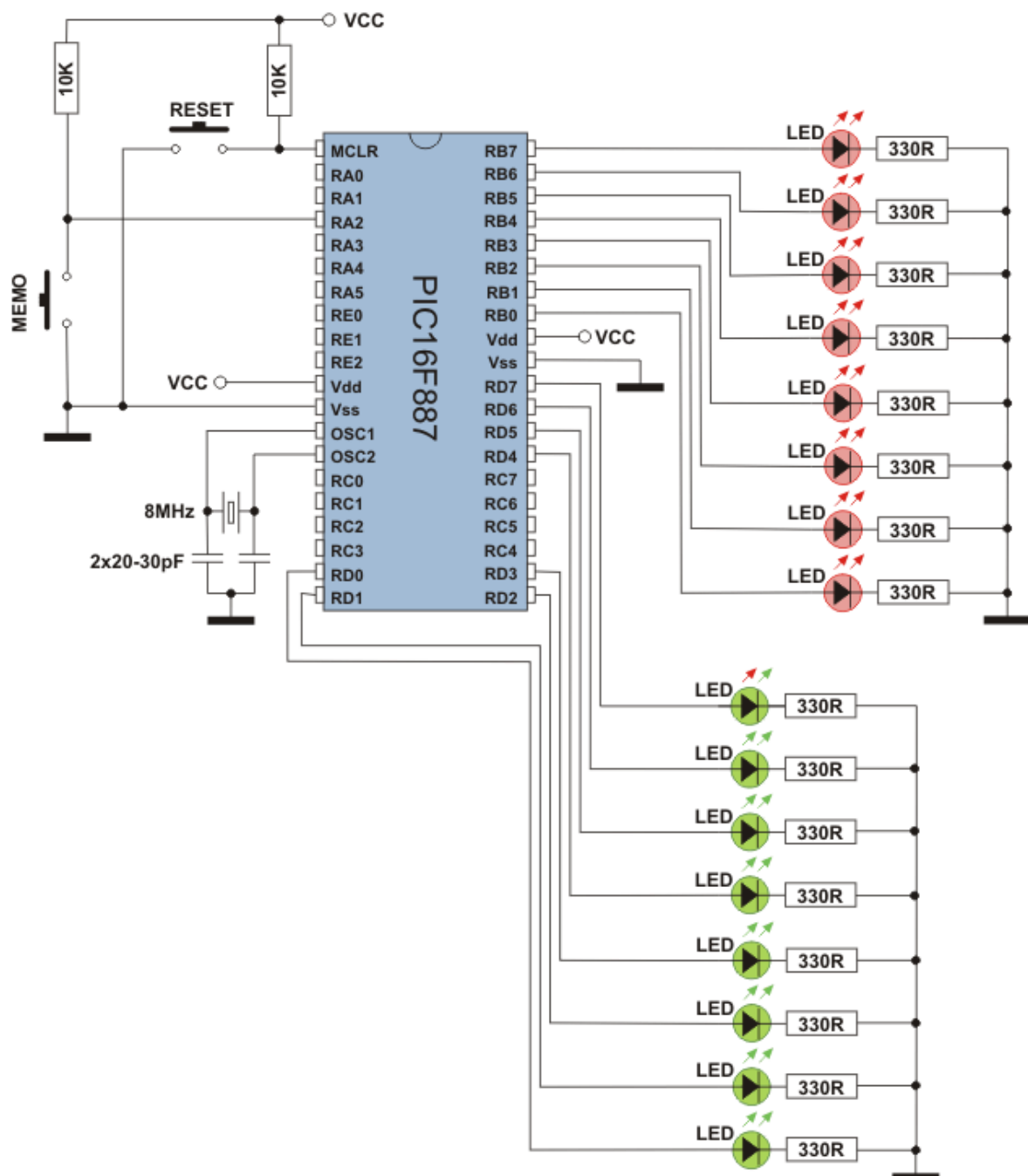
Para que este ejemplo funcione apropiadamente, es necesario marcar la librería ADC en la ventana Library Manager antes de compilar el programa:

- ADC

EJEMPLO 8

Utilizar memoria EEPROM

Este ejemplo muestra cómo escribir y leer la memoria EEPROM incorporada. El programa funciona de la siguiente manera. El bucle principal lee constantemente el contenido de localización de la memoria EEPROM en la dirección 5 (decimal). Luego el programa entra en el bucle infinito en el que el puerto PORTB se incrementa y se comprueba el estado de entradas del puerto PORTA.2. En el momento de presionar el botón denominado MEMO, un número almacenado en el puerto PORTB será guardado en la memoria EEPROM, directamente leído y visualizado en el puerto PORTD en forma binaria.



```

/*Cabecera*****
void main() {{
  ANSEL = 0;           // Todos los pines de E/S se configuran como digitales
  ANSELH = 0;
  PORTB = 0;          // Valor inicial del puerto PORTB
  TRISB = 0;          // Todos los pines del puerto PORTB se configuran

  // como salidas
  PORTD = 0;          // Valor inicial del puerto PORTB
  TRISD = 0;          // Todos los pines del puerto PORTD se configuran

  // como salidas
  TRISA = 0xFF;       // Todos los pines del puerto PORTA se configuran

  // como entradas
  PORTD = EEPROM_Read(5); // Leer la memoria EEPROM en la dirección 5

  do {
    PORTB = PORTB++; // Incrementar el puerto PORTB en 1
    Delay_ms(100);   // Tiempo de retardo de 100mS

    if (PORTA.F2)
      EEPROM_Write(5,PORTB); // Si se pulsa el botón MEMO, guardar el puerto PORTB

    PORTD = EEPROM_Read(5); // Leer el dato escrito

    do {
      while (PORTA.F2); // Quedarse en este bucle hasta que el botón esté pulsado
    }
  }
  while(1);           // Bucle infinito
}

```

Para comprobar el funcionamiento de este circuito, basta con pulsar el botón MEMO y apagar el dispositivo. Después de reiniciar el dispositivo, el programa visualizará el valor guardado en el puerto PORTD. Acuérdesse de que en el momento de escribir, el valor fue visualizado en el puerto PORTB.

Para que este ejemplo funcione apropiadamente, es necesario marcar la librería EEPROM en la ventana Library Manager antes de compilar el programa:

- EEPROM

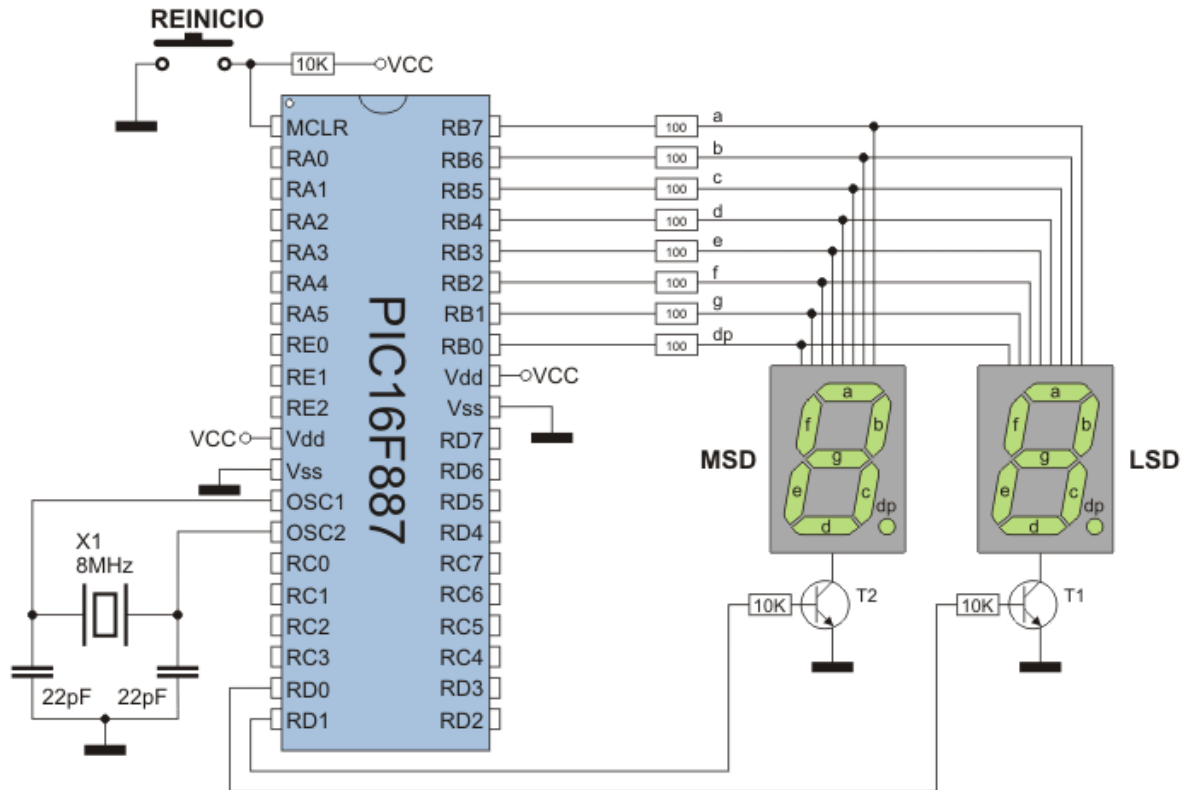
EJEMPLO 9

Contador de dos dígitos LED, multiplexión

En este ejemplo el microcontrolador funciona como un contador de dos dígitos. La variable *i* se incrementa (suficiente lentamente para ser visible) y su valor se visualiza en un visualizador de dos dígitos LED (99-0). El punto es habilitar una conversión de un número binario en un decimal y partirlo en dos dígitos (en decenas y unidades). Como los segmentos del visualizador LED se conectan en paralelo, es necesario asegurar que

alternen rápidamente para tener una impresión de que emiten la luz simultáneamente (multiplexión por división en tiempo).

En este ejemplo, el temporizador TMR0 está encargado de la multiplexión por división en tiempo, mientras que la función mask convierte un número binario a formato decimal.



*/*Cabecera******

```
unsigned short mask(unsigned short num);
unsigned short digit_no, digit10, digit1, digit, i;
```

```
void interrupt() {
  if (digit_no == 0) {
    PORTA = 0; // Apagar ambos visualizadores
    PORTD = digit1; // Colocar máscara para visualizar unidades en el
    // puerto PORTD
    PORTA = 1; // Encender el visualizador para las unidades (LSD)
    digit_no = 1;
  } else {
    PORTA = 0; // Apagar ambos visualizadores
    PORTD = digit10; // Colocar máscara para visualizar decenas en el
    // puerto PORTD
    PORTA = 2; // Encender el visualizador para las decenas (MSD)
    digit_no = 0;
  }
  TMR0 = 0; // Reiniciar el contador TMRO
  INTCON = 0x20; // Bit T0IF=0, T0IE=1
}
```

```
void main() {
```

```

OPTION_REG = 0x80; // Ajustar el temporizador TMR0
TMR0 = 0;
INTCON = 0xA0; // Deshabilitar las interrupciones PEIE,INTE,RBIE,T0IE
PORTA = 0; // Apagar ambos visualizadores
TRISA = 0; // Todos los pines del puerto PORTA se configuran

// como salidas
PORTD = 0; // Apagar todos los segmentos del visualizador
TRISD = 0; // Todos los pines del puerto PORTD se configuran

// como salidas
do {
    for (i = 0; i<=99; i++) { // Contar de 0 a 99
        digit = i % 10u;
        digit1 = mask(digit); // Preparar la máscara para visualizar unidades
        digit = (char)(i / 10u) % 10u;
        digit10 = mask(digit); // Preparar la máscara para visualizar decenas
        Delay_ms(1000);
    }
} while (1); // Bucle infinito
}

```

mask.c

```

/*Cabecera******/
unsigned short mask(unsigned short num) {
    switch (num) {
        case 0 : return 0x3F;
        case 1 : return 0x06;
        case 2 : return 0x5B;
        case 3 : return 0x4F;
        case 4 : return 0x66;
        case 5 : return 0x6D;
        case 6 : return 0x7D;
        case 7 : return 0x07;
        case 8 : return 0x7F;
        case 9 : return 0x6F;
    }
}

```

Para que este ejemplo funcione apropiadamente, es necesario incluir el archivo mask.c en el proyecto (aparte del archivo example9.c) en la ventana Project Manager antes de compilar el programa:

Example9.mcppi - Sources - Add File To Project

- mask.c
- example9.c

EJEMPLO 10

Utilizar el visualizador LCD

Este ejemplo muestra cómo utilizar un visualizador LCD alfanumérico. Las librerías de funciones simplifican este programa, lo que significa que al final el esfuerzo para crear el software vale la pena.

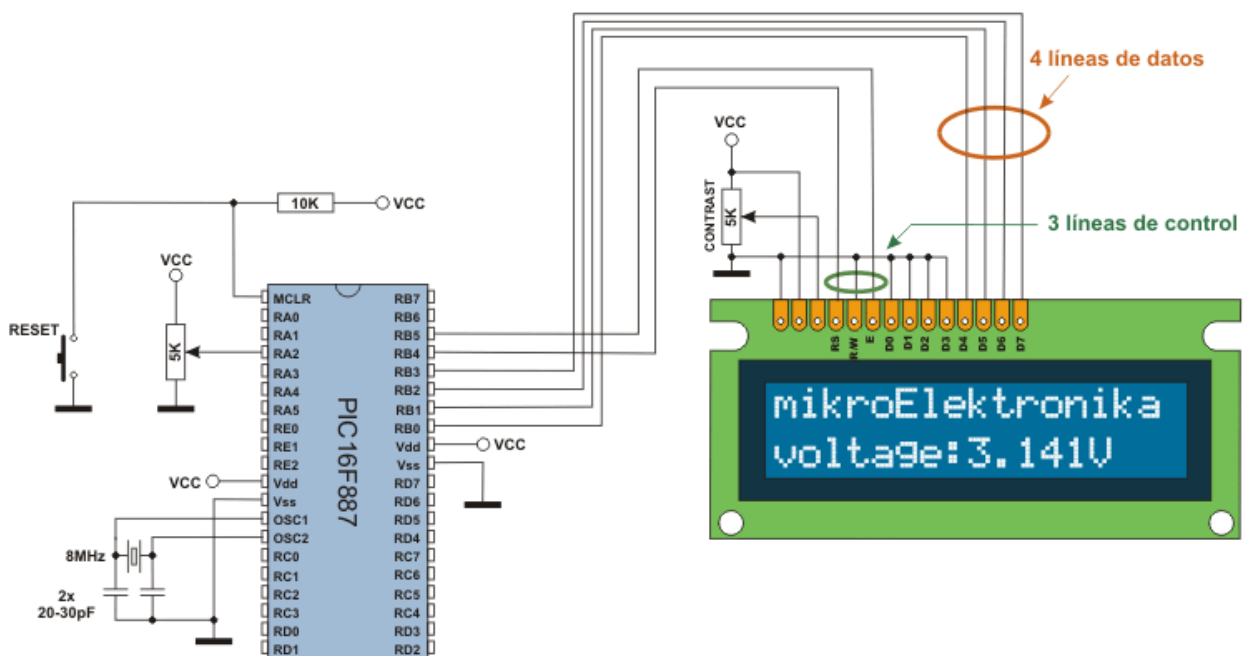
Un mensaje escrito en dos líneas aparece en el visualizador:

mikroElektronika
LCD example

Dos segundos más tarde, el mensaje en la segunda línea cambia, y se visualiza el voltaje presente en la entrada del convertidor A/D (el pin RA2). Por ejemplo:

mikroElektronika
voltage:3.141V

En un dispositivo real se puede visualizar temperatura actual o algún otro valor medido en vez de voltaje.



Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana Library Manager antes de compilar el programa:

- ADC
- LCD

*/*Cabecera******

```
// Conexiones del módulo LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
```

```

sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Final de las conexiones del módulo LCD

// Declarar variables
unsigned char ch;
unsigned int adc_rd;
char *text;
long tlong;

void main() {
    INTCON = 0;           // Todas las interrupciones deshabilitadas
    ANSEL = 0x04;        // Pin RA2 se configura como una entrada analógica
    TRISA = 0x04;
    ANSELH = 0;          // Los demás pines se configuran como digitales

    Lcd_Init();          // Inicialización del visualizador LCD
    Lcd_Cmd(_LCD_CURSOR_OFF); // Comando LCD (apagar el cursor)
    Lcd_Cmd(_LCD_CLEAR); // Comando LCD (borrar el LCD)

    text = "mikroElektronika"; // Definir el primer mensaje
    Lcd_Out(1,1,text); // Escribir el primer mensaje en la primera línea

    text = "LCD example"; // Definir el segundo mensaje
    Lcd_Out(2,1,text); // Definir el primer mensaje

    ADCON1 = 0x82;       // Voltaje de referencia para la conversión A/D es VCC
    TRISA = 0xFF;        // Todos los pines del puerto PORTA se configuran como entradas
    Delay_ms(2000);

    text = "voltage: "; // Definir el tercer mensaje

    while (1) {
        adc_rd = ADC_Read(2); // Conversión A/D. Pin RA2 es una entrada.
        Lcd_Out(2,1,text); // Escribir el resultado en la segunda línea
        tlong = (long)adc_rd * 5000; // Convertir el resultado en milivoltios
        tlong = tlong / 1023; // 0..1023 -> 0-5000mV
        ch = tlong / 1000; // Extraer voltios (miles de milivoltios)

        // del resultado
        Lcd_Chr(2,9,48+ch); // Escribir resultado en formato ASCII
        Lcd_Chr_CP('.');
        ch = (tlong / 100) % 10; // Extraer centenas de milivoltios
        Lcd_Chr_CP(48+ch); // Escribir resultado en formato ASCII
        ch = (tlong / 10) % 10; // Extraer decenas de milivoltios
        Lcd_Chr_CP(48+ch); // Escribir resultado en formato ASCII
        ch = tlong % 10; // Extraer unidades de milivoltios
        Lcd_Chr_CP(48+ch); // Escribir resultado en formato ASCII
        Lcd_Chr_CP('V');
        Delay_ms(1);
    }
}

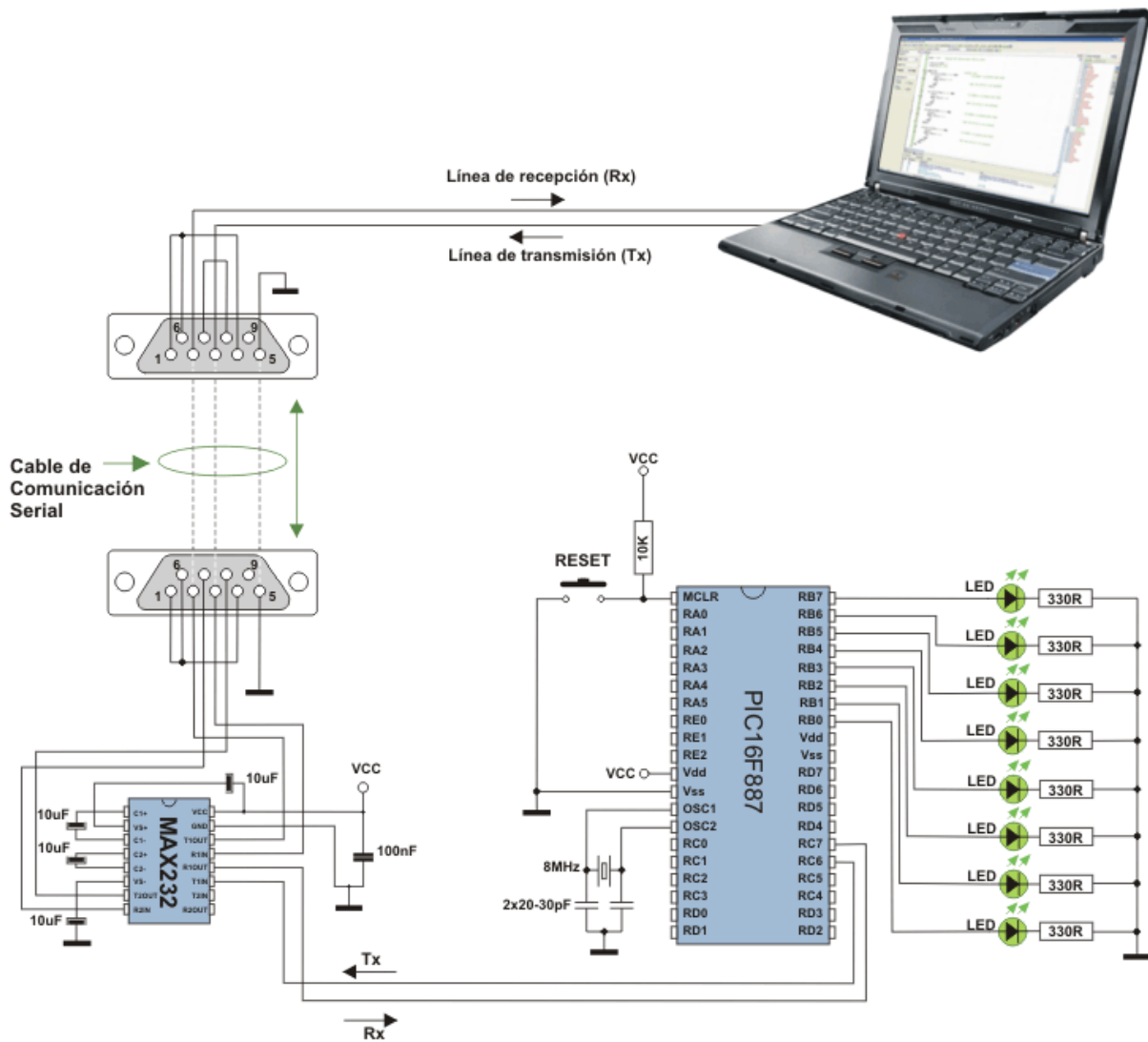
```

EJEMPLO 11

Comunicación serial RS-232

Este ejemplo muestra cómo utilizar el módulo EUSART del microcontrolador. La conexión a una PC se habilita por medio del estándar de comunicación RS-232. El

programa funciona de la siguiente manera. Cada byte recibido por medio de la comunicación serial se visualiza al utilizar los LEDs conectados al puerto PORTB y después se devuelve automáticamente al transmisor. Si ocurre un error en recepción, se lo indicará al encender el diodo LED. La manera más fácil es comprobar el funcionamiento del dispositivo en la práctica al utilizar un programa estándar de Windows denominado *Hyper Terminal*.



*/*Cabecera******

unsigned short i;

```
void main() {
    UART1_Init(19200); // Inicializar el módulo USART
    // (8 bits, tasa de baudios 19200, no hay bit
    // de paridad...)
```

```
while (1) {
    if (UART1_Data_Ready()) { // si se ha recibido un dato
        i = UART1_Read(); // leerlo
        UART1_Write(i); // enviarlo atrás
    }
}
```

}

Para que este ejemplo funcione apropiadamente, es necesario marcar la librería UART en la ventana Library Manager antes de compilar el programa:

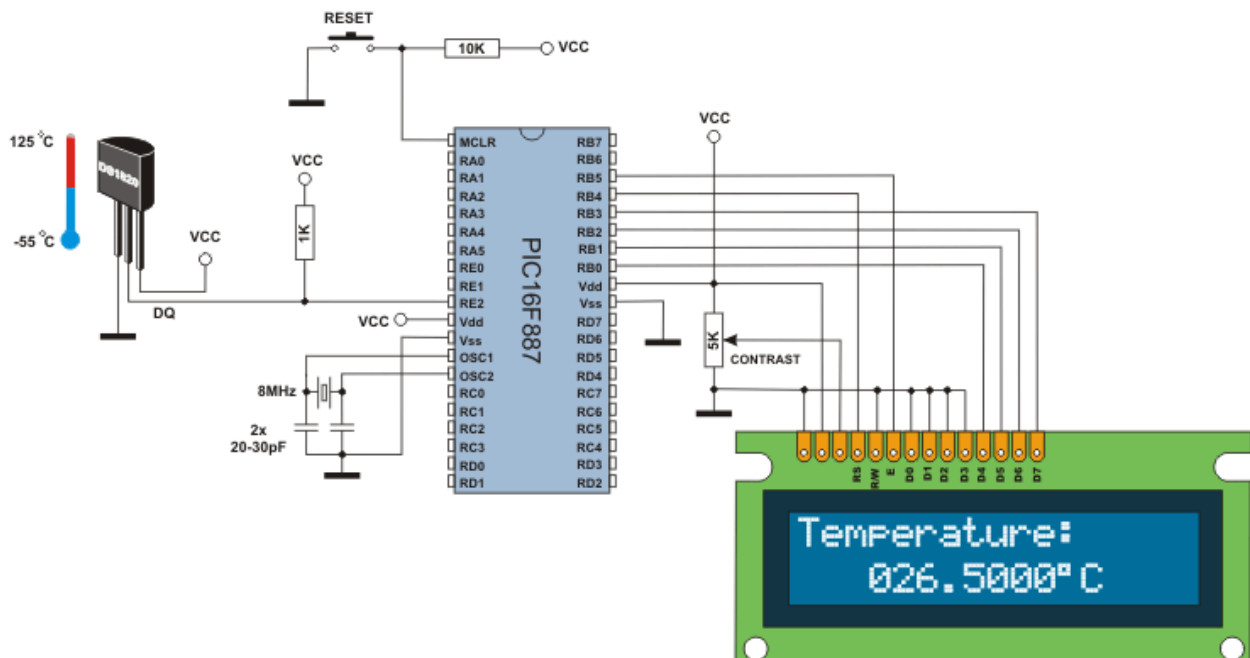
- UART

EJEMPLO 12

Medición de temperatura por medio del sensor DS1820. Uso del protocolo '1-wire'...

La medición de temperatura es una de las tareas más frecuentes realizadas por el microcontrolador. En este ejemplo, se utiliza un sensor DS1820 para medir. Es capaz de medir en el rango de 55 °C a 125 °C con exactitud de 0.5 °C. Para transmitir los datos al microcontrolador se utiliza un tipo especial de la comunicación serial denominado 1-wire. Debido al hecho de que estos sensores son simples de utilizar y de grandes capacidades, los comandos utilizados para hacerlos funcionar y controlarlos tienen la forma de funciones almacenadas en la librería One_Wire. En total son las siguientes tres funciones:

- **Ow_Reset** se utiliza para reiniciar el sensor;
- **Ow_Read** se utiliza para recibir los datos del sensor; y
- **Ow_Write** se utiliza para enviar los comandos al sensor



Este ejemplo muestra la ventaja de utilizar librerías con las funciones listas para ser utilizadas. Concretamente, no tiene que examinar la documentación proporcionada por el fabricante para utilizar el sensor. Basta con copiar alguna de estas funciones en el programa. Si le interesa saber cómo se declaran, basta con pulsar sobre alguna de ellas y seleccione la opción Help.

```

/*Cabecera*****
// Conexiones del módulo LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Final de conexiones del módulo LCD

const unsigned short TEMP_RESOLUTION = 9;
char *text = "000.0000";
unsigned temp;

void Display_Temperature(unsigned int temp2write) {
    const unsigned short RES_SHIFT = TEMP_RESOLUTION - 8;
    char temp_whole;
    unsigned int temp_fraction;

    // comprobar si la temperatura es negativa
    if (temp2write & 0x8000) {
        text[0] = '-';
        temp2write = ~temp2write + 1;
    }

    // extraer temp_whole
    temp_whole = temp2write >> RES_SHIFT ;
    // convertir temp_whole en caracteres
    if (temp_whole/100)
        text[0] = temp_whole/100 + 48;
    else
        text[0] = '0';
    text[1] = (temp_whole/10)%10 + 48; // Extraer dígito de decenas
    text[2] = temp_whole%10 + 48; // Extraer dígito de unidades

    // extraer temp_fraction y convertirlo en unsigned int
    temp_fraction = temp2write << (4-RES_SHIFT);
    temp_fraction &= 0x000F;
    temp_fraction *= 625;

    // convertir temp_fraction en caracteres
    text[4] = temp_fraction/1000 + 48; // Extraer dígito de miles
    text[5] = (temp_fraction/100)%10 + 48; // Extraer dígito de centenas
    text[6] = (temp_fraction/10)%10 + 48; // Extraer dígito de decenas
    text[7] = temp_fraction%10 + 48; // Extraer dígito de unidades

    // Visualizar temperatura en el LCD
    Lcd_Out(2, 5, text);
}

void main() {
    ANSEL = 0; // Configurar los pines AN como digitales
    ANSELH = 0;
    C1ON_bit = 0; // Deshabilitar los comparadores
    C2ON_bit = 0;
}

```

```

Lcd_Init();           // Inicializar el LCD
Lcd_Cmd(_LCD_CLEAR); // Borrar el LCD
Lcd_Cmd(_LCD_CURSOR_OFF); // Apagar el cursor
Lcd_Out(1, 1, " Temperatura: ");

// Visualizar el carácter de grado, 'C' para centígrados
Lcd_Chr(2,13,223); // distintos visualizadores LCD tienen diferentes códigos

// de caracteres para el grado
// si ve la letra griega Alfa, intente introducir 178 en vez de 223
Lcd_Chr(2,14,'C');

//--- bucle principal
do {
  //--- realizar lectura de temperatura
  Ow_Reset(&PORTE, 2); // Señal de reinicio de Onewire
  Ow_Write(&PORTE, 2, 0xCC); // Ejecutar el comando SKIP_ROM
  Ow_Write(&PORTE, 2, 0x44); // Ejecutar el comando CONVERT_T
  Delay_us(120);
  Ow_Reset(&PORTE, 2);
  Ow_Write(&PORTE, 2, 0xCC); // Ejecutar el comando SKIP_ROM
  Ow_Write(&PORTE, 2, 0xBE); // Ejecutar el comando READ_SCRATCHPAD
  temp = Ow_Read(&PORTE, 2);
  temp = (Ow_Read(&PORTE, 2) << 8) + temp;

  //--- Formatear y visualizar el resultado en el LCD
  Display_Temperature(temp);
  Delay_ms(500);
} while (1);
}

```

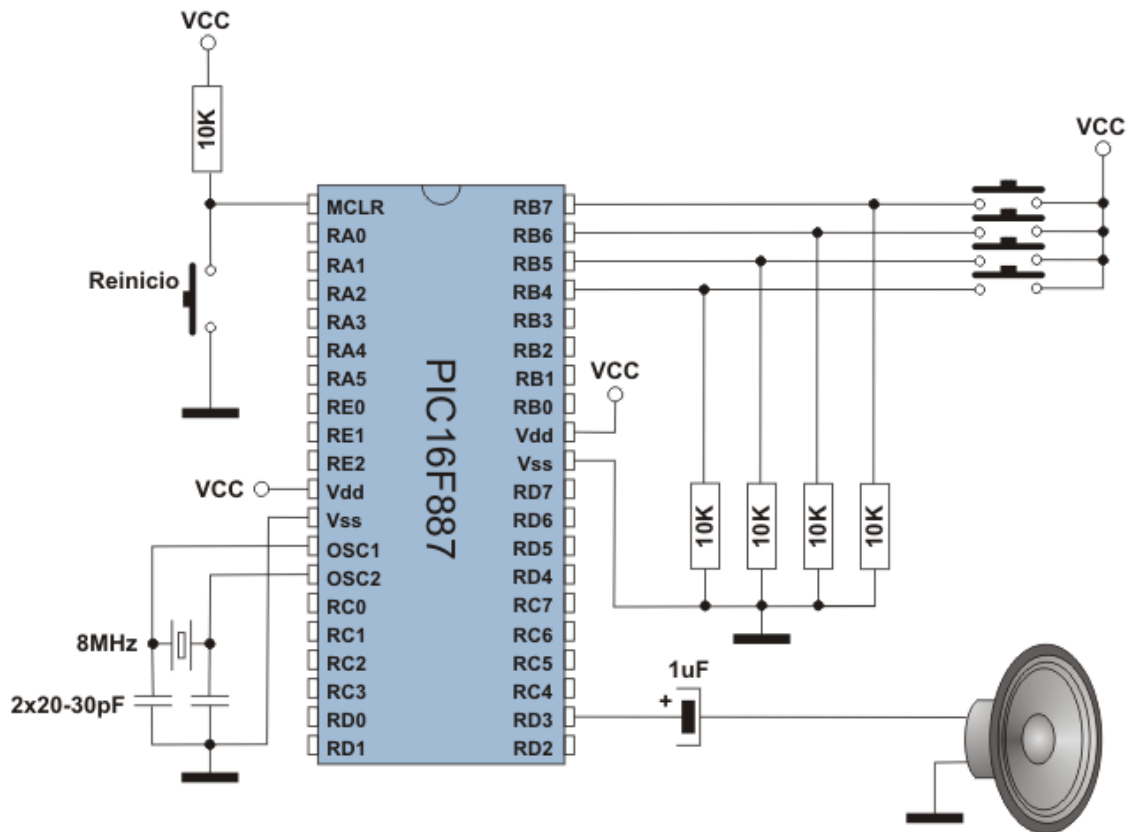
Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana Library Manager antes de compilar el programa:

- One_Wire
- LCD

EJEMPLO 13

Generación de sonido, librería de sonido...

Las señales de audio se utilizan con frecuencia cuando se necesita llamar la atención de usuario, confirmar que alguno de los botones se ha pulsado, avisar que se ha llegado hasta los valores mínimos o máximos etc. Pueden ser una simple señal de pitido así como melodías de una duración más larga o más corta. En este ejemplo se muestra la generación de sonido por medio de funciones que pertenecen a la librería Sound.



Además de estas funciones, la función Button que pertenece a la misma librería se utiliza para probar los botones de presión.

```

/*Cabecera*****
void Tone1() {
    Sound_Play(659, 250); // Frecuencia = 659Hz, duración = 250ms
}

void Tone2() {
    Sound_Play(698, 250); // Frecuencia = 698Hz, duración = 250ms
}

void Tone3() {
    Sound_Play(784, 250); // Frecuencia = 784Hz, duración = 250ms
}

void Melody1() { // Componer una melodía divertida 1
    Tone1(); Tone2(); Tone3(); Tone3();
    Tone1(); Tone2(); Tone3(); Tone3();
    Tone1(); Tone2(); Tone3();
    Tone1(); Tone2(); Tone3(); Tone3();
    Tone1(); Tone2(); Tone3();
    Tone3(); Tone3(); Tone2(); Tone2(); Tone1();
}

void ToneA() { // Tono A
    Sound_Play( 880, 50);

```

```

}

void ToneC() { // Tono C
  Sound_Play(1046, 50);
}

void ToneE() { // Tono E
  Sound_Play(1318, 50);
}

void Melody2() { // Componer una melodía divertida 2
  unsigned short i;

  for (i = 9; i > 0; i--) {
    ToneA(); ToneC(); ToneE();
  }
}

void main() {
  ANSEL = 0;           // Todos los pines de E/S son digitales
  ANSELH = 0;
  TRISB = 0xF0;       // Pines RB7-RB4 se configuran como entradas

  // RB3 se configura como salida
  Sound_Init(&PORTB, 3);
  Sound_Play(1000, 500);

  while (1) {
    if (Button(&PORTB,7,1,1)) // RB7 genera Tono1
      Tone1();
    while (PORTB & 0x80); // Esperar que se suelte el botón
    if (Button(&PORTB,6,1,1)) // RB6 genera Tono2
      Tone2();
    while (PORTB & 0x40); // Esperar que se suelte el botón
    if (Button(&PORTB,5,1,1)) // RB5 genera melodía 2
      Melody2();
    while (PORTB & 0x20); // Esperar que se suelte el botón
    if (Button(&PORTB,4,1,1)) // RB4 genera melodía 1
      Melody1();
    while (PORTB & 0x10); // Esperar que se suelte el botón
  }
}

```

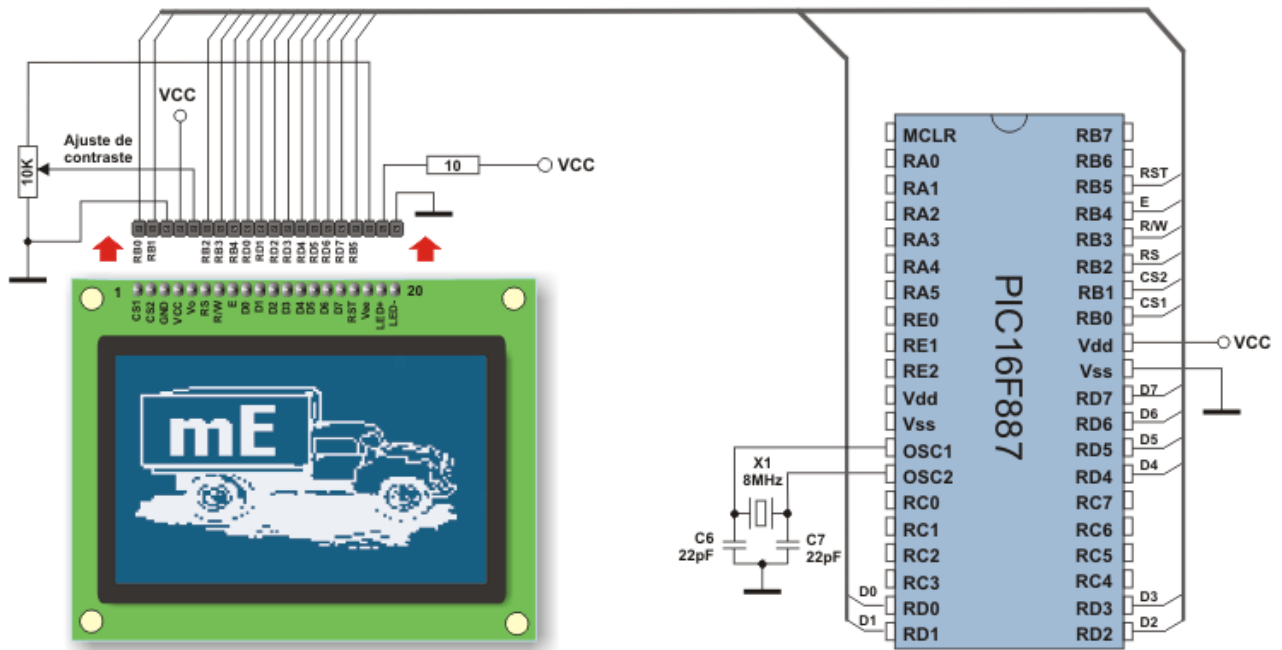
Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana Library Manager antes de compilar el programa:

- Button
- Sound

EJEMPLO 14

Utilizar el visualizador LCD gráfico

Un LCD gráfico (GLCD) proporciona un método avanzado para visualizar mensajes. Mientras que un LCD de caracteres puede visualizar sólo caracteres alfanuméricos, el LCD gráfico puede visualizar los mensajes en forma de dibujos y mapas de bits. El LCD gráfico utilizado con más frecuencia tiene una resolución de pantalla de 128x64 píxeles. El contraste de un GLCD se puede ajustar por medio del potenciómetro P1.



En este ejemplo el GLCD visualiza una secuencia de imágenes, de las que un mapa de bits representa un camión almacenado en otro archivo denominado truck_bmp.c. Para habilitar que este programa funcione apropiadamente, usted debe añadir este archivo como archivo fuente a su proyecto.

Las directivas del preprocesador incluidas en este ejemplo le permiten elegir si quiere visualizar toda la secuencia de imágenes o sólo una secuencia corta. Por defecto se visualizará toda la secuencia de imágenes. Sin embargo, al añadir un comentario delante de la directiva #define COMPLETE_EXAMPLE se deshabilita visualizar algunas imágenes de la secuencia. Si se comenta (o si se borra) esta directiva, las sentencias entre las directivas #ifdef COMPLETE_EXAMPLE y #endif no serán compiladas, así que no se ejecutarán.

```

/*Cabecera*****/

//Declaraciones-----
const code char truck_bmp[1024]; // Declarar la constante definida en truck_bmp.c
// para utilizarla en este archivo
//-----final-de-declaraciones

// Conexiones del módulo Glcd
char GLCD_DataPort at PORTD;
sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS at RB2_bit;
sbit GLCD_RW at RB3_bit;
sbit GLCD_EN at RB4_bit;
sbit GLCD_RST at RB5_bit;
sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction at TRISB2_bit;
sbit GLCD_RW_Direction at TRISB3_bit;
sbit GLCD_EN_Direction at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// Final de conexiones del módulo Glcd

void delay2S(){ // Función de tiempo de retardo de 2 segundos

```

```

    Delay_ms(2000);
}

void main() {
    unsigned short ii;
    char *someText;

    /* Esta directiva define un macro denominado COMPLETE_EXAMPLE. Más tarde en el
    programa, la directiva ifndef prueba si este macro está definido. Si se borra esta
    línea o si se transforma en un comentario, las secciones del código entre las
    directivas ifndef y endif no serán compiladas. */

    #define COMPLETE_EXAMPLE // Poner esta línea como un comentario si quiere
    // visualizar la versión corta de la secuencia

    ANSEL = 0;    // Configurar los pines AN como digitales
    ANSELH = 0;
    C1ON_bit = 0; // Deshabilitar comparadores
    C2ON_bit = 0;
    Glcd_Init(); // Inicializar el GLCD
    Glcd_Fill(0x00); // Borrar el GLCD

    while(1) { // bucle infinito, la secuencia se repite

        /* Dibujar la primera imagen */
        #ifndef COMPLETE_EXAMPLE
        Glcd_Image(truck_bmp); // Dibujar la imagen
        delay2S(); delay2S();
        #endif
        Glcd_Fill(0x00); // Borrar el GLCD

        /* Dibujar la segunda imagen */
        Glcd_Box(62,40,124,56,1); // Dibujar la caja
        Glcd_Rectangle(5,5,84,35,1); // Dibujar el rectángulo
        Glcd_Line(0, 0, 127, 63, 1); // Dibujar la línea

        delay2S();

        for(ii = 5; ii < 60; ii+=5) { // Dibujar líneas horizontales y verticales
            Delay_ms(250);
            Glcd_V_Line(2, 54, ii, 1);
            Glcd_H_Line(2, 120, ii, 1);
        }

        delay2S();
        Glcd_Fill(0x00); // Borrar el GLCD

        /* Dibujar la tercera imagen */

        #ifndef COMPLETE_EXAMPLE
        Glcd_Set_Font(Character8x7, 8, 7, 32); // Seleccionar la fuente, ver
        // __Lib_GLCDFonts.c en la carpeta Uses
        #endif

        Glcd_Write_Text("mikroE", 1, 7, 2); // Escribir la cadena
        for (ii = 1; ii <= 10; ii++) // Dibujar los círculos
            Glcd_Circle(63,32, 3*ii, 1);
        delay2S();

        Glcd_Box(12,20, 70,57, 2); // Dibujar la caja
        delay2S();

        /* Dibujar la cuarta imagen */

```



```

#ifdef COMPLETE_EXAMPLE
Glcd_Fill(0xFF); // Llenar el GLCD
Glcd_Set_Font(Character8x7, 8, 7, 32); // Cambiar de la fuente
someText = "8x7 Font";
Glcd_Write_Text(someText, 5, 0, 2); // Escribir la cadena
delay2S();
Glcd_Set_Font(System3x5, 3, 5, 32); // Cambiar de la fuente
someText = "3X5 CAPITALS ONLY";
Glcd_Write_Text(someText, 60, 2, 2); // Escribir la cadena
delay2S();
Glcd_Set_Font(font5x7, 5, 7, 32); // Cambiar de la fuente
someText = "5x7 Font";
Glcd_Write_Text(someText, 5, 4, 2); // Escribir la cadena
delay2S();
Glcd_Set_Font(FontSystem5x7_v2, 5, 7, 32); // Cambiar de la fuente
someText = "5x7 Font (v2)";
Glcd_Write_Text(someText, 5, 6, 2); // Escribir la cadena
delay2S();
#endif
}
}

```

truck_bmp.c:

```

/*Cabecera******/
unsigned char const truck_bmp[1024] = {
  0, 0, 0, 0, 0,248, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12,
  12, 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9, 9, 9, 9,
  9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 137,137,137,137,137,137,
  137,137,137,137,137,137,137, 9, 9, 9, 9, 9, 9, 9, 9, 9,
  9, 9, 13,253, 13,195, 6,252, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0,255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  240,240,240,240,240,224,224,240,240,240,240,240,224,192,192,224,
  240,240,240,240,240,224,192, 0, 0, 0,255,255,255,255,255,195,
  195,195,195,195,195,195, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0,255,240, 79,224,255, 96, 96, 96, 32, 32, 32, 32, 32,
  32, 32, 32, 32, 32, 32, 32, 32, 64, 64, 64, 64,128, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  255,255,255,255,255, 0, 0, 0, 0,255,255,255,255,255, 0, 0,
  0, 0,255,255,255,255,255, 0, 0, 0,255,255,255,255,255,129,
  129,129,129,129,129,129,128, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0,255, 1,248, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
  8, 8, 8, 8, 16,224, 24, 36,196, 70,130,130,133,217,102,112,
  160,192, 96, 96, 32, 32,160,160,224,224,192, 64, 64,128,128,192,
  64,128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 63, 96, 96, 96,224, 96, 96, 96, 96, 96,
  99, 99, 99, 99, 99, 96, 96, 96, 96, 99, 99, 99, 99, 96, 96,
  96, 96, 99, 99, 99, 99, 99, 96, 96, 96, 99, 99, 99, 99, 99,
  99, 99, 99, 99, 99, 99, 96, 96, 96, 96, 96, 96, 64, 64,
  64,224,224,255,246, 1, 14, 6, 6, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2,130, 67,114, 62, 35, 16, 16, 0, 7, 3, 3, 2,
  4, 4, 4, 4, 4, 4, 28, 16, 16, 16, 17, 17, 9, 9, 41,
  112, 32, 67, 5,240,126,174,128, 56, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
  1, 1,127,127,127,127,255,255,247,251,123,191, 95, 93,125,189,
  189, 63, 93, 89,177,115,243,229,207, 27, 63,119,255,207,191,255,

```

```

255,255,255,255,255,255,255,127,127,127,127,127,127,127,127,255,
255,255,127,127,125,120,120,120,120,120,248,120,120,120,120,120,
120,248,248,232,143, 0, 0, 0, 0, 0, 0, 0, 0, 0,128,240,248,
120,188,220, 92,252, 28, 28, 60, 92, 92, 60,120,248,248, 96,192,
143,168,216,136, 49, 68, 72, 50,160, 96, 0, 0, 0, 0, 0, 0,
0, 0, 0,128,192,248,248,248,248,252,254,254,254,254,254,254,
254,254,254,254,254,255,255,255,255,255,246,239,208,246,174,173,
169,128,209,208,224,247,249,255,255,252,220,240,127,255,223,255,
255,255,255,255,255,254,254,254,254,255,255,255,255,255,254,255,
255,255,255,255,255,255,254,254,254,254,254,254,254,254,254,254,
254,254,254,254,255,255,255,255,255,255,254,255,190,255,255,253,
240,239,221,223,254,168,136,170,196,208,228,230,248,127,126,156,
223,226,242,242,242,242,177, 32, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 1, 1, 3, 3, 3, 7, 7, 7, 7, 7, 15,
15, 15, 7, 15, 15, 15, 7, 7, 15, 14, 15, 13, 15, 47, 43, 43,
43, 43, 43, 47,111,239,255,253,253,255,254,255,255,255,255,255,
191,191,239,239,239,191,255,191,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,127,127,127,127,255,255,191,191,191,191,255,254,
255,253,255,255,255,251,255,255,255,127,125, 63, 31, 31, 31, 31,
31, 31, 63, 15, 15, 7, 7, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1, 3, 3, 3, 11, 11, 11, 11, 7, 3, 14, 6, 6,
6, 2, 18, 19, 19, 3, 23, 21, 21, 17, 1, 19, 19, 3, 6, 6,
14, 15, 15, 7, 15, 15, 15, 11, 2, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

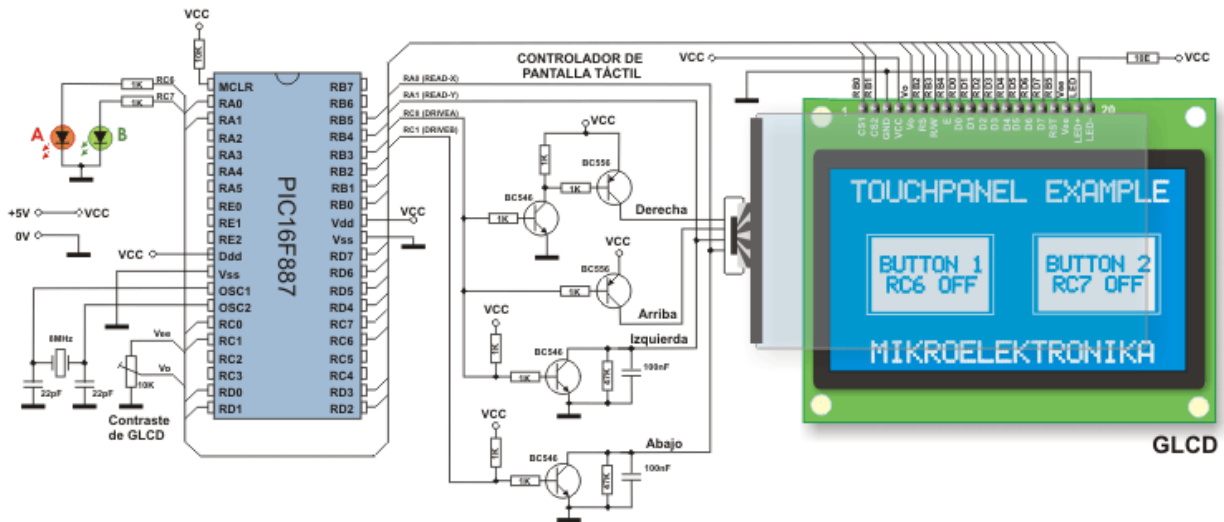
```

Para que este ejemplo funcione apropiadamente, es necesario marcar la librería GLCD en la ventana *Library Manager* antes de compilar el programa. Asimismo, es necesario incluir el documento *truck_bmp.c* en el proyecto.

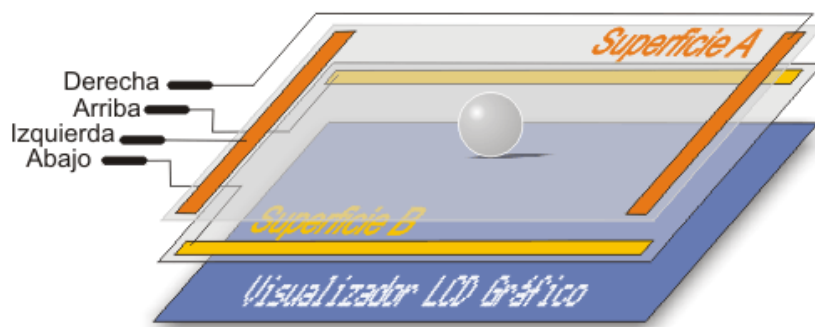
EJEMPLO 15

Utilizar el panel táctil...

Un panel táctil es un panel fino, autoadhesivo y transparente, colocado sobre la pantalla de un LCD gráfico. Es muy sensible a la presión así que un suave toque provoca algunos cambios en la señal de salida. Hay diferentes tipos de paneles táctiles. El más sencillo es el panel táctil resistivo del que se hablará aquí.

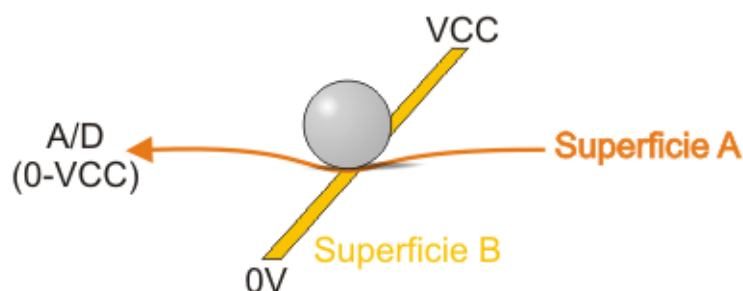


Un panel táctil está compuesto por dos láminas rígidas, formando una estructura de 'sándwich' que tiene capas resistivas en sus caras internas. La resistencia de estas capas no excede normalmente de 1Kohm. Los lados opuestos de las láminas disponen de los contactos para acceder a un cable plano.



El procedimiento para determinar las coordenadas de la posición del panel que ha sido presionado se puede dividir en dos pasos. El primero es determinación de la coordenada X, y el segundo es de determinar la coordenada Y de la posición.

Para determinar la coordenada X, es necesario conectar el contacto izquierdo en la superficie A a la masa (tierra) y el contacto derecho a la fuente de alimentación. Esto permite obtener un divisor de voltaje al presionar el panel táctil. El valor de voltaje obtenido en el divisor se puede leer en el contacto inferior de la superficie B. El voltaje variará en el rango de 0V al voltaje suministrado por la fuente de alimentación y depende de la coordenada X. Si el punto está próximo al contacto izquierdo de la superficie A, el voltaje estará próximo a 0V.



Para la determinación de la coordenada Y, es necesario conectar el contacto inferior de la superficie B a masa (tierra), mientras que el contacto superior se conectará a la fuente de alimentación. En este caso, el voltaje se puede leer en el contacto izquierdo de la superficie A.

Para conectar un panel táctil al microcontrolador es necesario crear un circuito para el control del panel táctil. Por medio de este circuito, el microcontrolador conecta los contactos adecuados del panel táctil a masa y a la voltaje de alimentación (como describimos anteriormente) para determinar las coordenadas X y Y. El contacto inferior de la superficie B y el contacto izquierdo de la superficie A están conectados al convertidor A/D del microcontrolador. Las coordenadas X e Y se determinan midiendo el voltaje en los respectivos contactos. El software consiste en mostrar un menú en una pantalla LCD gráfica, conmutar de encendido a apagado del panel táctil (control del panel táctil) y leer los valores del convertidor A/D que representan realmente las coordenadas X e Y de la posición.

Una vez determinadas las coordenadas, es posible decidir qué es lo que deseamos que haga el microcontrolador. En este ejemplo se explica cómo conmutar entre encendido y apagado dos pines digitales del microcontrolador, conectados a los LEDs A y B.

En este ejemplo se utilizan las funciones que pertenecen a las librerías Glcd y ADC.

Teniendo en cuenta que la superficie del panel táctil es ligeramente mayor que la del LCD gráfico, en caso de requerir una mayor precisión en la determinación de las coordenadas, es necesario incluir el software de calibración del panel táctil.

```
/* Cabecera *****/
```

```
// Conexiones del módulo Glcd
```

```
char GLCD_DataPort at PORTD;
sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS at RB2_bit;
sbit GLCD_RW at RB3_bit;
sbit GLCD_EN at RB4_bit;
sbit GLCD_RST at RB5_bit;
sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction at TRISB2_bit;
sbit GLCD_RW_Direction at TRISB3_bit;
sbit GLCD_EN_Direction at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// Final de conexiones del módulo Glcd
```

```
// Declaración de la cadena a visualizar en el GLCD
```

```
char msg1[] = "TOUCHPANEL EXAMPLE";
char msg2[] = "MIKROELEKTRONIKA";
char msg3[] = "BUTTON1";
char msg4[] = "BUTTON2";
char msg5[] = "RC6 OFF";
char msg6[] = "RC7 OFF";
char msg7[] = "RC6 ON ";
char msg8[] = "RC7 ON ";
```

```
// Declaración de variables globales
```

```
long x_coord, y_coord, x_coord128, y_coord64; // almacenar la posición de las
// coordenadas x e y
```

```

// Leer la coordenada X
unsigned int GetX() {
    //reading X
    PORTC.F0 = 1;    // DRIVEA = 1 (electrodo izquierdo (LEFT) conectado, electrodo
                    // derecho (RIGHT) conectado, electrodo superior (TOP)desconectado)
    PORTC.F1 = 0;    // DRIVEB = 0 (electrodo inferior (BOTTOM) desconectado)
    Delay_ms(5);
    return ADC_Read(0); // leer el valor de X de RA0(BOTTOM)
}

// Leer la coordenada Y
unsigned int GetY() {
    //Leer la Y
    PORTC.F0 = 0;    // DRIVEA = 0 (electrodo izquierdo (LEFT) desconectado, electrodo
                    // derecho (RIGHT) desconectado, electrodo superior (TOP) conectado)
    PORTC.F1 = 1;    // DRIVEB = 1 (electrodo inferior (BOTTOM) conectado)
    Delay_ms(5);
    return ADC_Read(1); // leer el valor de Y de RA1 (del eléctrodo izquierdo LEFT)
}

void main() {
    PORTA = 0x00;
    TRISA = 0x03; // RA0 y RA1 son entradas analógicas
    ANSEL = 0x03;
    ANSELH = 0; // Configurar otros pines AN como digitales de E/S
    PORTC = 0 ; // Todos los pines del puerto PORTC están a 0 (incluyendo los
                // pines RC6 y RC7)

    TRISC = 0 ; // PORTC es una salida

    // Inicialización del GLCD
    Glcd_Init();           // Glcd_Init_EP5
    Glcd_Set_Font(FontSystem5x7_v2, 5, 7, 32); // Seleccionar el tamaño de fuente 5x7
    Glcd_Fill(0);         // Borrar GLCD
    Glcd_Write_Text(msg1,10,0,1);
    Glcd_Write_Text(msg2,17,7,1);

    // Visualizar botones en el GLCD:
    Glcd_Rectangle(8,16,60,48,1);
    Glcd_Rectangle(68,16,120,48,1);
    Glcd_Box(10,18,58,46,1);
    Glcd_Box(70,18,118,46,1);

    // Visualizar los mensajes en los botones
    Glcd_Write_Text(msg3,14,3,0);
    Glcd_Write_Text(msg5,14,4,0);
    Glcd_Write_Text(msg4,74,3,0);
    Glcd_Write_Text(msg6,74,4,0);

    while (1) {
        // leer X-Y y convertirlo en la resolución de 128x64 píxeles
        x_coord = GetX();
        y_coord = GetY();
        x_coord128 = (x_coord * 128) / 1024;
        y_coord64 = 64 -((y_coord *64) / 1024);

        //Si BUTTON1 ha sido presionado
        if ((x_coord128 >= 10) && (x_coord128 <= 58) && (y_coord64 >= 18) &&
            (y_coord64 <= 46)) {
            if(PORTC.F6 == 0) {           // Si RC6 = 0
                PORTC.F6 = 1;           // Invertir el estado lógico del pin RC6
                Glcd_Write_Text(msg7,14,4,0); // Visualizar un nuevo mensaje: RC6 ON
            }
        }
    }

```

```

else { // Si RC6 = 1
    PORTC.F6 = 0;           // Invertir el estado lógico del pin RC6
    Glcd_Write_Text(msg5,14,4,0); // Visualizar un nuevo mensaje: RC6 OFF
}
}

// Si BUTTON2 ha sido presionado
if ((x_coord128 >= 70) && (x_coord128 <= 118) && (y_coord64 >= 18) &&
(y_coord64 <= 46)) {
    if(PORTC.F7 == 0) { // Si RC7 = 0
        PORTC.F7 = 1;           // Invertir el estado lógico del pin RC7
        Glcd_Write_Text(msg8,74,4,0); // Visualizar un nuevo mensaje: RC7 ON
    }
    else { // Si RC7 = 1
        PORTC.F7 = 0;           // Invertir el estado lógico del pin RC7
        Glcd_Write_Text(msg6,74,4,0); // Visualizar un nuevo mensaje: RC7 OFF
    }
}
}
Delay_ms(100);
}
}

```

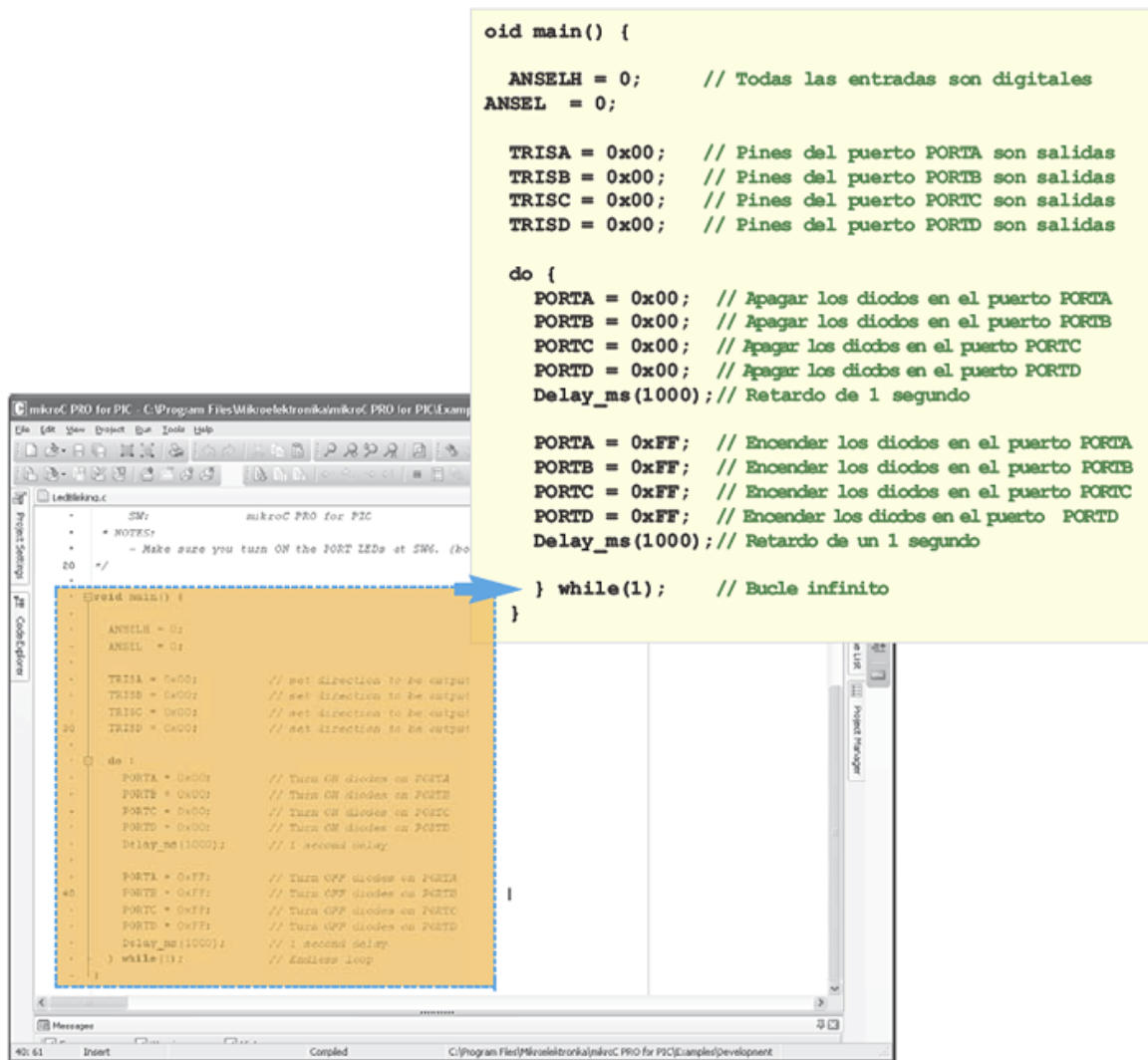
Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana *Library Manager* antes de compilar el programa.

- GLCD
- ADC
- C_Stdlib

VAMOS A EMPEZAR...

Los programas especiales en el entorno de Windows se utilizan para escribir un programa para el microcontrolador. Este libro describe el programa denominado mikroC PRO for PIC. La ventaja principal de este programa son las herramientas adicionales instaladas para facilitar el proceso de desarrollo.

Si tiene experiencia en escribir programas, entonces sabe que se trata de escribir todas las instrucciones en el orden en el que se deben ejecutar por el microcontrolador y observar las reglas del lenguaje C. En otras palabras, sólo tiene que seguir su idea al escribir el programa. ¡Esto es todo!



```

oid main() {

    ANSELH = 0;    // Todas las entradas son digitales
    ANSEL  = 0;

    TRISA = 0x00; // Pines del puerto PORTA son salidas
    TRISB = 0x00; // Pines del puerto PORTB son salidas
    TRISC = 0x00; // Pines del puerto PORTC son salidas
    TRISD = 0x00; // Pines del puerto PORTD son salidas

    do {
        PORTA = 0x00; // Apagar los diodos en el puerto PORTA
        PORTB = 0x00; // Apagar los diodos en el puerto PORTB
        PORTC = 0x00; // Apagar los diodos en el puerto PORTC
        PORTD = 0x00; // Apagar los diodos en el puerto PORTD
        Delay_ms(1000); // Retardo de 1 segundo

        PORTA = 0xFF; // Encender los diodos en el puerto PORTA
        PORTB = 0xFF; // Encender los diodos en el puerto PORTB
        PORTC = 0xFF; // Encender los diodos en el puerto PORTC
        PORTD = 0xFF; // Encender los diodos en el puerto PORTD
        Delay_ms(1000); // Retardo de un 1 segundo

    } while(1); // Bucle infinito
}

```

COMPILACIÓN DE PROGRAMA

El microcontrolador no entiende los lenguajes de alto nivel de programación, de ahí que sea necesario compilar el programa en lenguaje máquina. Basta con pulsar sólo una vez sobre el icono apropiado dentro del compilador para crear un documento nuevo con extensión .hex. En realidad, es el mismo programa, pero compilado en lenguaje máquina que el microcontrolador entiende perfectamente. Este programa se le denomina con frecuencia un código hex y forma una secuencia de números hexadecimales aparentemente sin significado.

```

:03000000020100FA1001000075813F
7590FFB29012010D80F97A1479D40
90110003278589EAF3698E8EB25B
A585FEA2569AD96E6D8FED9FAD
AF6DD00000001FF255AFED589EA
F3698E8EB25BA585FEA2569AD96
DAC59700D00000278E6D8FED9FA
DAF6DD00000001FF255AFED8FED
9FADAF6DD000F7590FFB29013278
E6D8FED9FADAF6DD00000001FF2
55AFED589EAF3698E8EB25BA585
FEA2569AD96DAC59D9FADAF6D
D00000001FF255AFED8FED9FADA
F6DD000F7590FFB29013278E6D82
78E6D8FED9FA589EAF3698E8EB2
5BA585FEA2569AD96DAF6DD000
00001FF2DAF6DD00000001FF255A
ADAF6DD00000001FF255AFED8FE
D9FA

```

Una vez compilado, el programa se debe cargar en el chip. Usted necesita un hardware apropiado para hacerlo posible - un programador.

PROGRAMAR EL MICROCONTROLADOR

Como hemos mencionado, para habilitar cargar un código hex en el microcontrolador es necesario proporcionar un dispositivo especial, denominado el programador, con software apropiado. Un gran número de programas y circuitos electrónicos utilizados con este propósito se pueden encontrar en Internet. El procedimiento es básicamente el mismo para todos ellos y se parece a lo siguiente:

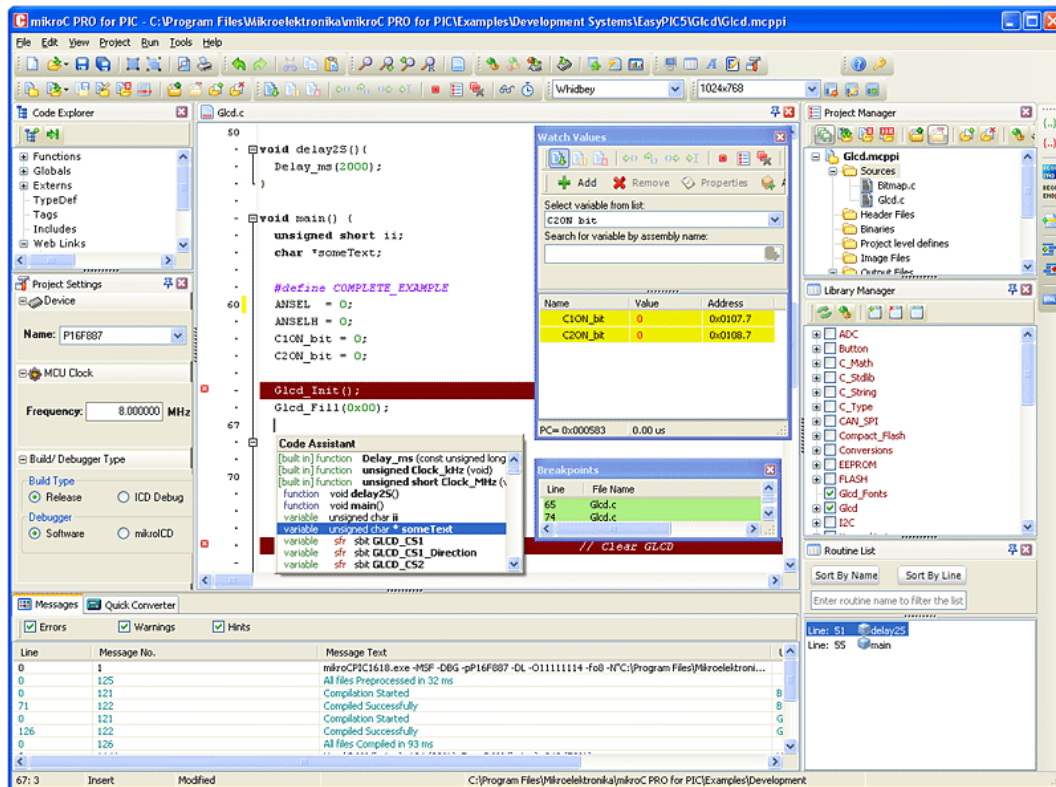
1. Coloque el microcontrolador en el zócalo apropiado del programador;
2. Utilice un cable adecuado para conectar el programador a una PC;
3. Abra el programa en código hex dentro de software del programador, ajuste varios parámetros, y pulse sobre el icono para transmitir el código. Pocos segundos después, una secuencia de ceros y unos se va a programar en el microcontrolador.

Sólo ha quedado instalar el chip programado en el dispositivo destino. Si es necesario hacer algunos cambios en el programa, el procedimiento anterior se puede repetir un número ilimitado de veces.

¿SERÁ UN FINAL FELIZ?

Esta sección describe en breve el uso del programa (compilador) *mikroC PRO for PIC* y del software de programación (programador) *PICflash*. Todo es muy simple...

Usted ya tiene instalado el *mikroC PRO for PIC*, ¿verdad? Al iniciarlo, abra un proyecto nuevo y un documento nuevo con extensión `.c` dentro del mismo. Escriba su programa...



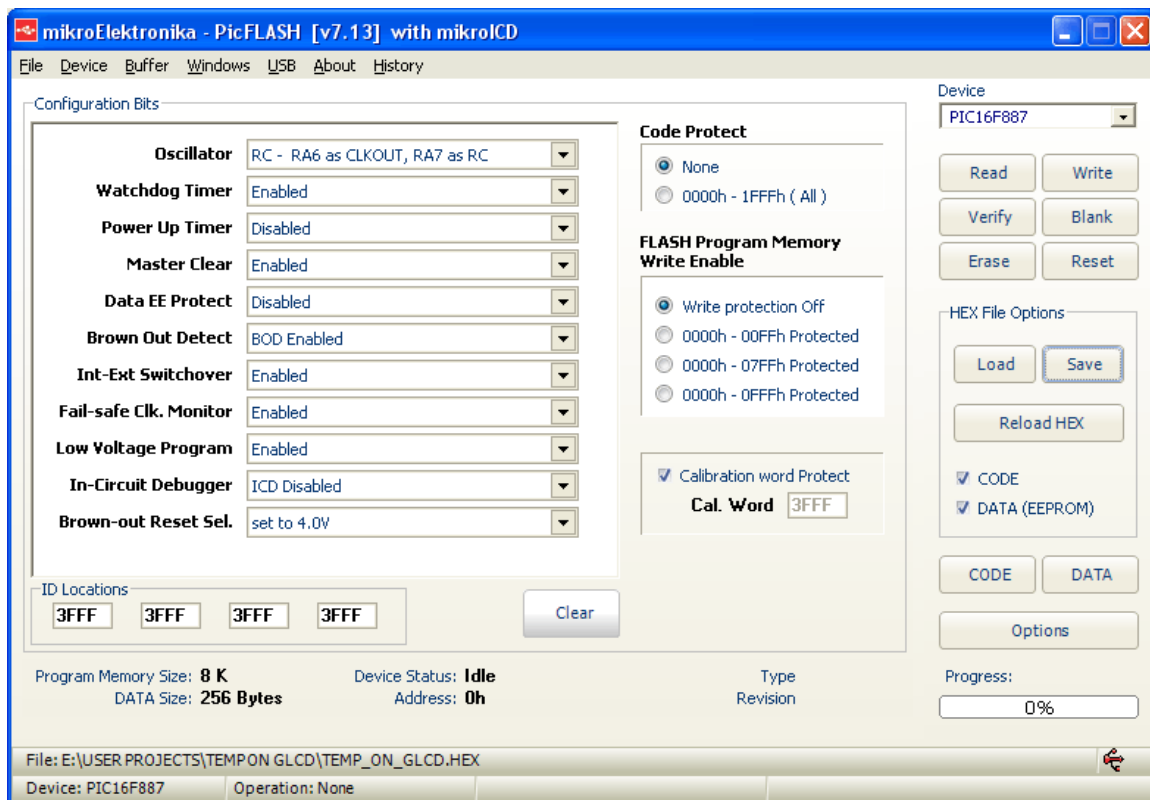
OK. The program has been written and tested with the simulator. It did not report any errors during the process of compiling into the *hex code*? It seems that everything is under control...

De acuerdo. El programa ha sido escrito y probado con el simulador. ¿No ha informado de ningún error durante el proceso de compilación en el código hex? Parece que todo funciona perfecto...

El programa ha sido compilado con éxito. Sólo queda cargarlo en el microcontrolador. Ahora necesita un *programador* que está compuesto por software y hardware. Inicie el programa *PICFlash*.

La configuración es simple y no hacen falta explicaciones adicionales (tipo de microcontrolador, frecuencia y reloj del oscilador etc.).

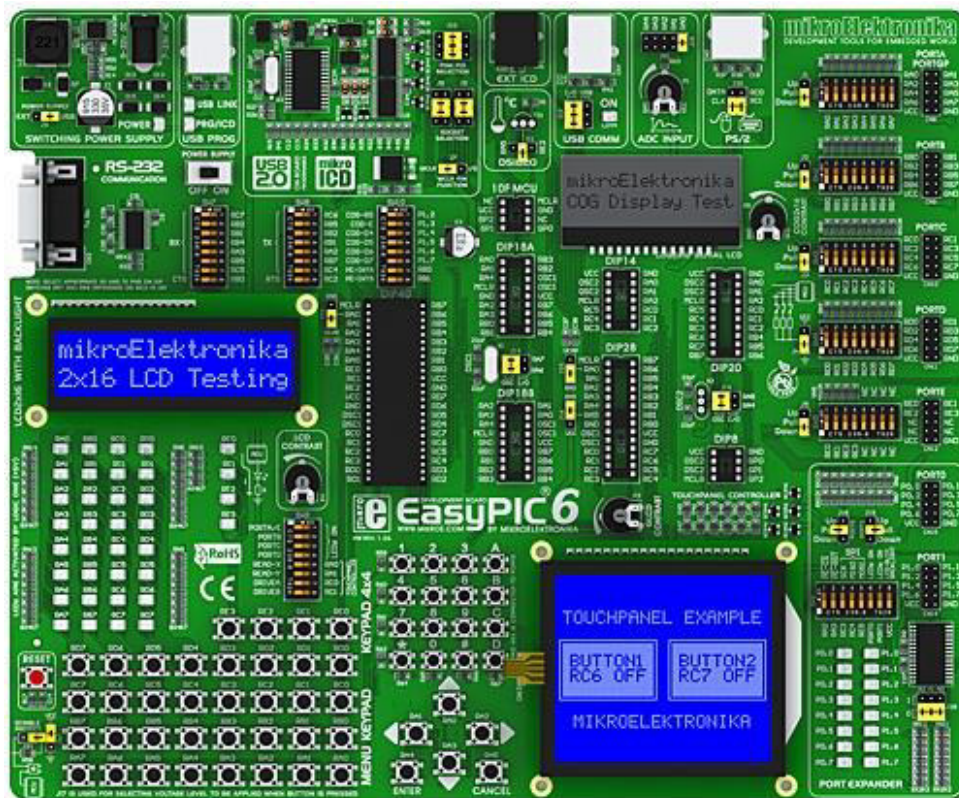
- Conecte la PC con el hardware del programador por un cable USB;
- Cargue el código hex utilizando el comando: File a Load HEX; y
- Pulse sobre el botón Write y espere...



¡Esto es todo! El microcontrolador está programado y todo está listo para su funcionamiento. Si no está satisfecho, haga algunos cambios en el programa y repita el procedimiento. ¿Hasta cuándo? Hasta quedar satisfecho...

SISTEMAS DE DESARROLLO

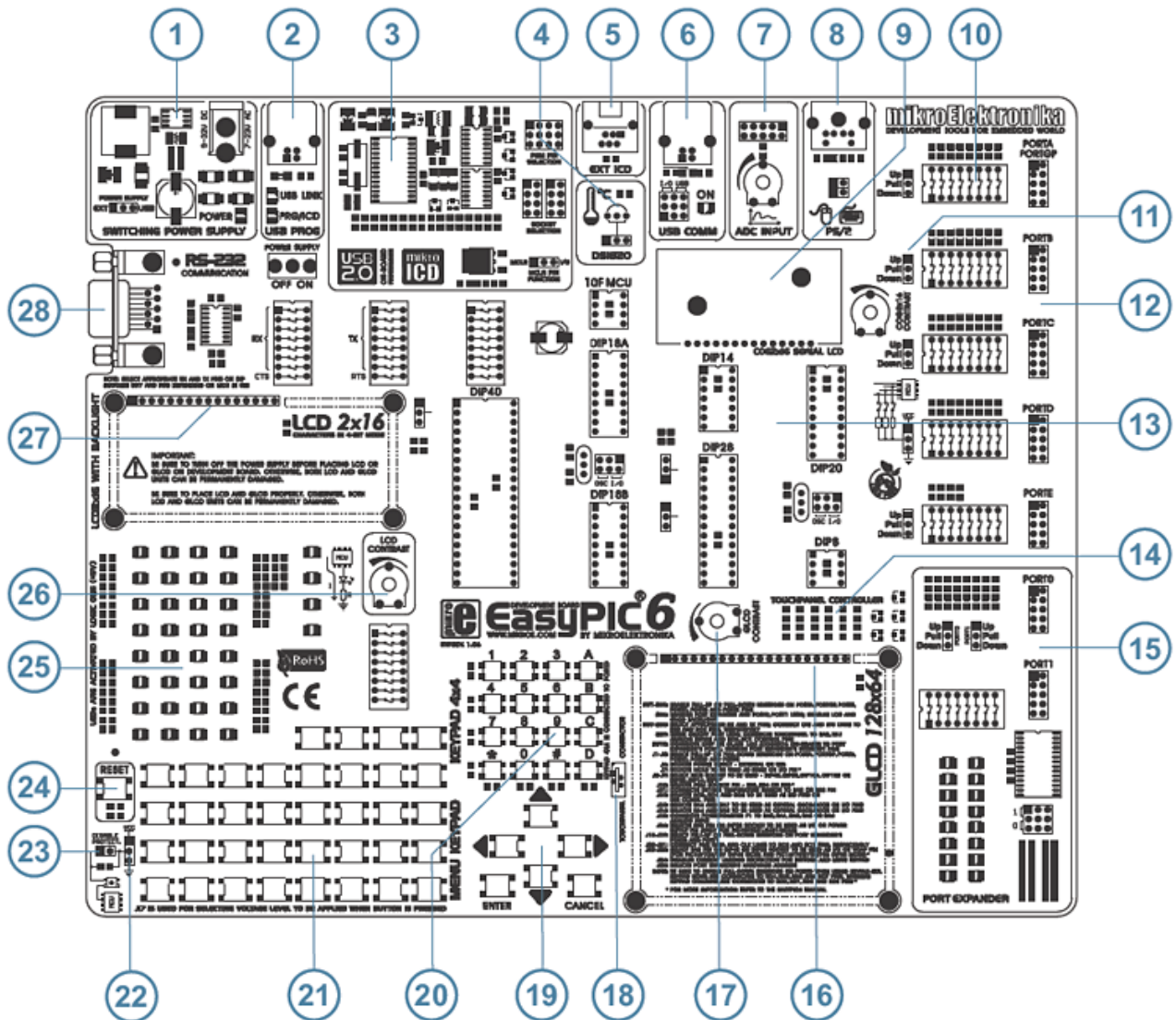
Un dispositivo que puede simular cualquier dispositivo en la fase de prueba, es denominado un sistema de desarrollo. Aparte del programador, unidad de alimentación, zócalo del microcontrolador, el sistema de desarrollo dispone de los componentes para activar los pines de entrada y monitorear los pines de salida. La versión más simple tiene cada pin conectado a su respectivo botón de presión y un LED.



Una versión de calidad alta tiene los pines conectados a los visualizadores LED, visualizadores LCD, sensores de temperatura u otros componentes por los que puede estar compuesto un dispositivo destino. Si es necesario, todos estos periféricos pueden estar conectados al microcontrolador por medio de los puentes. Esto permite probar el programa entero en la práctica aún durante el proceso de desarrollo, porque el microcontrolador no “sabe o no le interesa” si su entrada está activada por un botón de presión o un sensor incorporado en un dispositivo real.

Si dispone de un sistema de desarrollo, el proceso de programar y probar un programa es aún más sencillo. Teniendo en cuenta que el compilador mikroC PRO for PIC (en su PC) y el hardware del programador PICflash (en su sistema de desarrollo) colaboran perfectamente, el proceso de compilar un programa y programar el microcontrolador se lleva a cabo en un simple paso - al pulsar sobre el icono Build and Program dentro del compilador. Desde este momento, cualquier cambio en el programa afectará inmediatamente al funcionamiento de alguno de los componentes del sistema de desarrollo.

Características principales del sistema de desarrollo EasyPIC6



1. Regulador de voltaje de alimentación
2. Conector USB para el programador en la placa
3. Programador USB 2.0 con soporte de mikroICD
4. Zócalo para el sensor de temperatura DS1820
5. Conector para el depurador externo (ICD2 o ICD3) de Microchip
6. Conector para la comunicación USB
7. Entradas de prueba del convertidor A/D
8. Conector PS/2
9. LCD 2x16 en la placa
10. Interruptores DIP permiten el funcionamiento de las resistencias pull-up/pull-down
11. Puente para seleccionar las resistencias pull-up/pull-down
12. Conectores de los puertos E/S
13. Zócalo para colocar el microcontrolador PIC
14. Controlador del panel tácti
15. Extensor de puertos
16. Conector del LCD gráfico 128x64
17. Potenciómetro de contraste del LCD gráfico

18. Conector de panel táctil
19. Teclado Menu
20. Teclado 4x4
21. Botones de presión para simular las entradas digitales
22. Puente para seleccionar el estado lógico de los botones de presión
23. Puente para poner en cortocircuito la resistencia de protección
24. Botón para reiniciar el microcontrolador
25. 36 diodos LED indican el estado lógico de los pines
26. Ajuste de contraste del LCD alfanumérico
27. Conector del LCD alfanumérico
28. Conector para la comunicación RS-232