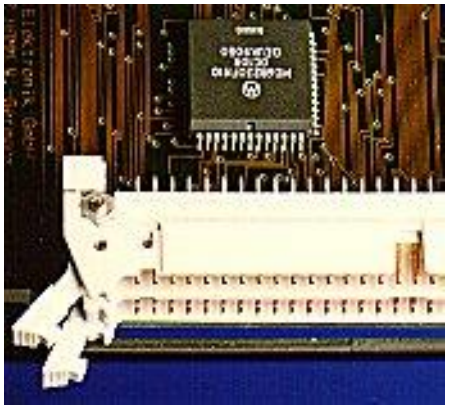


Puertos de Entrada y Salida

Características generales de los puertos en el PIC16F877

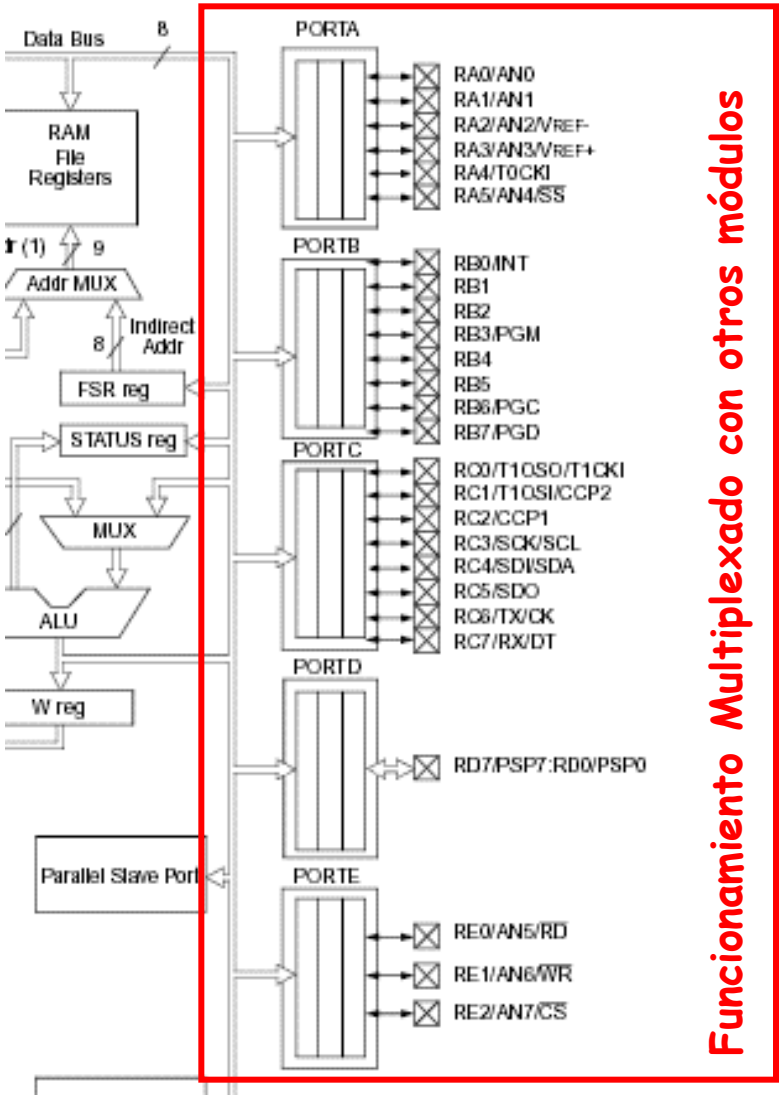
Son 5 Puertos de E/S configurables

PORTA: 6 pines (dir 0x05)
PORTB: 8 pines (dir 0x06 y 0x106)
PORTC: 8 pines (dir 0x07)
PORTD: 8 pines (dir 0x08)
PORTE: 3 pines (dir 0x09)

TOTAL: 33 pines de E/S

Dirección de los datos configurables registros de dirección de datos:

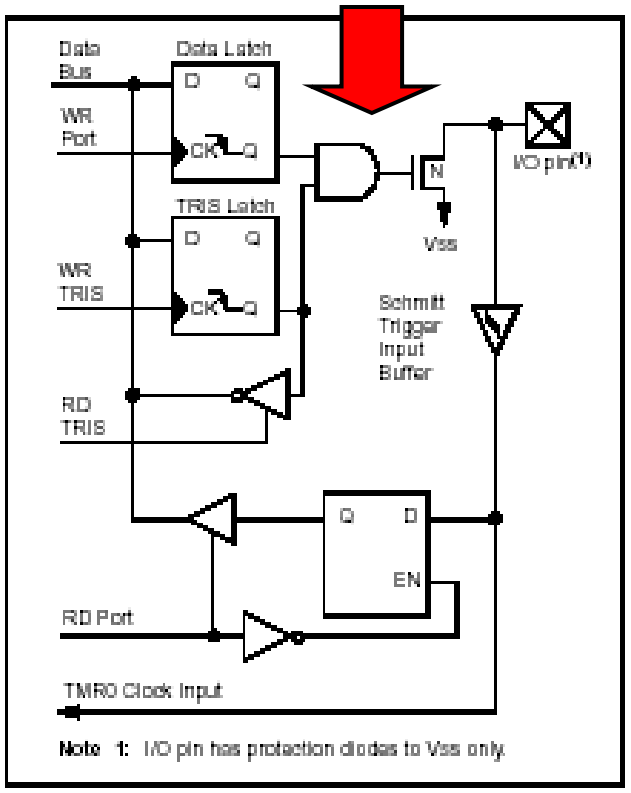
TRISA	(dir 0x85)
TRISB	(dir 0x86 y 0x186)
TRISC	(dir 0x87)
TRISD	(dir 0x88)
TRISE	(dir 0x89)

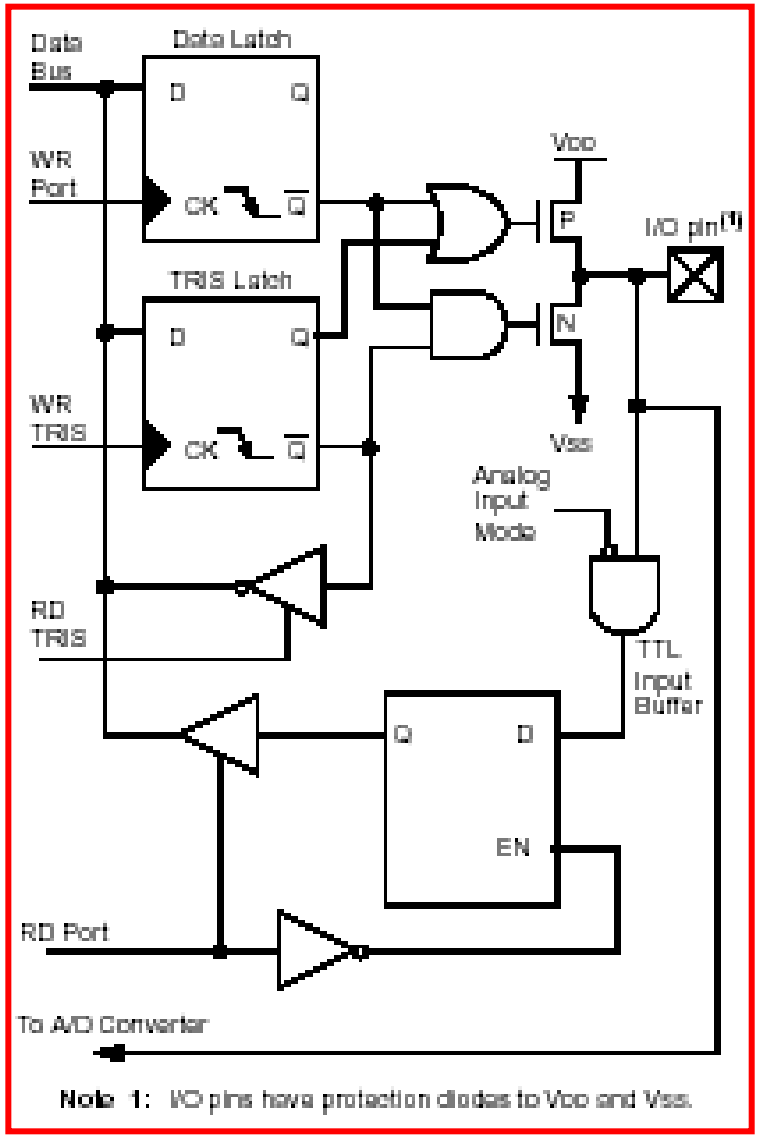


PORTA

Diagrama de Bloques de los pines RA5, RA3, RA2, RA1 y RA0 (multiplexadas con módulo A/D)

El pin RA4 salida de drenador abierto (entrada multiplexada con T0CKI)





PORTB

Es un puerto de 8 líneas bidireccionales

Tiene la posibilidad de activar unas resistencias de polarización en las líneas definidas como entradas (pull-up resistors con OPTION_REG<7>=0)
En cuanto pasan a ser salidas se desactivan las resistencias

RB0/INT puede generar una interrupción ante la aparición de un flanco (configurable de subida o de bajada con OPTION_REG<6>)

RB3 se emplea para programar con baja tensión (LVP)

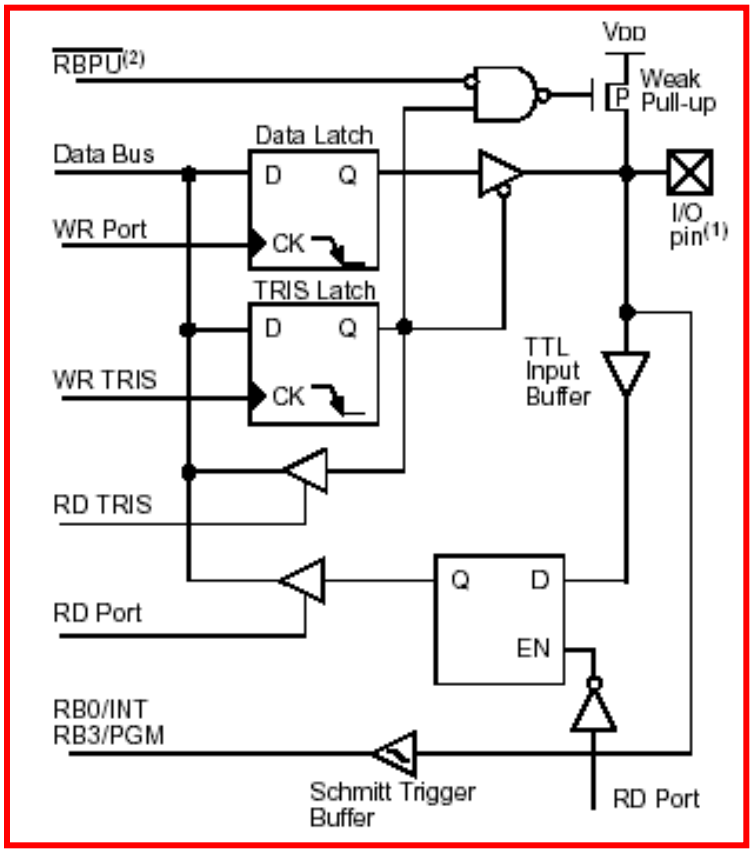


Diagrama de Bloques de RB3 a RB0

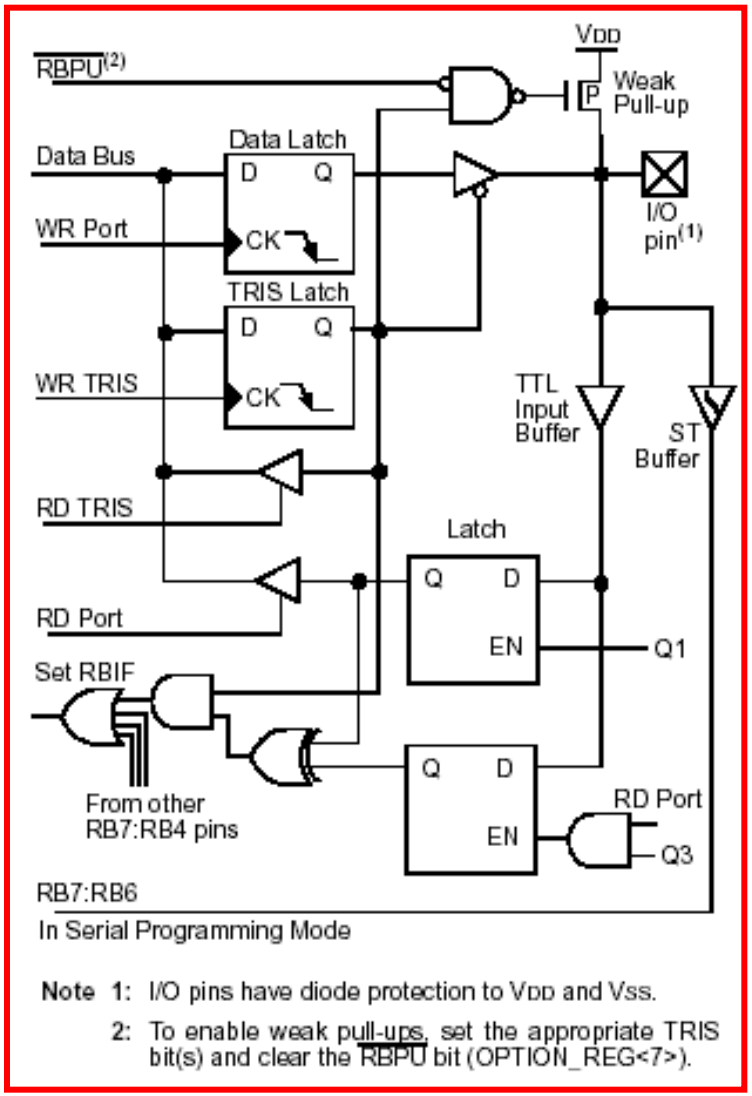
PORTB (II)

Las líneas RB4 a RB7 pueden generar una interrupción si se detecta cambio entre el estado actual y el de la última lectura del puerto

Las líneas RB6 y RB7 son utilizadas para la programación del microcontrolador cuando éste trabaja en modo depuración (debugger)

Cuando trabajemos con el ICD2 como debugger, no será posible utilizar esas dos líneas

Diagrama de bloques de líneas RB7 a RB4



PORTC

Puerto de 8 líneas bidireccionales

Los pines están multiplexados con varias funciones de periféricos y como entradas disponen de buffers Schmitt-Trigger

Algunos periféricos definen las líneas como entradas o salidas de manera independiente del estado de TRISC si hay definida una determinada funcionalidad para el pin

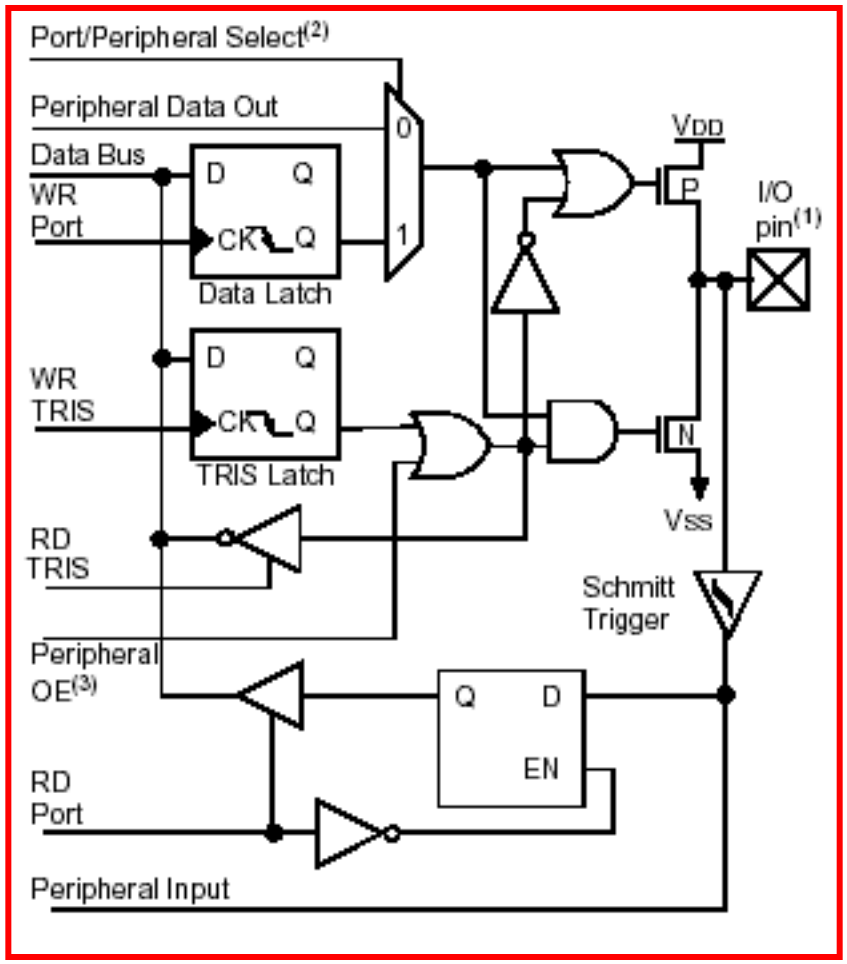


Diagrama de bloques de las líneas RC7 a RC5, RC2 a RC0

(en RC3 y RC4 se añade un multiplexor de entrada para selección de nivel de bus I2C normal ó SMBUS)

PORTD

Puerto de 8 líneas bidireccionales

Las entradas disponen de buffers Schmitt-Trigger

El PORTD se puede configurar como un puerto esclavo paralelo (PSP), controlable mediante las líneas del PORTE .

Se podría conectar de este modo a los buses de un sistema microprocesador

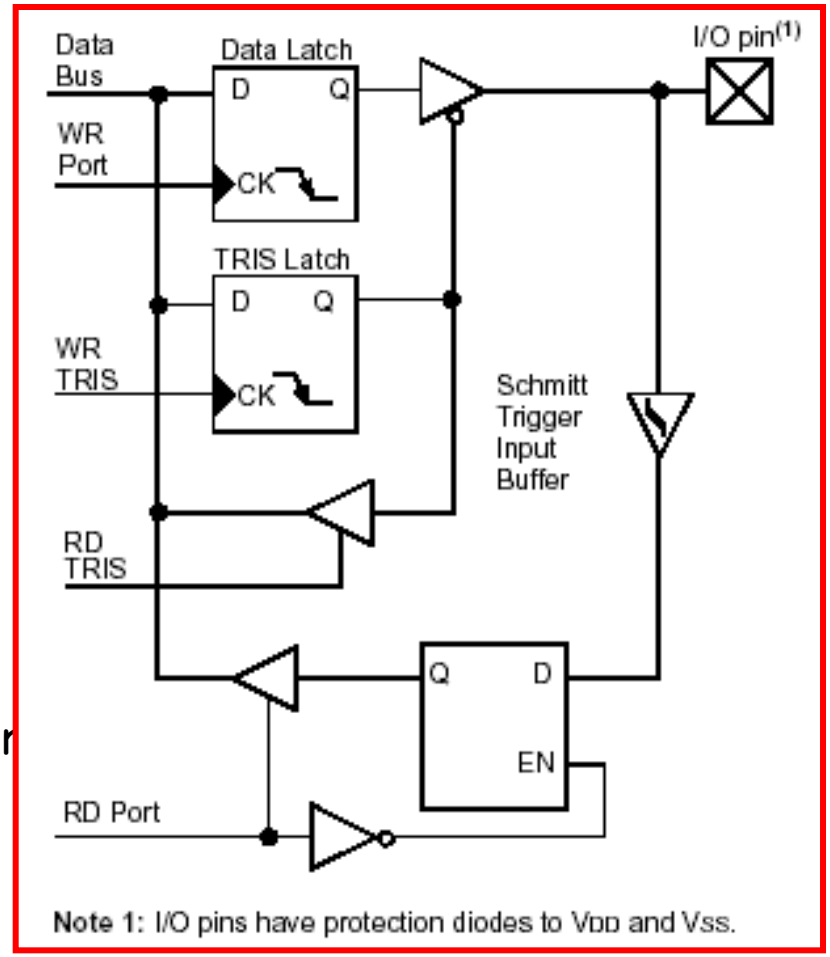


Diagrama de bloques de RD7 a RD0

PORTD como puerto esclavo paralelo (PSP)

Las líneas de control serían

$\overline{CS}$: Chip Select (RE2)

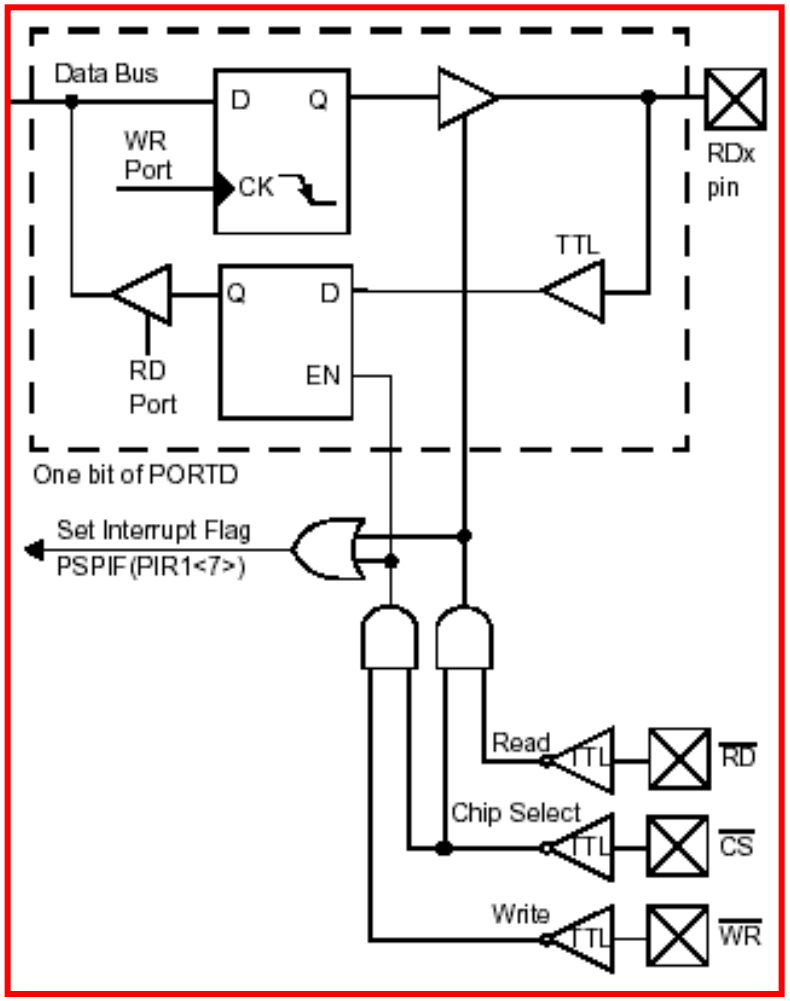
$\overline{RD}$: Lectura (RE0)

$\overline{WR}$: Escritura (RE1)

El acceso externo al PSP puede generar una interrupción

Existen funciones específicas de C para gestión del puerto esclavo paralelo:

setup_psp( )
psp_overflow( )
psp_input_full( )
psp_output_full( )

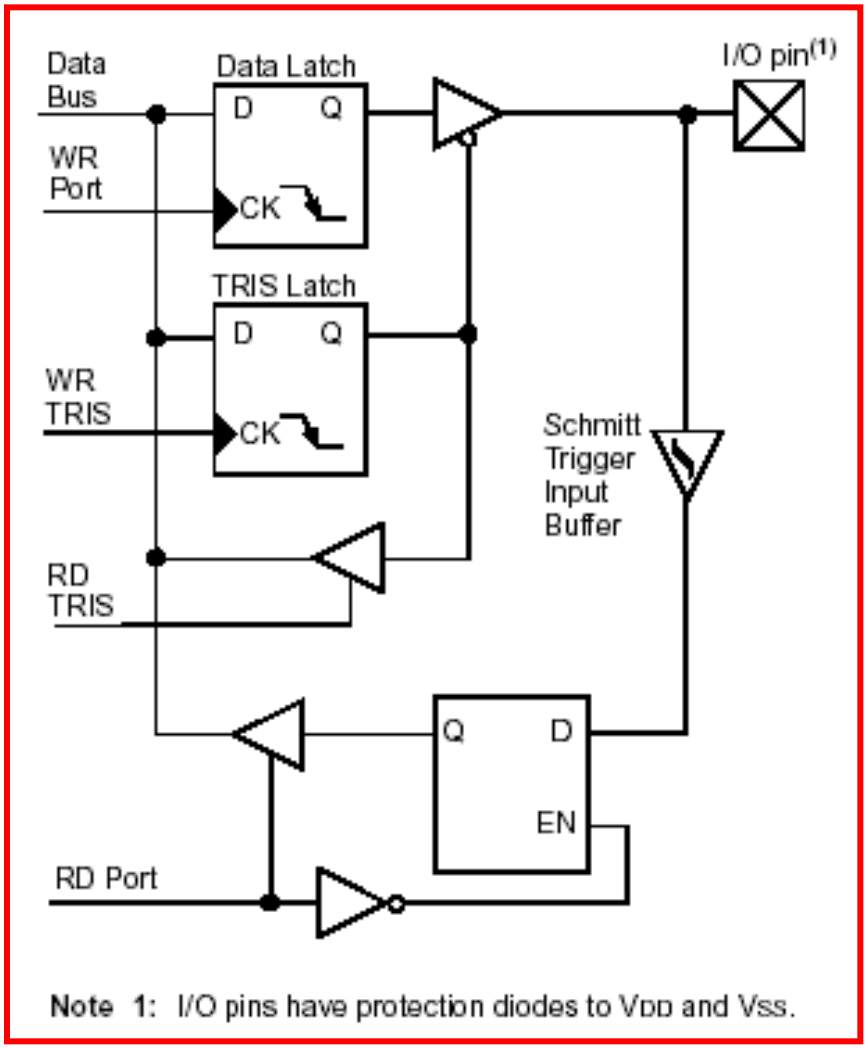


PORTE

Puerto bidireccional de 3 pines

Las líneas están multiplexadas con entradas analógicas del módulo de conversión A/D

Diagrama de bloques de RE2 a RE0



Gestión de los puertos de E/S

Básicamente existen dos opciones para la configuración (definición de las líneas de entrada y de salida) y el manejo (lectura o escritura) de los puertos de Entrada/Salida de los microcontroladores:

a).- Definiendo los registros como variables localizadas en RAM
Se definirían los puertos y los registros de dirección como variables de C y se colocarían en las posiciones reales de esos registros en la memoria RAM de datos

b).- Utilizando las funciones integradas específicas del compilador
Se definiría la dirección de los datos si fuera necesario y se asignarían salidas o explorarían entradas mediante las funciones relativas al manejo de todo el puerto o de bits particulares del mismo. En este caso la manera de asignar las salidas o leer las entradas ya no será tan directo como en el primer caso y el código que introduce el compilador puede variar en cuanto a tamaño y tiempo de ejecución ya que dependerá de la activación de ciertas directivas de preprocesado (#USE FAST_IO…)

Opción a).- Definir los puertos (PORTx) y los registros de dirección de datos (TRISx) como variables de tamaño byte y situarlas en las posiciones que les correspondan en el mapa de memoria del microcontrolador. La directiva #BYTE (o #LOCATE) son las más adecuadas ya que permiten realizar ambas funciones de manera simultánea:

```
#BYTE  TRISB = 0x86    //define la variable TRISB y la coloca en 0x86
#BYTE  PORTB = 0x06   //define la variable PORTB y la sitúa en 0x06
```

A partir de ahí esas variables controlan los puertos y se pueden utilizar en sentencias de asignación

p.e.:

```
TRISB = 0xF0; //4 bits altos entradas y 4 bajos salidas
...
PORTB = 0x0A;//asignación a los 4 bits de salida
numero = PORTB; //lectura del puerto B
...
if (PORTB & 0xF0)  PORTB|=0x0F;
```

Con este procedimiento también sería posible definir una estructura para un puerto definiendo diferentes campos para los distintos bits del mismo

p.e.:

```
struct  {
        short   enable;
        short   rs;
        short   rw;
        short   nada;
        int     medio : 4; } PUERTO_D;
```

Primero se define la estructura con los campos del byte y se declara la variable

```
#BYTE  PUERTO_D = 0x08;
```

Se sitúa esa variable en la posición 8

```
//se puede acceder posteriormente a los bits del puerto como campos

PUERTO_D.enable = 1;     PUERTO_D.medio =0xA;
```

También se puede declarar un bit de un puerto (o registro) con una variable mediante la directiva #BIT y hacer referencia a la variable para manipularla

```
#BIT    nombre = posición.bit
```

-> p.e.:

```
#BIT RA4 = 0x05,4
RA4=0;
```

Funciones de Manipulación de Bits para manejo de Puertos

Una vez definido un puerto como una variable tipo byte, sería posible manejar y leer los bits de un determinado puerto con algunas de estas funciones:

bit_clear(variable,bit)	pone a 0 el bit especificado de variable
bit_set(variable,bit)	pone a 1 el bit indicado de la variable señalada
bit_test(variable,bit)	muestrea el bit especificado de la variable
swap(variable_byte)	intercambia los nibbles (medio byte) de la variable indicada

p.e.

```
#BYTE PUERTOB = 0x06 //Declaración previa del PUERTOB

bit_set(PUERTOB,5);

if(!bit_test(PUERTOB,2))  bit_clear(PUERTOB,2);
```

Opción b).- Gestión de E/S con Funciones Integradas en el Compilador

output_low(pin)	Sitúa en estado bajo el pin especificado
output_high(pin)	Coloca en estado alto el pin indicado
output_bit(pin, valor)	Pone el pin señalado con el valor (0 ó 1) que se indique
output_float(pin)	Define el pin como de entrada quedando a tensión flotante, simulando una salida de drenador abierto
output_A(valor) output_B(valor) output_C(valor) output_D(valor) output_E(valor)	Saca el valor indicado al correspondiente puerto
port_b_pullups(TRUE o FALSE)	Activa las resistencias de pull-up de las entradas del PORTB

set_tris_A(valor) set_tris_B(valor) set_tris_C(valor) set_tris_D(valor) set_tris_E(valor)	Carga el registro de dirección de datos con el valor indicado como parámetro
input(pin)	Devuelve el estado del pin señalado
input_A( ) input_B( ) input_C( ) input_D( ) input_E( )	Devuelve el valor presente en el puerto correspondiente

La generación de código para las funciones output_xxx( ) e input_xxx( ) depende de la última directiva del tipo #USE *_IO que esté activa y su activación es muy importante porque puede condicionar el funcionamiento o no de un programa

Especificación de pin

Cuando es necesario especificar un pin en alguna de las funciones anteriores, se emplean unos identificadores para los pines del puerto del microcontrolador.

Están definidos en el fichero .h del dispositivo (p.e. 16f877.h) con directivas #DEFINE que asignan identificadores del tipo PIN_Xn (X sería el puerto y n el número del pin: PIN_A0) a direcciones de bits.

La dirección de un bit se obtiene multiplicando por 8 la dirección del puerto y sumándole el número del pin (PIN_B3 -> 6*8+3 = 51)

Ejemplo en fichero .h:

```
#define PIN_A0  40
#define PIN_A1  41
#define PIN_A2  42
#define PIN_A3  43
#define PIN_A4  44
#define PIN_A5  45
```

Directivas #USE *_IO

Existen 3 directivas que condicionan el modo en que se genera código para las funciones integradas relativas a los puertos de entrada/salida

#USE FAST_IO(PUERTO) PUERTO: desde A hasta E

Cada vez que se emplee una función del tipo output…( ) ó input…( ), se saca ese valor directamente al puerto o al bit si se trata de output o se lee el puerto si es input, pero no se modifican previamente el registro de dirección de datos para todo el puerto o para el bit . Debemos de asegurarnos que los registros de dirección de datos estén cargados adecuadamente antes de llamar a las funciones

p.e. #USE FAST_IO(B)

(se puede ver el efecto en un fichero .lst resultado de la compilación)

#USE STANDARD_IO(PUERTO) PUERTO: desde A hasta E

En este caso, cada vez que aparece una función output…, se inserta código previo para forzar a que el bit particular o el puerto completo sean de salida (mediante la carga del TRIS correspondiente). Si se trata de una función input… se carga código para definir bit o puerto completo como entrada. Este es el método activo por defecto

p.e.: # USE STANDARD_IO(D)

#USE FIXED_IO(PORT_OUTPUTS=PIN, PIN?) PORT: A hasta E

Se genera código relativo a la dirección de los datos de manera previa cada vez que aparece una función integrada del tipo input…( ) ó output…( ), pero los pines se configuran de acuerdo con la información que acompaña a la directiva (están definidos los pines de salida) y no dependiendo de que la operación sea de entrada o de salida como sucede con #USE STANDARD_IO(PUERTO)

p.e.: # USE FIXED_IO(B_OUTPUTS=PIN_B2, PIN_B3)

Funciones integradas en el compilador para retardos

Este tipo de funciones pueden ser útiles para establecer ciertos retardos o temporizaciones a la hora de explorar o manejar puertos de entrada/salida. Todas ellas las generan por software (no usan temporizadores):

- delay_cycles(contador) siendo contador un valor entre 1 y 255

Introduce código para generar un retardo de contador ciclos de instrucción

- delay_ms(tiempo) tiempo es una variable de 0 a 255 o bien una constante comprendida entre 0 y 65535

Genera un retardo software del valor indicado en milisegundos, lo calcula a partir del oscilador definido con la directiva #USE DELAY(clock=…)

- delay_us(tiempo) tiempo: var. de 0 a 255 o constante de 0 a 65535

Genera un retardo del tiempo especificado en microsegundos

A continuación:

Algunos Ejemplos para manejar E/S