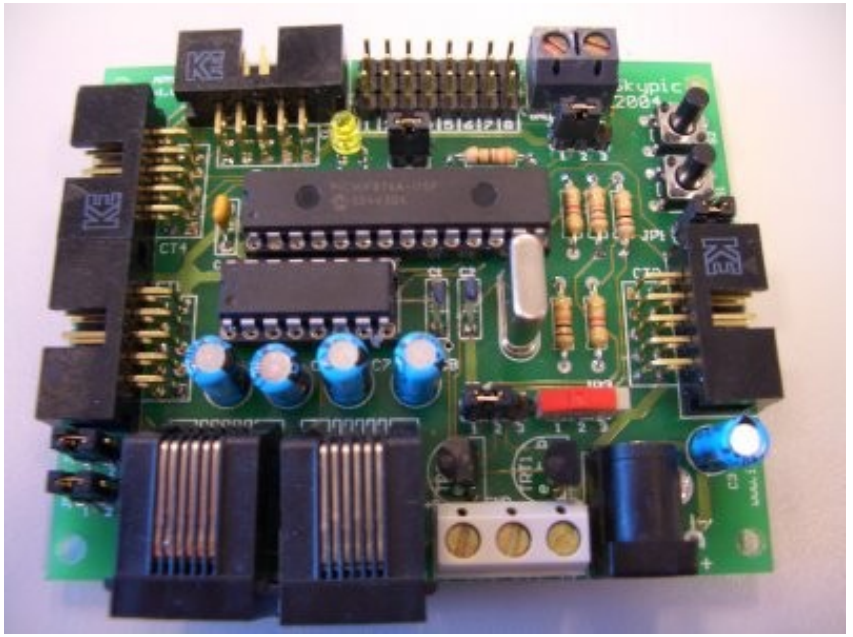
PIC 16F87X



Juan González

Escuela Politécnica Superior
Universidad Autónoma de Madrid

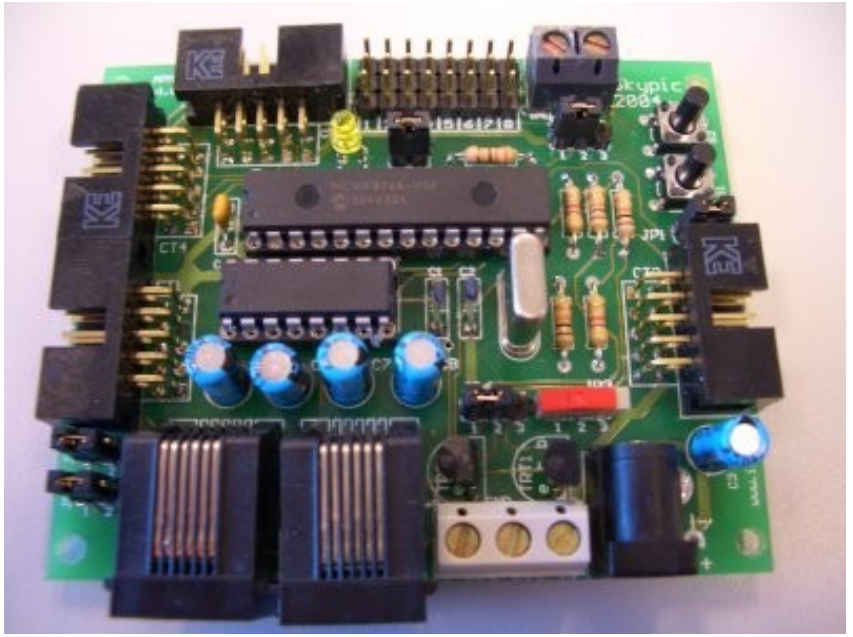
Andrés Prieto-Moreno

Flir Networked Systems

Ricardo Gómez

Flir Networked Systems

PIC 16F87X

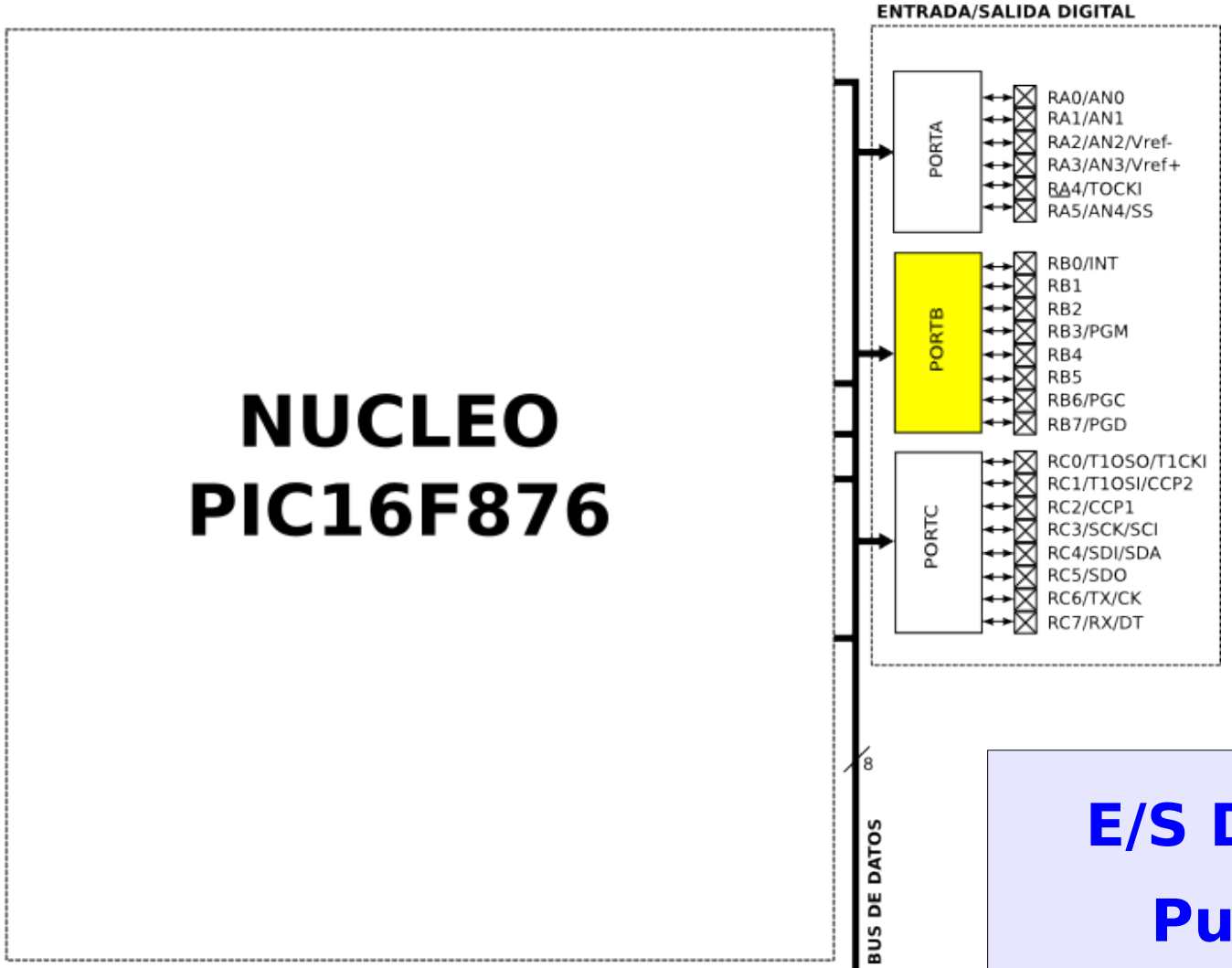


MÓDULO 2:

Puertos de E/S digitales

Introducción al lenguaje C (I)

Programación de periféricos

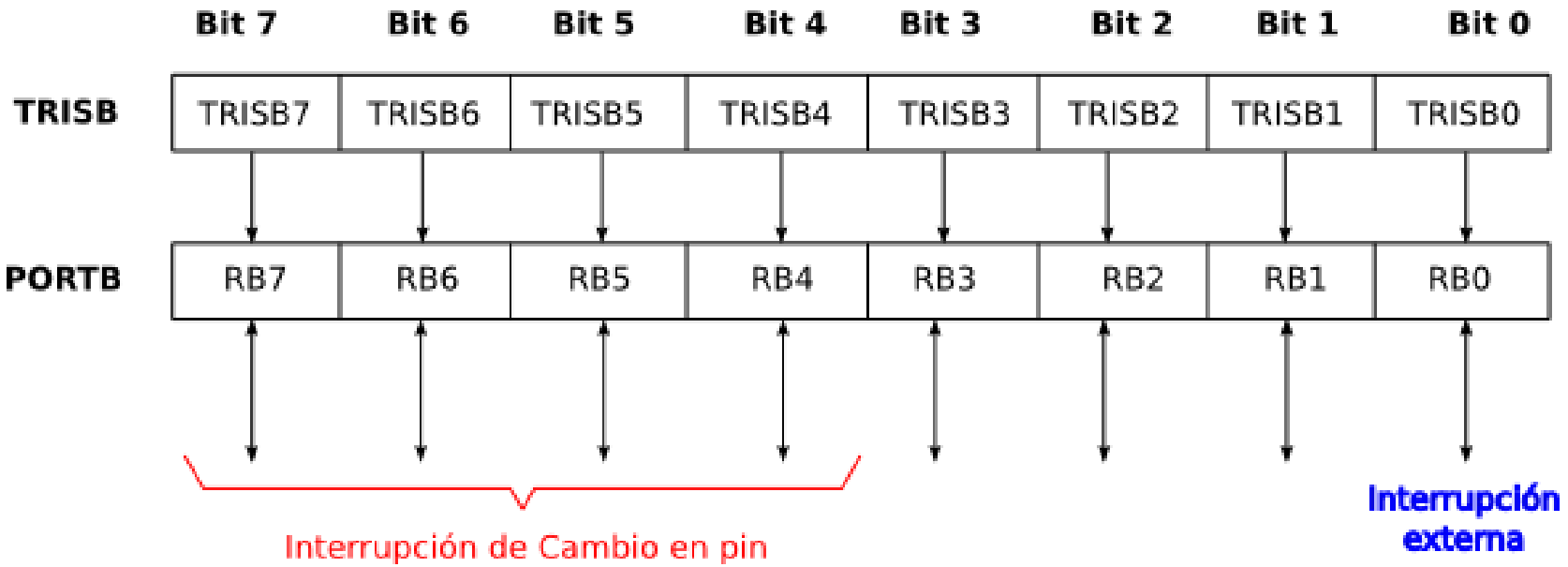


E/S Digital:

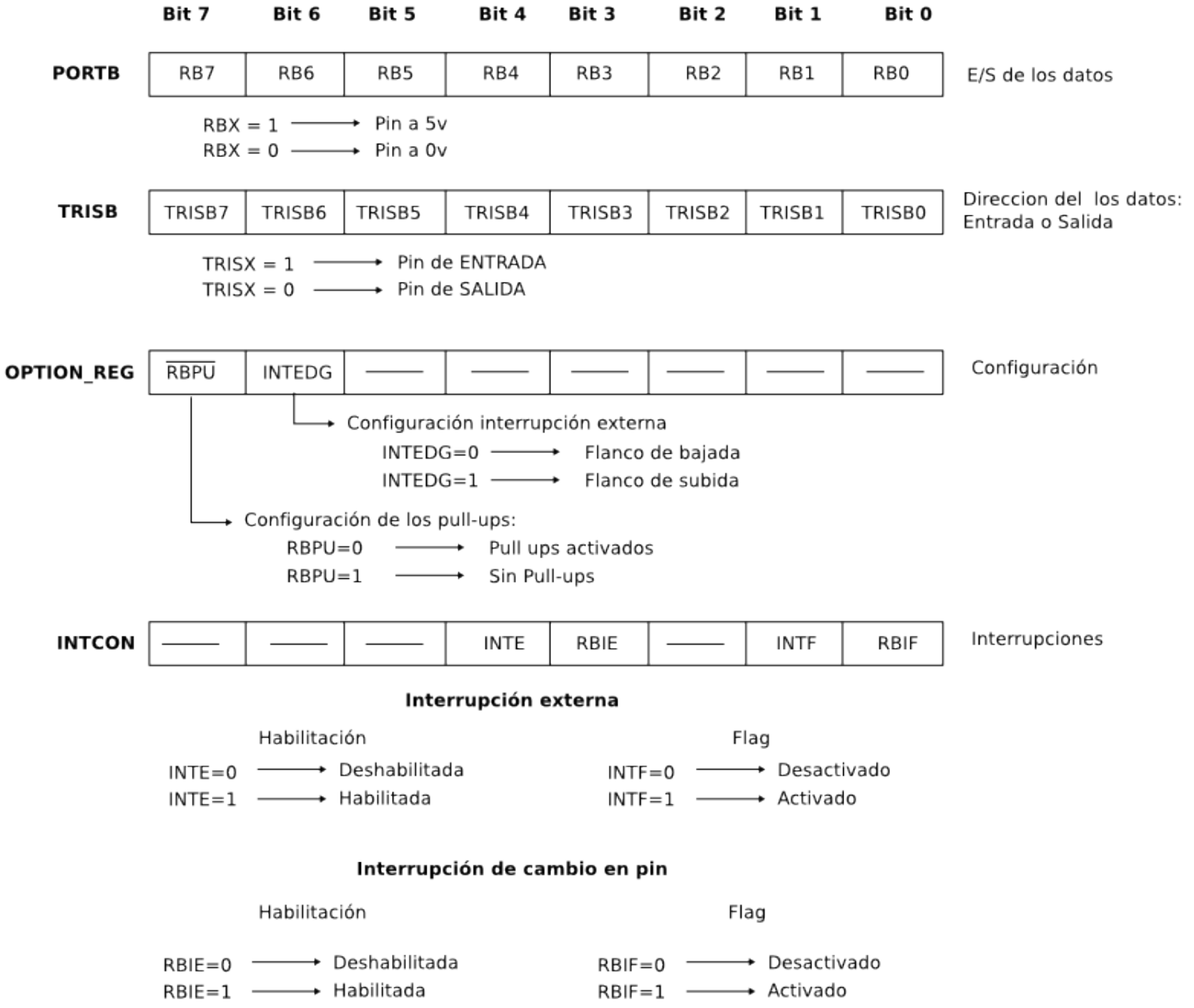
Puerto B

Entrada/salida digital: Puerto B

- **8 pines** independientes configurables para E/S
- Resistencias de pull-up internas (opcionales)
- **Interrupción** cuando cambia el estado de los pines **RB7 – RB4**
- **Interrupción externa** configurable en flanco subida o bajada



REGISTROS DEL PUERTO B



Ejemplo: Activar un pin de salida (I)

Ledon.c


```
#include <pic16f876a.h>

void main()
{
  TRISB1 = 0;
  RB1 = 1;
  while(1);
}
```

Configurar pin RB1 para salida

Activar pin RB1

Bucle infinito

Ejemplo: Activar un pin de salida (II)

Ledon2.c

```
#include <pic16f876a.h>

#define LED RB1
#define CONFIGURAR_LED  TRISB1=0
#define ON  1
#define OFF 0

void main(void)
{
  CONFIGURAR_LED;
  LED=ON;
  while(1);
}
```

- Con #define se mejora la legibilidad del programa
- No se genera código adicional
- Independencia entre el programa y los valores de los bits
- Mejora de la portabilidad
- Mejora la documentación

Ejemplo: Puerto de salida de 8 bits

salida8.c

```
#include <pic16f876a.h>

void main(void)
{
  TRISB=0x00;
  PORTB = 0xAA;
  while(1);
}
```

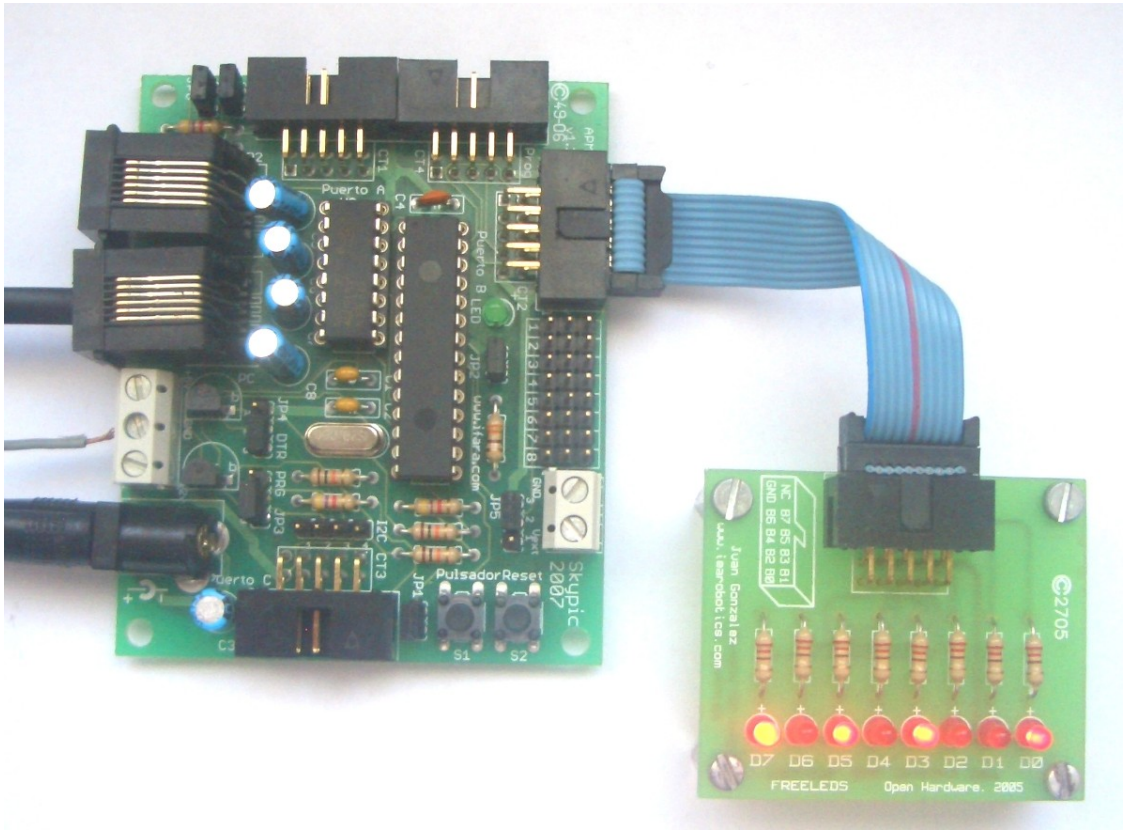
Configurar todos los pines para salida

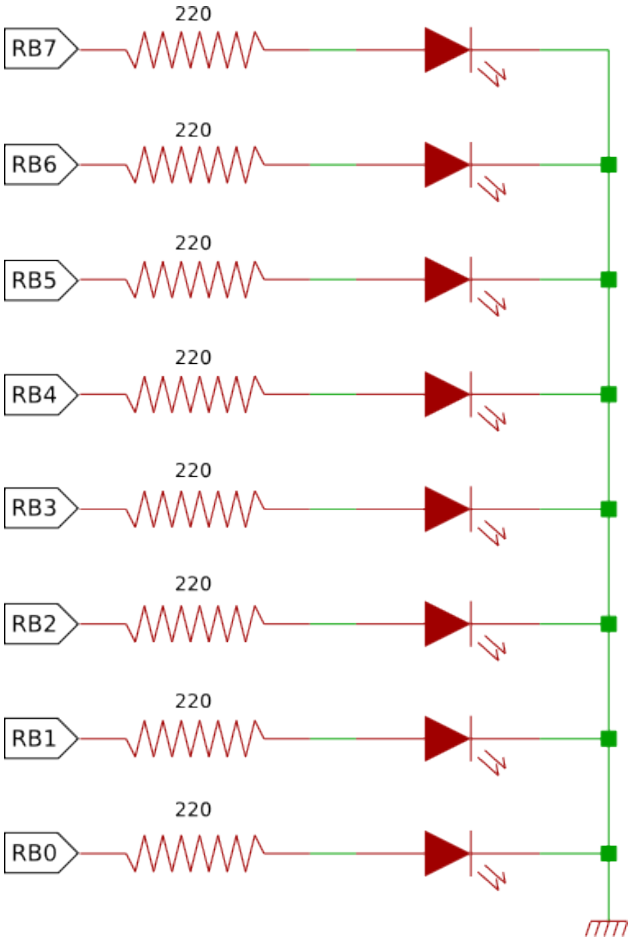
Enviar el valor 0xAA (hexadecimal) a los pines del puerto B

0x Indica que se trata de un número hexadecimal

0xAA = 10101010 en binario

Ejemplo: Puerto de salida de 8 bits





- **Tarjeta Freeleds** (Hardware libre)
  - http://www.iearobotics.com/personal/juan/proyectos/freeleds
- El valor enviado se muestra en binario en los leds
- ¡¡ Muy útil para depurar !!

Ejercicio “hola mundo”

- Modificar el programa **salidas8.c** para que se envíe el siguiente valor binario por el puerto B:

1 1 0 0 1 0 0 1

- Modificar el programa
- Compilarlo
- Descargarlo en la Skypic

Variables en C

variables1.c

```
#include <pic16f876a.h>
#define PUERTOB_SALIDA TRISB=0x00

void main(void)
{
  unsigned char i;
  unsigned char j;

  PUERTOB_SALIDA;
  i=1;
  j=i+2;
  PORTB = j;
}
```

- Tipos de datos soportados por el SDCC:
  - Enteros de 8 bits: ***char***, ***unsigned char***
  - Enteros de 16 bits: ***int***, ***unsigned int***
  - Enteros de 32 bits: ***long, unsigned long***
  - Flotantes (32 bits, IEEE): ***float***

Declaración de las variables i,j

Valores entre 0 y 255 (unsigned char)

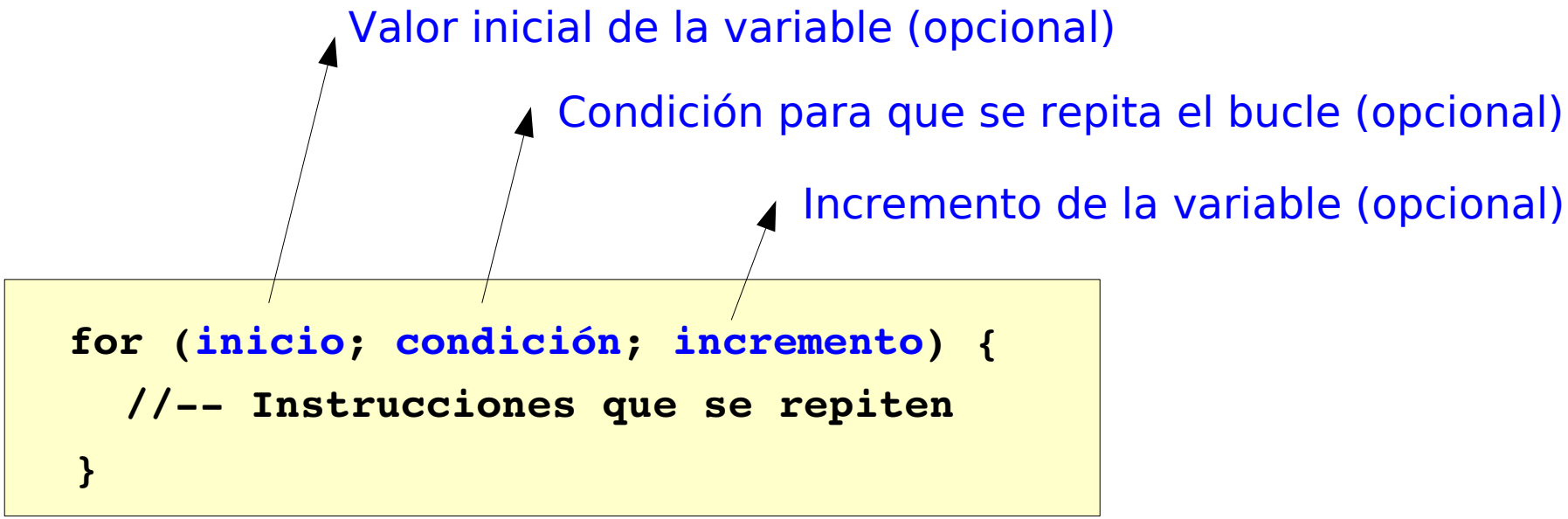
Operaciones con las variables

Sacar la variable j por el puerto B

Principio de Economía de recursos: Siempre que sea posible, hay que utilizar variables de los tipos ***char*** o ***unsigned char***

Bucle for

- **Sintáxis**:



- **Ejemplo:**

```
for (i=0; i<50; i++) {
   ...
}
```

Bucle que se repite 50 veces

- **Ejemplo:**

```
for (; ;) {
   ...
}
```

Bucle infinito

Bucle for (II)

contador1.c

```
#include <pic16f876a.h>

#define PUERTOB_SALIDA TRISB=0x00

void main(void)
{
  unsigned char i;

  PUERTOB_SALIDA;

  for (i=0; i<=128; i++) {
    PORTB=i;
  }

  while(1);
}
```

Ejemplo: Contador de 0 a 128 que se muestra por los leds...

Declaración de la variable que hace de contador

Este bucle se repite 129 veces. La i toma valores desde 0 hasta 128

Sacar la variable i por el puerto B

Ejecutar el programa...

¿Qué ocurre? ¿Qué es lo que se ve en los leds?

Bucle for (III)

¿Qué ocurre? ¿Qué es lo que se ve en los leds?

Respuesta:

Por los leds se verá el siguiente valor:

1 0 0 0 0 0 0 0

Explicación:

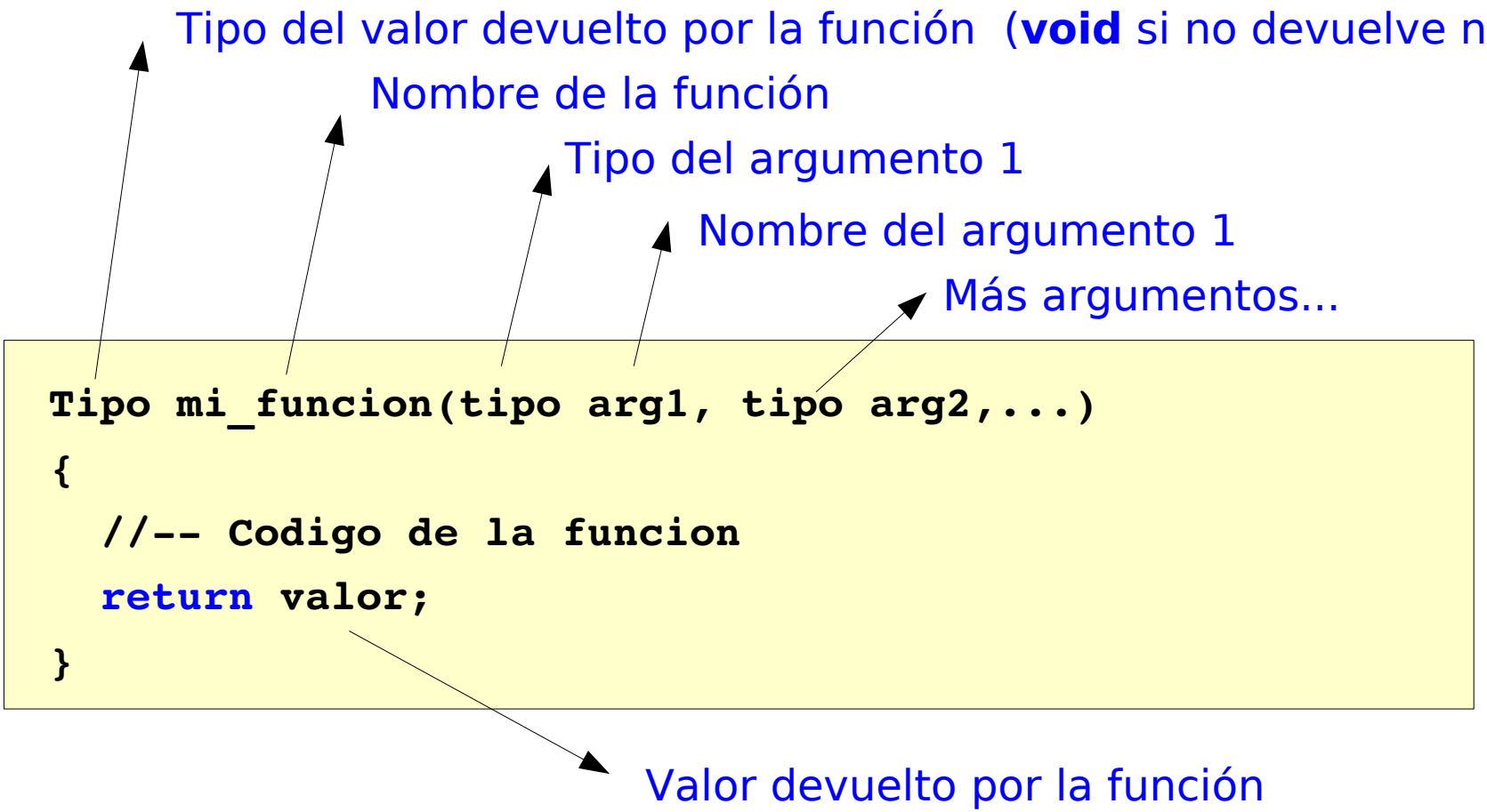
- El bucle se ejecuta tan rápido que no da tiempo a ver los valores intermedios que se envían a los leds
- Sólo se ve el valor final 1000000 que se corresponde con el número decimal 128

Solución:

- Hay que hacerse una función de pausa ;-)

Funciones en C

- **Sintáxis**:



Funciones en C (II)

- **Ejemplo 1**:

```
void configurar()
{
...
}
```

Función que no devuelve nada y no tiene argumentos

- **Ejemplo 2**:

```
void pausa(unsigned char tiempo)
{
...
}
```

Función que no devuelve nada y tiene un argumento del tipo unsigned char

- **Ejemplo 3**:

```
int operar(int op1, int op2)
{
...
}
```

Función que devuelve un valor entero y tiene dos argumentos también enteros (int)

- **Invocando las funciones:**

```
void main()
{
  int a,i;
  i=5;
  configurar();
  pausa(10);
  a = operar(5,i);
}
```

Función de pausa

- Vamos a hacer una función de pausa, que consuma tiempo de la CPU
- De momento no estamos interesados en la exactitud ni en el valor de nuestras unidades de tiempo. Cuando programemos los **Timers** conseguiremos pausas de los tiempos exactos que queramos
- Usaremos bucles anidados

```
void pausa(unsigned char tiempo)
{
  unsigned char i;
  unsigned int j;
  int temp=0;

  for (i=0; i<tiempo; i++) {
    for (j=0; j<0x8000; j++) {
      temp=temp+1;
    }
  }
}
```

Variables para los bucles

Variable temporal

Cálculo “inútil” para perder tiempo

La función no devuelve nada

Programa contador. Versión 2

contador2.c

```
#include <pic16f876a.h>

#define PUERTOB_SALIDA TRISB=0x00

void pausa(unsigned char tiempo)
{
  unsigned char i;
  unsigned int j;
  int temp=0;

  for (i=0; i<tiempo; i++) {
    for (j=0; j<0x8000; j++) {
      temp=temp+1;
    }
  }
}

void main(void)
{
  unsigned char i;

  PUERTOB_SALIDA;

  for (i=0; i<=128; i++) {
    PORTB=i;
    pausa(1);
  }

  while(1);
}
```

- Esto ya es otra cosa :-)
- Ahora se puede ver cómo el contador se incrementa

Ejercicios:

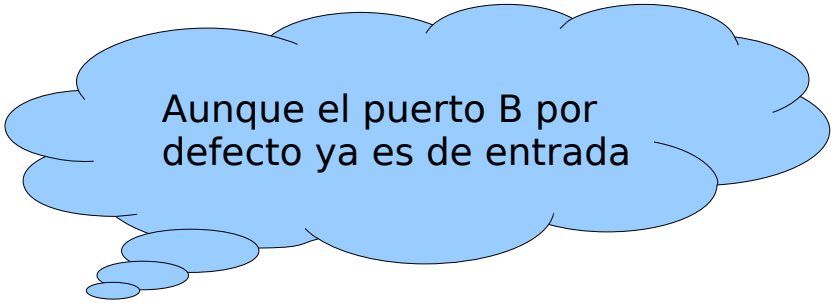
- Modificar el programa para que el contador vaya más lento y más rápido
- Modificar la rutina de pausa para que admita como segundo argumento el número de repeticiones del bucle interno
- Hacer el programa **ledp.c**, que haga parpadear el led

Invocar la función de pausa

Ejemplo: Lectura de pines

pulsador-led.c

```
#include <pic16f876a.h>

void main(void)
{
 TRISB0=1;
 TRISB1=0;

 while(1) {
  if (RB0==1) {
    RB1=0;
  }
 else {
    RB1=1;
 }
 }
}
```



Configurar RB0 para entrada (Pulsador)

Configurar RB1 para salida (led)

Bucle principal

== Operador de igualdad

= Operador de asignación

Este programa muestra por RB1 lo contrario de lo que recibe por RB0
Cada vez que se apriete el pulsador, se encenderá el led

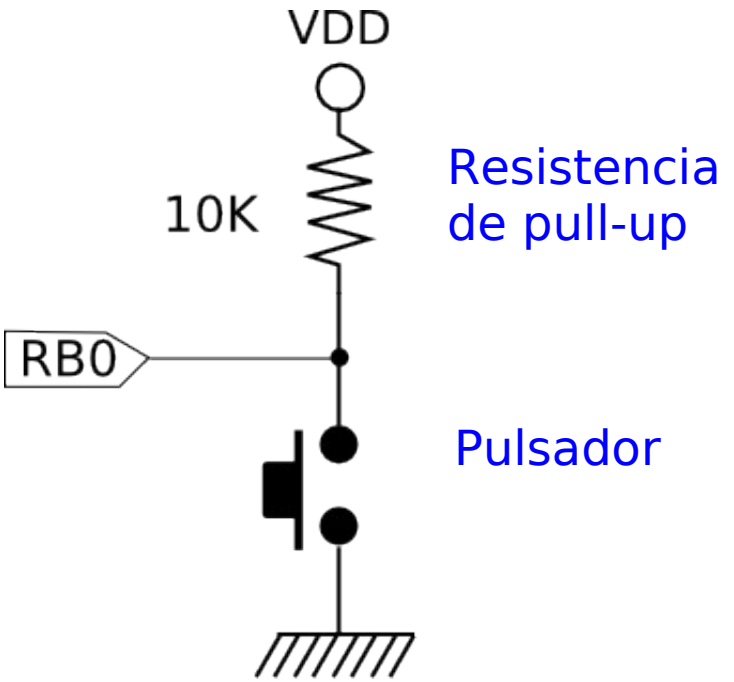
```
#include <pic16f876a.h>

#define LED       RB1
#define PULSADOR  RB0
#define ON        1
#define OFF       0
#define APRETADO  0
#define CONFIGURAR_LED       TRISB0=1
#define CONFIGURAR_PULSADOR  TRISB1=0

void main(void)
{
  CONFIGURAR_LED;
  CONFIGURAR_PULSADOR;
  while(1) {
    if (PULSADOR==APRETADO) {
      LED=ON;
    }
    else {
      LED=OFF;
    }
  }
}
```

pulsador-led2.c

Ejemplo: Lectura de pines (II)

- Mejora de la legibilidad y documentación con **#define**
- Ahora cualquiera entiende lo que hace el programa con sólo mirar el *main()*

Hardware:



Entrada RB0. Lectura por espera activa

contador-int1.c

```
#include <pic16f876a.h>

void main(void)
{
  TRISB=0x01;
  INTEDG=0;

  while(1) {
    INTF=0;
    while(INTF==0);
    RB1=RB1^1;
  }
}
```

Ejemplo: Cambiar el estado del led cada vez que llega un flanco de bajada por RB0

Configurar RB0 para entrada (Pulsador) y resto como salidas

Configurar RB0 para funcionar por flanco de bajada

Inicializar flag

Esperar a que se produzca el flanco. Cuando llega INTF se pone a '1'

Cambiar el LED de estado
El Operador ^ es un xor
También se podría usar: RB1^=1;

Opcional: Añadir una pausa para eliminar los rebotes que pudiesen venir del pulsador

Desconectar la tarjeta FREELEDS!

Entrada RB0. Lectura por Interrupciones

contador-int2.c

```
#include <pic16f876a.h>

void isr() interrupt 0
{
  INTF=0;
  RB1^=1;
}

void main(void)
{
  TRISB=0x01;
  INTEDG=0;
  INTE=1;
  GIE=1;

  while(1) {
  }
}
```

Rutina de atención a las interrupciones. Cada vez que llegue un flanco de bajada en RB0 se ejecuta esta rutina

Permitir las interrupciones por RB0

Activar las interrupciones del PIC!

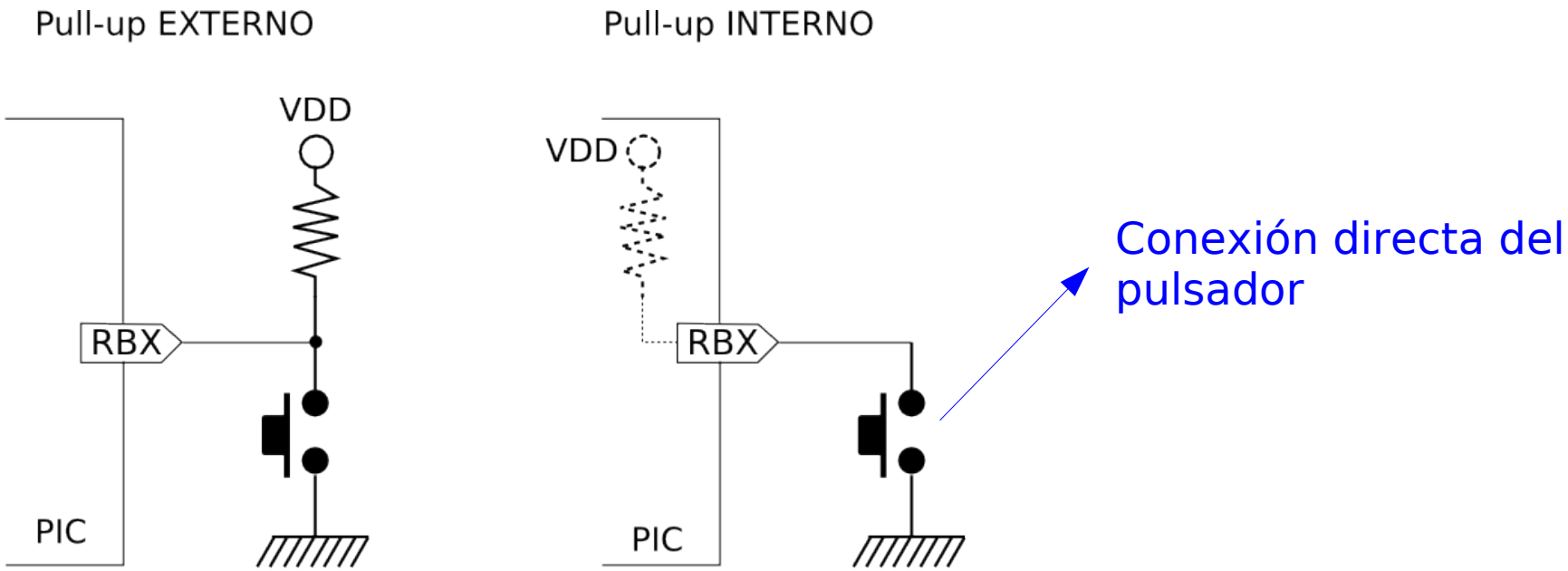
El programa principal “no hace nada”. Todo se hace mediante interrupciones. El pic podría hacer otras cosas mientras se cuentan los flancos

Desconectar la tarjeta FREELEDS!

Activación pull-ups internos

• Los pines del puerto B se pueden configurar como entradas normales o bien con pull-ups internos.

• Usar el pull-up interno es muy cómodo para poder conectar directamente pulsadores sin tener que añadir resistencias



Activación pull-ups internos (II)

pull-ups.c

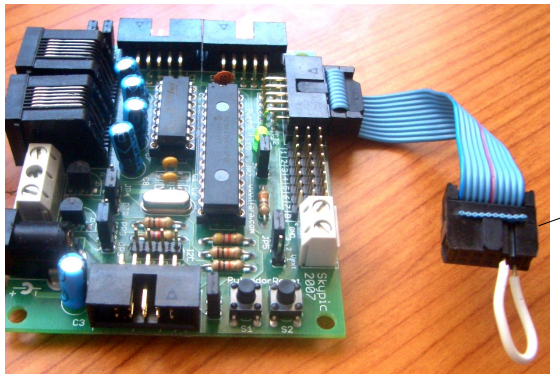
```
#include <pic16f876a.h>

#define LED RB1

void main(void)
{
  TRISB1=0;
  NOT_RBPU=0;

  while(1) {
    LED=RB7;
  }
}
```

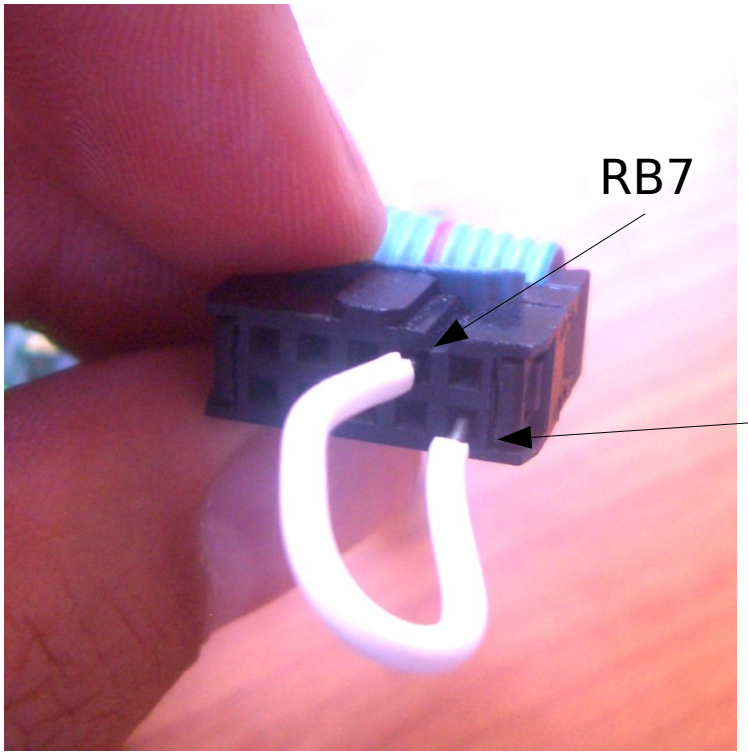
Activación de las resistencias del PULL-UP

Sacar por el LED el estado del pin RB7

Usamos **un cable** para unir directamente RB7 con GND. Al unirlo, el LED se apaga. Al quitarlo, se enciende.

Para probar este ejemplo:





Entradas RB7-RB4. Detección de flancos (I)

• Cuando hay un **cambio en los pines del RB7 al RB4**, se activa el **flag RBIF** y opcionalmente se generará una interrupción. Esto permite detectar todos los flancos que se reciben (de subida y bajada)

• Para limpiar este flag, hay que escribir/leer en el puerto y poner a cero RBIF

Programa de ejemplo: Detectar flancos por RB7

• La detección de los flancos se hace por espera activa

• Se cambia el estado del LED cada vez que viene un flanco

Entradas RB7-RB4. Detección de flancos (I)

flanco1.c

```
#include <pic16f876a.h>
#define LED RB1

volatile unsigned char temp;

void main(void)
{
  TRISB=0x80;
  NOT_RBPU=0;

  while(1) {
    temp=PORTB;
    RBIF=0;
    while(RBIF==0);
    LED^=1;
  }
}
```

Pin RB7 configurado como entrada, resto salidas

Activar pull-ups. Lo hacemos porque usaremos un cable como pulsador ;-)

Limpiar el flag: Primero se hace una lectura del puerto B y luego ponemos el flag a 0

Esperar a que ocurra un cambio en RB7

Cambiar de estado el led

La palabra clave volatile le indica al optimizador del compilador que variable temp **NO** se elimine, a pesar de que no se use.

Más sobre el tipo volatile

Examinar este fragmento de código:

```
void main(void)
{
  int t;

  ...
  t=0xAA;
  t=0x55;
  ...
}
```

- Si pensamos en este programa como un algoritmo, no tiene ningún sentido. Asignamos un valor a **t** y justo a continuación **lo machacamos**, sin haberlo usado
- El compilador tomará la decisión de **eliminar la sentencia t=0xAA** por lo que el código sería equivalente a este otro:

```
void main(void)
{
  int t;

  ...
  t=0x55;
  ...
}
```

- Pero si la variable t se define como volatile, el optimizador no simplificará nada.
- Este tipo se usa para **acceder a variables cuya lectura o escritura influyen directamente en el hardware**, como por ejemplo escribir información en un puerto, cambiar el estado de los flags, etc.

Entradas RB7-RB3. Detección de flancos (II)

flanco2.c

```
#include <pic16f876a.h>
#define LED RB1

volatile unsigned char temp;

void isr() interrupt 0
{
  temp=PORTB;
  RBIF=0;
  LED^=1;
}

void main(void)
{
  TRISB=0x80;
  NOT_RBPU=0;
  RBIE=1;
  GIE=1;

  while(1) {
  }
}
```

Detección mediante interrupciones

Rutina de atención a la interrupción

Limpiar el flag de interrupción

Cambiar el led de estado

Pin RB7 configurado como entrada, resto salidas

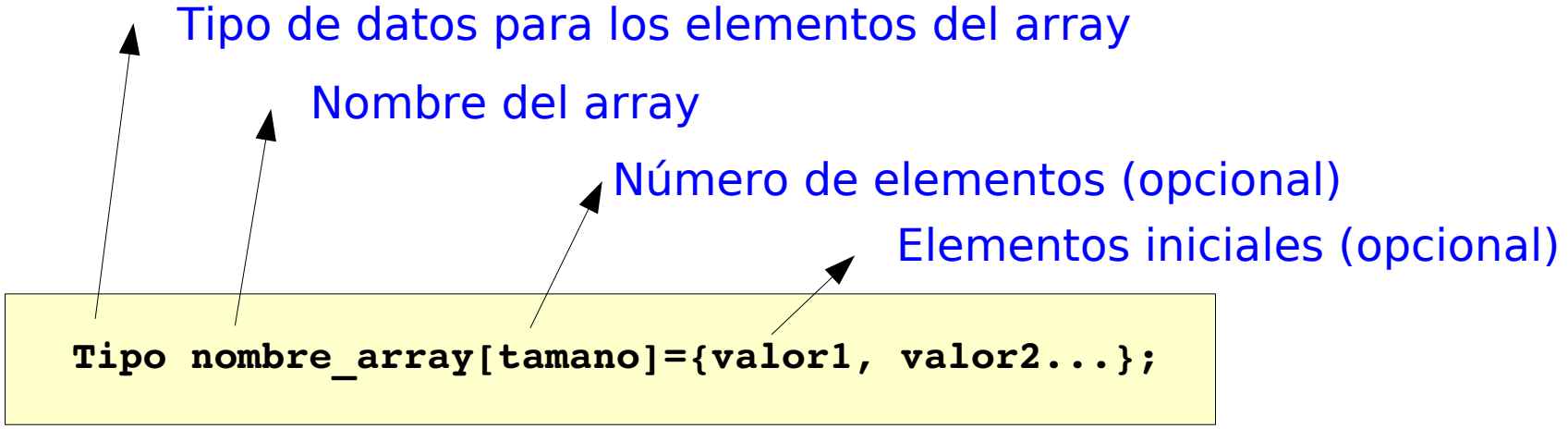
Activar pull-ups. Lo hacemos porque usaremos un cable como pulsador ;-)

Activar interrupción de cambio

Activar interrupciones del PIC

Arrays en C

- **Sintáxis**:



- **Ejemplo:**

```
int prueba[50];
```

Array de 50 enteros

- **Ejemplo:**

```
char tabla[]={40,25,22,35};
```

Tabla de 4 bytes. El número de elementos lo calcula el compilador a partir de los valores iniciales

Arrays en C (II)

- **Acceso a los arrays:**

```
int prueba[50];
void main()
{
  int ultimo;
  ..
  prueba[0]=20;
  ..
  ultimo = prueba[49];
  ..
}
```

Lectura del primer elemento. ¡En C se empieza a contar desde 0!

Lectura del último elemento. Como se empieza a contar desde 0, el último elemento es el de índice 49

- **Recorrer una tabla:**

```
Unsigned char tabla[]={30,50,60}
void main()
{
  unsigned char i;
  unsigned char elemento;
  for (i=0; i<2; i++) {
    ..
    elemento=tabla[i];
    ..
  }
}
```

Juego de luces en los leds (I)

luces.c

```
#include <pic16f876a.h>

unsigned char tabla[]={0x55,0xAA};
void pausa(unsigned char tiempo)
{...}

void main(void)
{
  unsigned char i;

  TRISB=0x00;
  while(1) {
    for (i=0; i<1; i++) {
      PORTB=tabla[i];
      pausa(2);
    }
  }
}
```

Tabla con los valores a sacar por los leds

Función de pausa usada en los ejemplos anteriores

Variable índice para recorrer la tabla

Configurar puerto B pasa salida

Recorrer la tabla. Solo hay dos elementos: tabla[0] y tabla[1]

Sacar valores por los leds

Realizar una pausa

Juego de luces en los leds (II)

- Vamos a mejorar un poco el programa anterior
- ***¿Cómo hacemos para conocer el número de elementos y poder usarlo para recorrer la tabla?***

Operador Sizeof

- Devuelve el tamaño en bytes de cualquier tabla, variable o tipo. Se calcula en tiempo de compilación, por lo que no genera código en ensamblador

- **Ejemplo:**
  - Dado el siguiente array: `unsigned char tabla[]={...};`
  - El número de elementos se calcula así:

```
unsigned char size = sizeof(tabla)/sizeof(unsigned char);
```

Juego de luces en los leds (III)

luces2.c

```
#include <pic16f876a.h>

unsigned char tabla[]={0x55,0xAA};
unsigned char size =
     sizeof(tabla)/sizeof(unsigned char);

void pausa(unsigned char tiempo)
{...}

void main(void)
{
  unsigned char i;
  TRISB=0x00;

  while(1) {
    for (i=0; i<size; i++) {
      PORTB=tabla[i];
      pausa(2);
    }
  }
}
```

Obtener el numero de elementos de la tabla

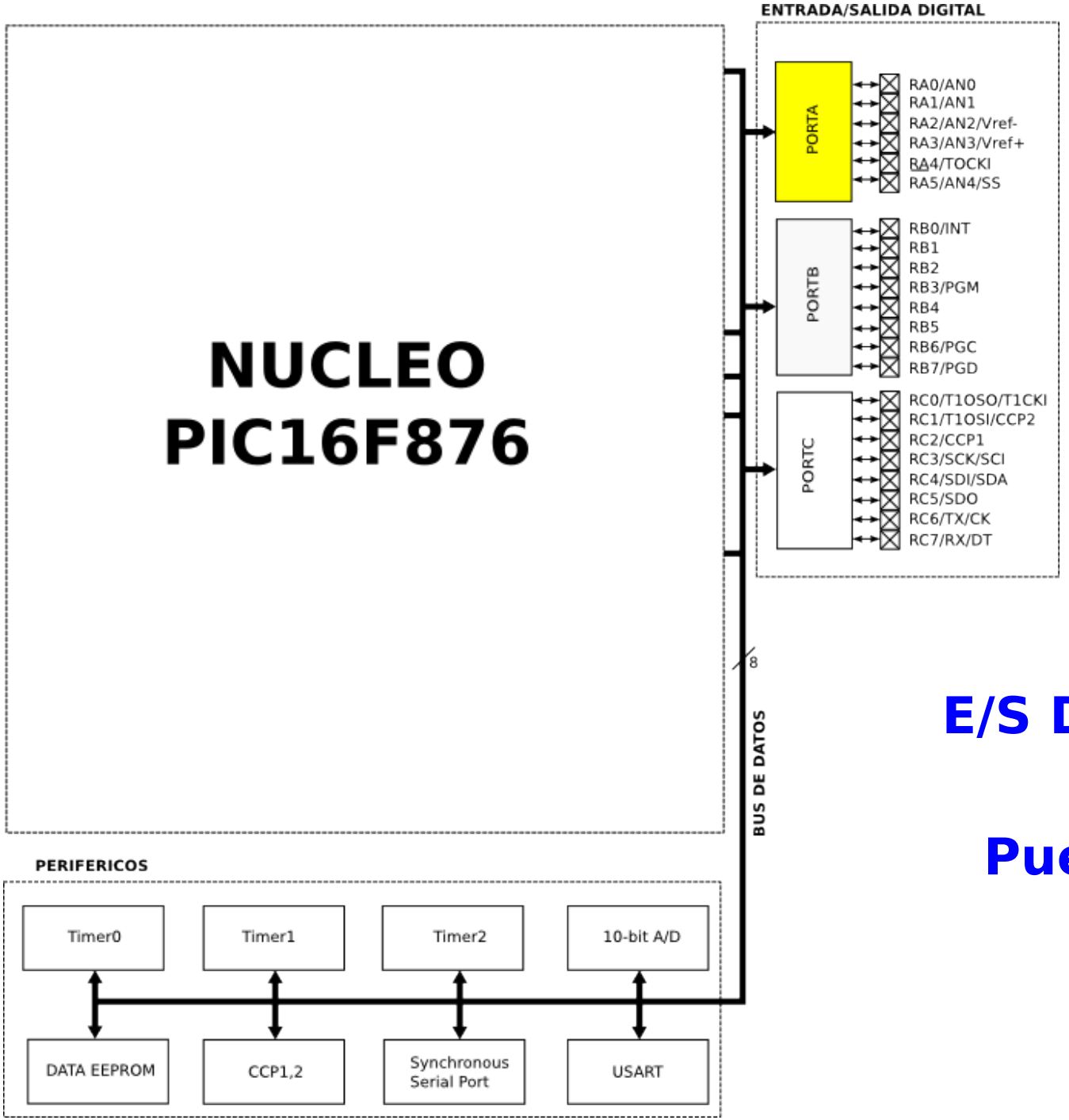
Este código funciona para tablas de cualquier longitud

• Ahora se pueden añadir nuevos valores a la secuencia de movimiento simplemente añadiéndolos a la tabla. **No hay que tocar el resto del código**

Ejercicio:

• Crear nuevas secuencias y cambiar sus velocidades (¿El coche fantástico?) :-)

Programación de periféricos

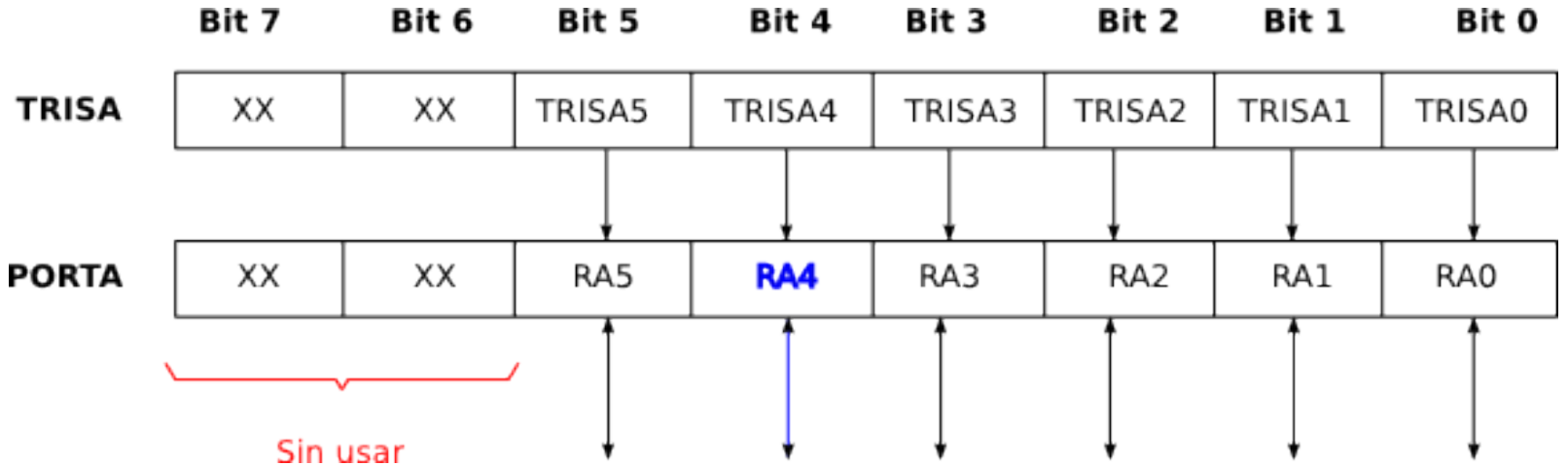


E/S Digital:

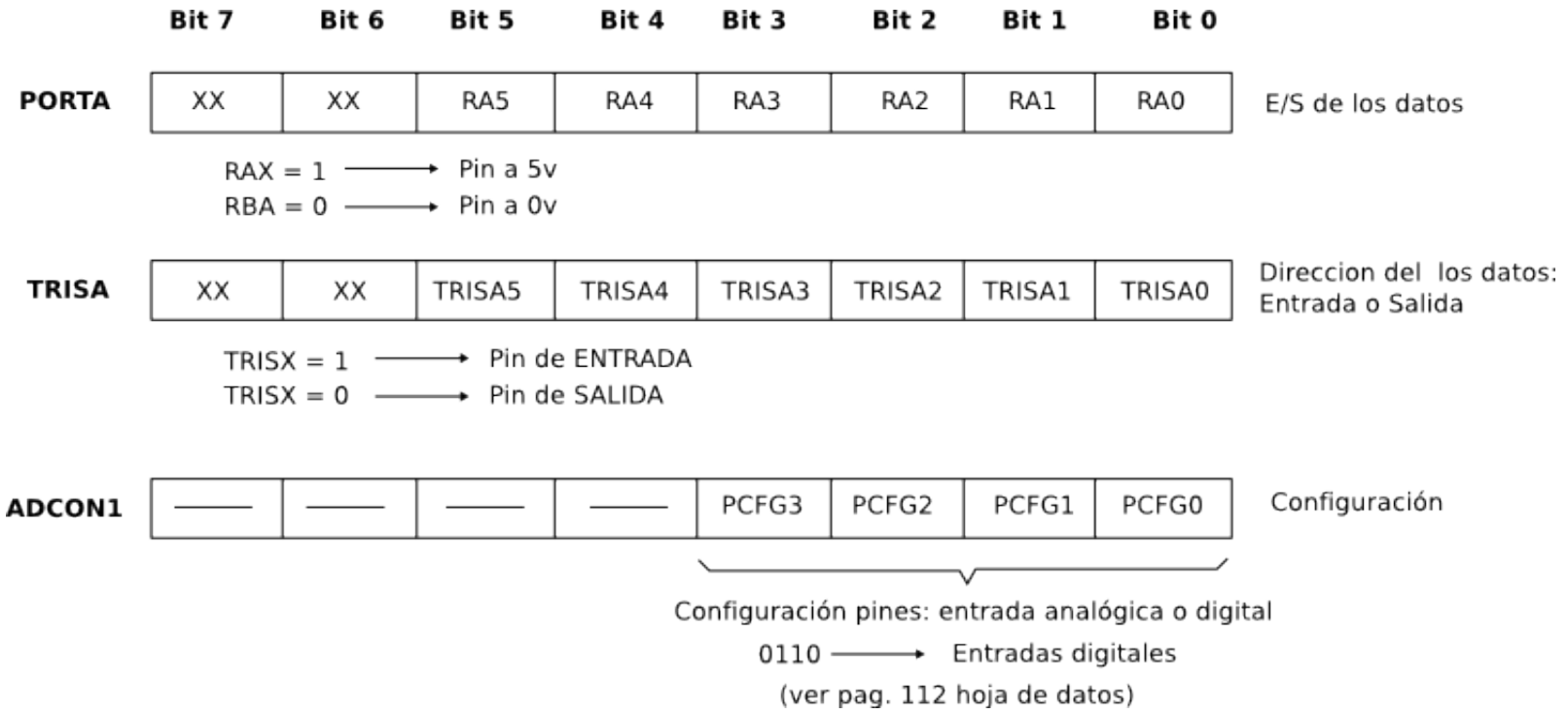
Puerto A

Entrada/salida digital: Puerto A

- **6 pines** independientes configurables para E/S
- Pines RA0,RA1,RA2,RA3 y RA5 se pueden usar como **entradas analógicas**
- Pin **RA4**:
  - Se puede usar como entrada de reloj del Timer0
  - Su salida es en **drenador abierto**



REGISTROS DEL PUERTO A



Ejemplo: Activar un pin de salida (I)

RA0-on.c

```
#include <pic16f876a.h>

void main(void)
{
  ADCON1=0x06;
  TRISA0 = 0;
  RA0 = 1;
  while(1);
}
```

Configurar todos los pines como Digitales

Configurar RA0 como salida

Activar pin RA0

Ejemplo: Puerto de salida de 6 bits

salida6.c

```
#include <pic16f876a.h>
void main(void)
{
  TRISA=0x00;
  ADCON1=0x06;
  PORTA = 0xFF;
  while(1);
}
```

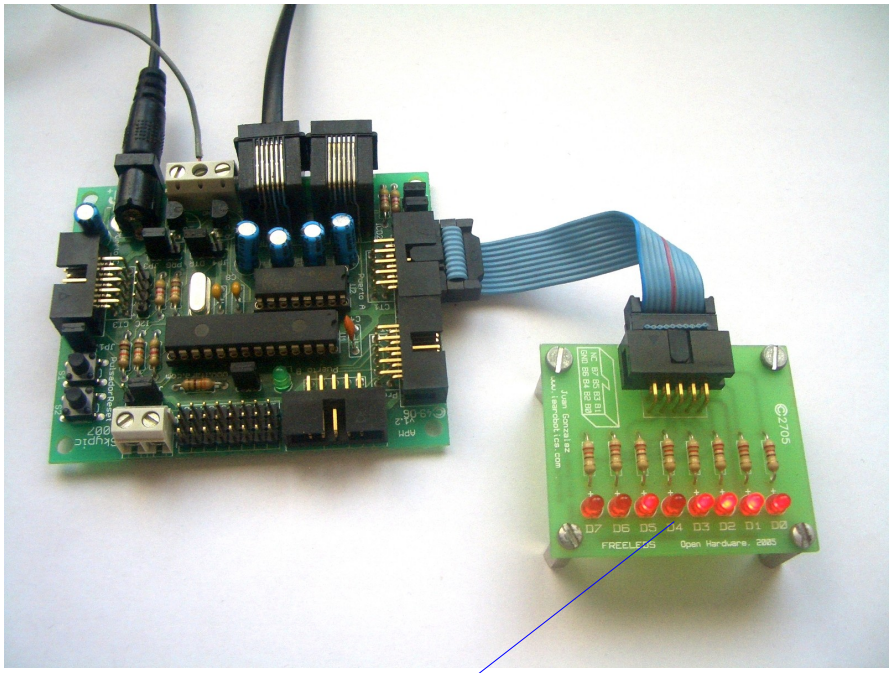
Configurar todos los pines para salida

Configurar todos los pines como Digitales

Activar todos los pines

- Conectar la tarjeta Freeleds para ver qué pasa

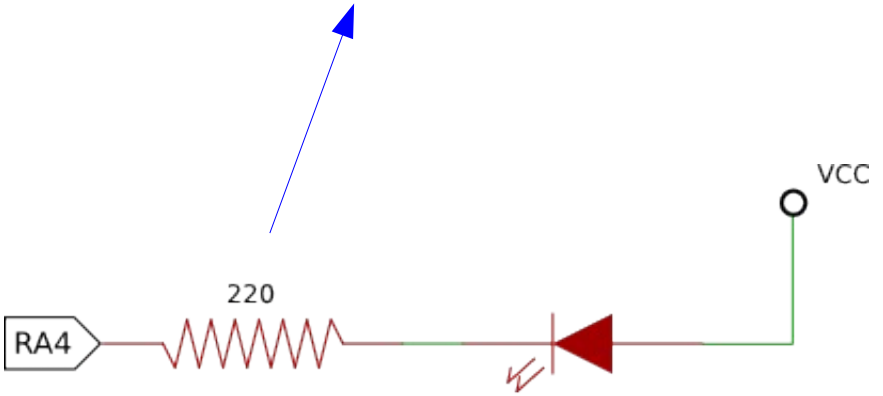
Ejemplo: Puerto de salida de 6 bits (II)



Ese led **NO** se enciende.

RA4 tiene el drenador abierto

Conexión de leds al pin RA4



Conexión de leds a los pines RA0-RA3,RA5

Ejemplo: Lectura del puerto A

RA0-led.c

```
#include <pic16f876a.h>
#define LED RB1

void main(void)
{
  ADCON1=0x06;
  TRISA0 = 1;
  TRISB1=0;

  while(1) {
    LED = RA0;
  }
}
```

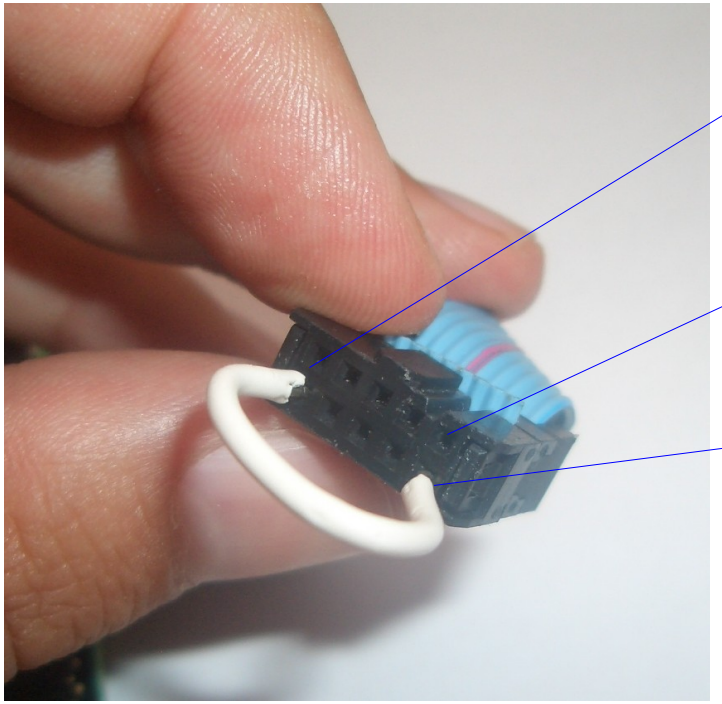
Sacar el estado de RA0 por el led

Configurar todos los pines como Digitales

Configurar RA0 como entrada

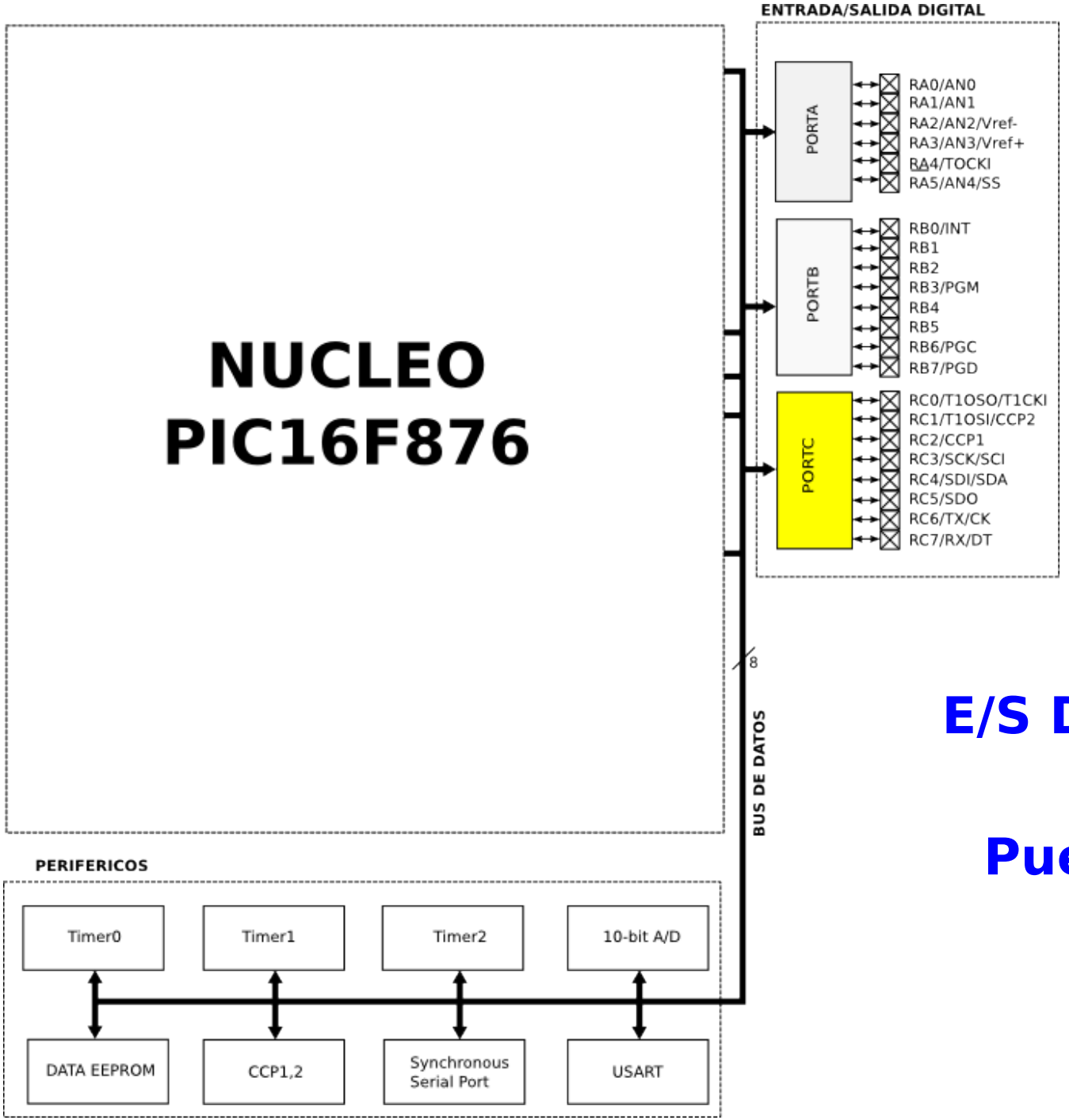
Configurar RB1 para salida (led)

Sacar por el led el estado de RA0



Para probarlo podemos usar un cable para conectar RA0 con GND o VCC

Programación de periféricos

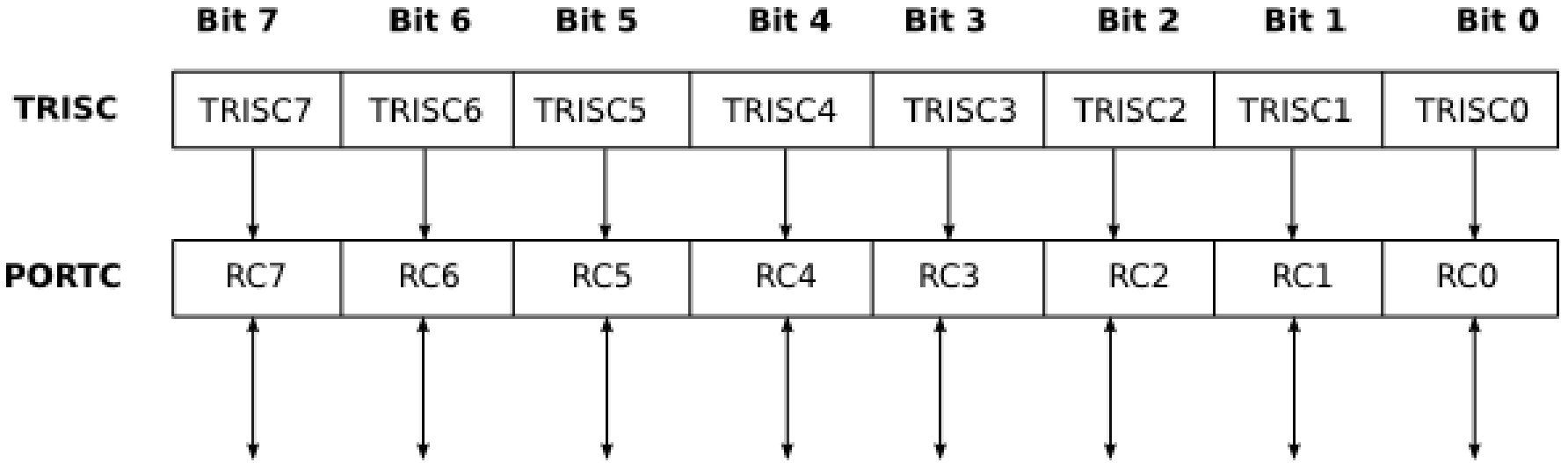


E/S Digital:

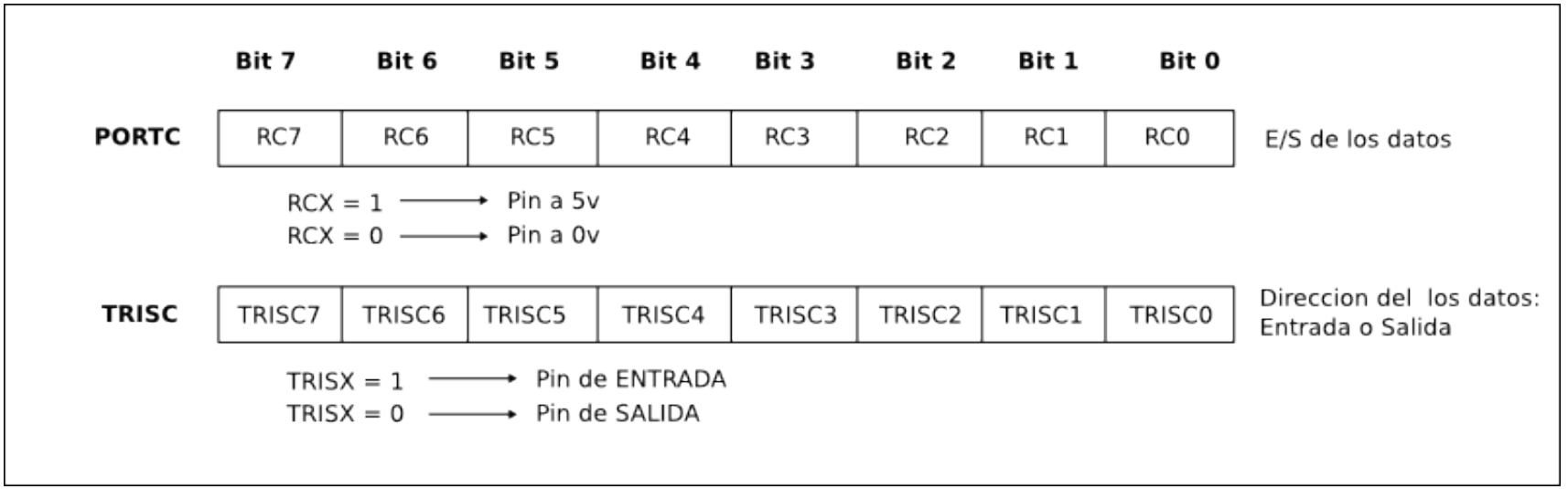
Puerto C

Entrada/salida digital: Puerto C

- 8 **pines** independientes configurables para E/S



REGISTROS DEL PUERTO C



Ejemplo: Activar un pin de salida (I)

RC0-on.c

```
#include <pic16f876a.h>
void main(void)
{
  TRISC0 = 0;
  RC0 = 1;
  while(1);
}
```

Configurar RC0 como salida

Activar pin RC0

Ejemplo: Puerto de salida de 8 bits

C-salida8.c

```
#include <pic16f876a.h>
void main(void)
{
  TRISC=0x00;
  PORTC = 0x55;
  while(1);
}
```

Configurar todos los pines para salida

Escribir un byte en el puerto C

Ejemplo: Lectura del puerto C

RC0-led.c

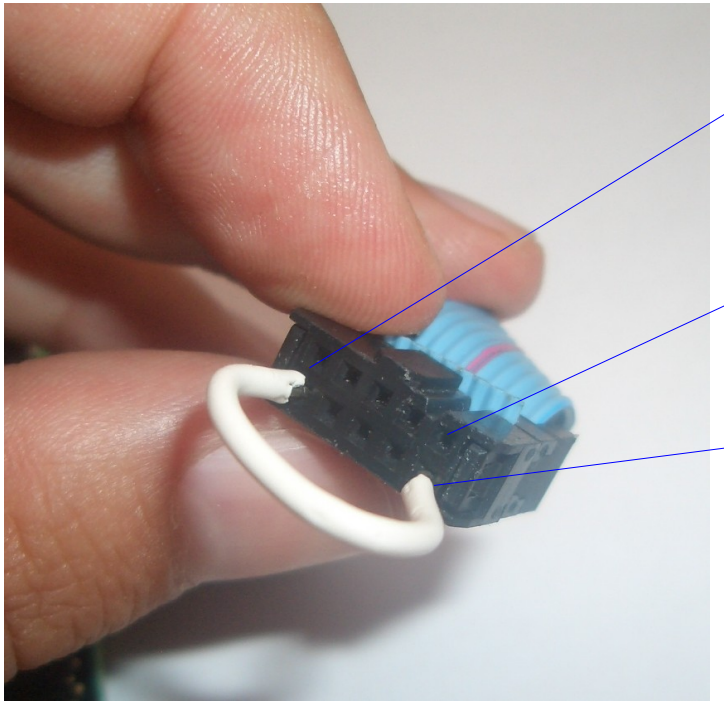
```
#include <pic16f876a.h>
#define LED RB1

void main(void)
{
  TRISC0 = 1;
  TRISB1=0;

  while(1) {
    LED = RC0;
  }
}
```

Sacar el estado de RC0 por el led

Configurar RC0 como entrada

Configurar RB1 para salida (led)

Sacar por el led el estado de RC0



Para probarlo podemos usar un cable para conectar RC0 con GND o VCC

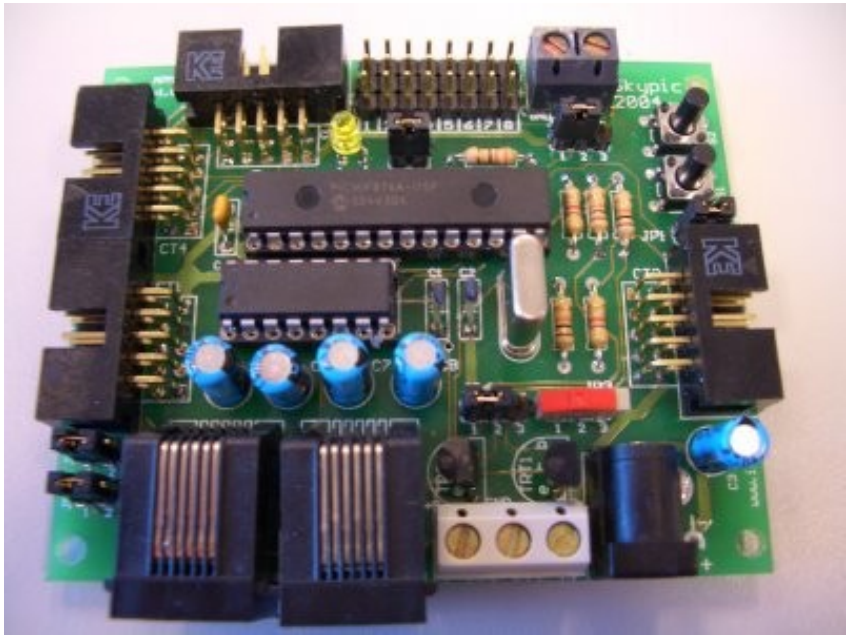
Ejercicio final de puertos

- Implementar un **contador de flancos de subida**
- El valor de este contador se deberá **sacar por el puerto C**, para ser visualizado en los leds
- Se usará el pin **RB0 como entrada de flancos**
- Programarlo mediante **espera activa**

MEJORAS:

- Que funcione mediante **interrupciones**
- En paralelo, que el LED del puerto B esté parpadeando a una frecuencia fija

PIC 16F87X



Juan González

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Andrés Prieto-Moreno

Flir Networked Systems

Ricardo Gómez

Flir Networked Systems