Práctica 7
Programación de interrupciones y temporizadores con C de CCS

- Introducción: Objetivos
- Gestión de interrupciones con C de CCS
- Gestión de temporizadores con C de CCS
- Tareas a realizar
- Anexo: funciones **`rand()`** y **`srand()`**

Introducción

Objetivos

- Dominar la gestión de las interrupciones y temporizadores del PIC16F877A mediante el lenguaje C de CCS.

Interrupciones: gestión en C de CCS

- El compilador de C de CCS incorpora varias directivas del preprocesador y funciones que facilitan mucho la gestión de interrupciones.
- Las directivas principales son del tipo **`#INT_causa`** donde **`"causa"`** indica una posible causa de interrupción (los posibles valores aparecen en la siguiente transparencia).
- Estas directivas indican al compilador que la función que se sitúa a continuación es la rutina de atención de interrupción correspondiente a la causa que indica la directiva. Ejemplo:

```
#INT_EXT
VOID rutina_ext_int(){
   instruccion_atencion_1;
   instrucción_atencion_2;
   .
   .
}
```

- Al usar estas directivas, el compilador se encarga de la comprobación de los *flags* (para seleccionar la función), de actualizarlos, y de guardar y recuperar el contexto

Interrupciones: gestión en C de CCS

- Directivas **`#INT_causa`** (no todas funcionan en todas las MCUs):

#INT_AD	Analog to digital conversion complete
#INT_ADOF	Analog to digital conversion timeout
#INT_BUSCOL	Bus collision
#INT_BUSCOL2	Bus collision 2 detected
#INT_BUTTON	Pushbutton
#INT_CANERR	An error has occurred in the CAN module
#INT_CANIRX	An invalid message has occurred on the CAN bus
#INT_CANRX0	CAN Receive buffer 0 has received a new message
#INT_CANRX1	CAN Receive buffer 1 has received a new message
#INT_CANTX0	CAN Transmit buffer 0 has completed transmission
#INT_CANTX1	CAN Transmit buffer 0 has completed transmission
#INT_CANTX2	CAN Transmit buffer 0 has completed transmission
#INT_CANWAKE	Bus Activity wake-up has occurred on the CAN bus
#INT_CCP1	Capture or Compare on unit 1
#INT_CCP2	Capture or Compare on unit 2
#INT_CCP3	Capture or Compare on unit 3
#INT_CCP4	Capture or Compare on unit 4
#INT_CCP5	Capture or Compare on unit 5
#INT_COMP	Comparator detect
#INT_COMP0	Comparator 0 detect
#INT_COMP1	Comparator 1 detect
#INT_COMP2	Comparator 2 detect
#INT_CR	Cryptographic activity complete
#INT_EEPROM	Write complete
#INT_ETH	Ethernet module interrupt
#INT_EXT	External interrupt
#INT_EXT1	External interrupt #1
#INT_EXT2	External interrupt #2
#INT_EXT3	External interrupt #3
#INT_I2C	I2C interrupt (only on 14000)
#INT_IC1	Input Capture #1
#INT_IC2QEI	Input Capture 2 / QEI Interrupt
#IC3DR	Input Capture 3 / Direction Change Interrupt
#INT_LCD	LCD activity
#INT_LOWVOLT	Low voltage detected
#INT_LVD	Low voltage detected
#INT_OSC_FAIL	System oscillator failed
#INT_OSCF	System oscillator failed
#INT_PMP	Parallel Master Port interrupt
#INT_PSP	Parallel Slave Port data in
#INT_PWMTB	PWM Time Base
#INT_RA	Port A any change on A0_A5
#INT_RB	Port B any change on B4-B7
#INT_RC	Port C any change on C4-C7
#INT_RDA	RS232 receive data available
#INT_RDA0	RS232 receive data available in buffer 0
#INT_RDA1	RS232 receive data available in buffer 1
#INT_RDA2	RS232 receive data available in buffer 2
#INT_RTCC	Timer 0 (RTCC) overflow
#INT_SPP	Streaming Parallel Port Read/Write
#INT_SSP	SPI or I2C activity
#INT_SSP2	SPI or I2C activity for Port 2
#INT_TBE	RS232 transmit buffer empty
#INT_TBE0	RS232 transmit buffer 0 empty
#INT_TBE1	RS232 transmit buffer 1 empty
#INT_TBE2	RS232 transmit buffer 2 empty
#INT_TIMER0	Timer 0 (RTCC) overflow
#INT_TIMER1	Timer 1 overflow
#INT_TIMER2	Timer 2 overflow
#INT_TIMER3	Timer 3 overflow
#INT_TIMER4	Timer 4 overflow
#INT_TIMER5	Timer 5 overflow
#INT_ULPWU	Ultra-low power wake up interrupt
#INT_USB	Universal Serial Bus activity

Interrupciones: gestión en C de CCS

- Las interrupciones se habilitan selectivamente mediante la función **`enable_interrupts(causa_habil)`** donde **`"causa_habil"`** indica el tipo de interrupción a habilitar. Los posibles valores de este parámetro se corresponden con (algunas de) las directivas de la transparencia anterior, y se definen en el fichero de cabecera principal de la MCU como constantes simbólicas. Los posibles valores para el PIC16F877A, son los siguientes:
  - GLOBAL
  - INT_RTCC
  - INT_RB
  - INT_EXT
  - INT_AD
  - INT_TBE
  - INT_RDA
  - INT_TIMER1
  - INT_TIMER2
  - INT_CCP1
  - INT_CCP2
  - INT_SSP
  - INT_PSP
  - INT_BUSCOL
  - INT_EEPROM
  - INT_TIMER0
  - INT_COMP
- El valor “GLOBAL” sirve para habilitar las interrupciones globalmente. Es decir, hasta que no se “habilita” este valor, no tiene efecto habilitar cualquier causa concreta de interrupción.

Interrupciones: gestión en C de CCS

- Pueden habilitarse varios tipos de interrupción en una misma llamada a la función **`enable_interrupts(causa_habil)`** si **`"causa_habil"`** es una serie de causas separadas por **`'|'`**. Ejemplo:

```
enable_interrupts(INT_EXT|INT_RB);
```

- Por el contrario, la función **`disable_interrupts(causa_inhabil)`** deshabilita la(s) causa(s) que se indican como parámetro.

- Puede seleccionarse el flanco de activación de la interrupción externa asociada al pin INT (o sea, RB0) mediante la función **`ext_int_edge(flanco)`**, donde el parámetro **`"flanco"`** puede tomar uno de estos dos valores definidos en el fichero de cabecera principal de la MCU como constantes simbólicas:
  - **`H_TO_L`** (selecciona flanco de bajada)
  - **`L_TO_H`** (selecciona flanco de subida)

Interrupciones: ejemplo

```
#include <16f877a.h>
#fuses HS,NOWDT,NOLVP
#use delay (clock =8000000 )
#USE FAST_IO(B)
#BYTE PORTB=0x06

#INT_EXT                          //Directiva para identificar ISR por pin INT
 VOID ext_isr(){                  //Definicion ISR de pin INT
  output_toggle(pin_b2);          //Cambia estado pin RB2 cada llamada a ISR
}

void main() {
  set_tris_b(0b00000001);         //RB0 como entrada
  portb=0x0;                      //Inicia puerto B a cero
  enable_interrupts(int_ext);     //Habilita interrupcion por pin INT
  ext_int_edge(l_to_h);           //Selecciona int. por flanco subida en pin INT
  enable_interrupts(global);      //Habilitacion global de interrupciones
  while(1){}
}
```

Temporizadores: gestión en C de CCS

- Las funciones para configurar el comportamiento de los temporizadores 0 y 1 en el C de CCS son del tipo **`setup_timer_X(modo)`** donde **`"X"`** indica el número de temporizador (0 ó 1) y **`"modo"`** puede tomar distintos valores que se definen en el fichero de cabecera principal de la MCU como constantes simbólicas, y que básicamente son máscaras para configurar los registros de control correspondientes.
- En el PIC16F877A, los posibles valores de **`"modo"`**, para los timers 0 y 1 y los registros afectados son los siguientes:

Modo Timer 0	Bits 5:0 OPTION_REG
RTCC_INTERNAL	0
RTCC_EXT_L_TO_H	32
RTCC_EXT_H_TO_L	48
RTCC_DIV_1	8
RTCC_DIV_2	0
RTCC_DIV_4	1
RTCC_DIV_8	2
RTCC_DIV_16	3
RTCC_DIV_32	4
RTCC_DIV_64	5
RTCC_DIV_128	6
RTCC_DIV_256	7

Modo Timer 1	Bits 5:0 T1CON
T1_DISABLED	0
T1_INTERNAL	0x85
T1_EXTERNAL	0x87
T1_EXTERNAL_SYNC	0x83
T1_CLK_OUT	8
T1_DIV_BY_1	0
T1_DIV_BY_2	0x10
T1_DIV_BY_4	0x20
T1_DIV_BY_8	0x30

Temporizadores: gestión en C de CCS

- Pueden definirse varios modos (si son compatibles) en una misma llamada a la función **`setup_timer_X(modo)`** si **`"modo"`** es una serie de valores separados por `'|'`. Ejemplo:

```
setup_timer_0(RTCC_internal|RTCC_DIV_128);
```

- Las funciones para leer el valor de los timers son del tipo **`get_timerX()`** donde **`"X"`** indica el número de temporizador. La función devuelve un entero de 8 ó16 bits dependiendo del temporizador. Ejemplo:

```
valor=get_timer0();
```

- Las funciones para cargar (inicializar) un valor en los timers son del tipo **`set_timerX(valor)`** donde **`"valor" es`** un entero de 8 ó16 bits dependiendo del temporizador. Ejemplo:

```
set_timer0(50);
```

Temporizadores: gestión en C de CCS

- El cálculo del tiempo que tardará realmente un timer en desbordarse depende por tanto de varios factores. Una forma de calcularlo idealmente sería (ejemplo para Timer0):

  $T = T_{CM} * Valor\_Prescaler * (256 - Carga\_Timer0)$

  Donde T_{CM} es el tiempo de un ciclo de máquina ($T_{CM} = 4 * T_{CReloj}$).

- Ejemplo para calcular la carga del Timer0 de modo que el tiempo de desbordamiento ideal sea de 16ms con un valor de prescaler de 256 y trabajando a 8MHz:

  $Carga\_Timer0 = 256 - 16ms/(4/8000000*256) = 131$

- Realmente, este cálculo es sólo una aproximación debido a ineficiencias del compilador.

Temporizadores: ejemplo

```
#include <16f877a.h>
#fuses HS,NOWDT,NOLVP
#use delay (clock =8000000)

int conta=0;

#INT_TIMER0                 //Directiva para identificar ISR por overflow timer 0
 VOID timer0_isr(){         //Definicion ISR de overflow del timer 0
  conta++;
  if (conta==100){          //Sólo actúa cada 100 overflows del timer 0
   output_toggle(pin_B2); //Cambia estado pin RB2 cada llamada a ISR
   conta=0;
  }
}

void main() {
  setup_timer_0(RTCC_internal|RTCC_DIV_128); //Reloj interno y prescaler de 128
  set_timer0(0);                             //Inicializa timer 0 con valor 0
  enable_interrupts(int_timer0);    //Habilita interrupcion por overflow timer 0
  enable_interrupts(global);        //Habilitacion global de interrupciones
  while(1){}
}
```

Tareas a realizar

1. Adaptar la tarea 4 de la práctica 5 para que la detección del pulsador RB0 se gestione mediante interrupciones.
2. Adaptar la tarea 3 de la práctica 5 para que la detección del pulsador RB0 se gestione mediante interrupciones, y el periodo de encendido de LEDs mediante el temporizador 0.
3. Programar un juego de “caza de moscas”. En el LCD aparecerán “moscas” (sólo una en un momento dado) representadas por el símbolo “*”. Las moscas aparecerán aleatoriamente en la fila de arriba o en la de abajo del LCD, y aleatoriamente en la posición de carácter 6,7,8 ó 9. Las moscas se desplazarán a derecha o izquierda aleatoriamente desde su casilla de aparición, con una velocidad de un carácter cada 0,5 segundos. Una mosca que “sale” de la pantalla por la izquierda, o que va más allá del carácter 14 por la derecha de la fila, se salva, y aparece una nueva mosca. Además, en el LCD aparecerá una “mira” representada por el símbolo “+” que el jugador puede mover mediante pulsación de los pines RB7 a RB4 (arriba, abajo, izquierda y derecha respectivamente). La mira comienza la partida en la posición (7,1). Cuando la mira y la mosca coinciden en una posición, el jugador debe “disparar” (pulsación de RB0) para matar a la mosca, y si es así aparecerá una nueva mosca. En las posiciones (15,1) y (16,1) se mostrará el número de moscas que se han salvado. En las posiciones (15,2) y (16,2) se mostrará el número de moscas cazadas. La partida acaba cuando se salva o muere la mosca número 20.

Debe entregarse sólo el programa desarrollado para la tarea 3.

ANEXO: Funciones `rand()` y `srand()`

- Estas funciones son útiles para las selecciones aleatorias de la tarea 3.
- Se definen en la librería <STDLIB.h>
- **`rand()`** . Devuelve un valor comprendido entre 0 y RAND_MAX.
- **`srand(n)`** . Se usa el valor **`n`** como semilla para reiniciar la secuencia de generación de números pseudoaleatorios.