Universidad de Barcelona

Arthur Font Gouveia 20222613

Ángel Rubio Giménez 20222484

Programación de Arquitecturas Empotradas

Práctica 3 - Timers e Interrupciones

Barcelona

2020

Índice

1. Proyecto

1.1 Objetivos

Los objetivos de esta práctica han sido la configuración de los Timers. El microcontrolador tiene cuatro timers diferentes con cinco contadores, de los cuales hemos utilizado dos, uno para controlar cuándo encender y apagar los LEDs y otro para configurar un reloj con alarma.

1.2 Recursos utilizados

Primero tenemos que habilitar la interrupción de cada timer en los tres niveles: a nivel de microcontrolador, a nivel del NVIC y a nivel de periférico.

Table 6-39. NVIC Interrupts (continued)

NVIC INTERRUPT INPUT	SOURCE	FLAGS IN SOURCE
INTISR[3]	WDT_A	
INTISR[4]	FPU_INT (2)	Combined interrupt from flags in the FPSCR (part of Cortex-M4 FPU)
INTISR[5]	FLCTL	Flash Controller interrupt flags
INTISR[6]	COMP_E0	Comparator_E0 interrupt flags
INTISR[7]	COMP_E1	Comparator_E1 interrupt flags
INTISR[8]	Timer_A0	TA0CCTL0.CCIFG
INTISR[9]	Timer_A0	TA0CCTLx.CCIFG (x = 1 through 4), TA0CTL.TAIFG
INTISR[10]	Timer_A1	TA1CCTL0.CCIFG
INTISR[11]	Timer_A1	TA1CCTLx.CCIFG (x = 1 through 4), TA1CTL.TAIFG

Figura 1: Tabla de las interrupciones del NVIC

Después hay que seleccionar la fuente del clock, nosotros hemos escogido el SMCLK que funciona a 24MHz, y también hay que seleccionar el modo de trabajo, en nuestro caso hemos empezado en stop y cuándo necesitamos el clock lo ponemos en modo up. Para acabar hay que darle un valor a uno de los contadores(TAxCCRn) de cada clock utilizado.

1.3 Problemas

El mayor problema que hemos tenido ha sido conseguir configurar correctamente el clock la primera vez. Al principio nos dejabamos algún nivel de interrupción sin activar, pero una vez que teníamos todos los niveles activados también nos olvidamos de ponerlo en un modo diferente de stop.

Table 17-4. TAxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit resets TAxR, the timer clock divider logic, and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Figura 2: Tabla del registro de control de los timers

1.4 Conclusiones

Esta práctica nos ha servido para entender cómo funciona la habilitación de interrupciones en los tres niveles y comprender cómo funcionan los timers del robot.

2. Código comentado

```
/*********************************
 *
 * Practica_02_PAE Programació de Ports
 * i pràctica de les instruccions de control de flux:
 * "do ... while", "switch ... case", "if" i "for"
 * UB, 02/2017.
 ***************************/

#include <msp432p401r.h>
#include <stdio.h>
#include <stdint.h>
#include "lib_PAE2.h" //Libreria grafica + configuracion reloj MSP432

#define Button_S1 1
#define Button_S2 2
#define Jstick_Left 3
#define Jstick_Right 4
#define Jstick_Up 5
#define Jstick_Down 6
#define Jstick_Center 7

char saludo[16] = " PRACTICA 2 PAE";//max 15 caracteres visibles
char cadena[16];//Una linea entera con 15 caracteres visibles + uno
oculto de terminacion de cadena (codigo ASCII 0)
char limite[16]; //Mostrar aviso de tiempo limite
char milisecs[16]; //Mostrar retraso actual de los LEDs
char hora[16]; //Mostrar la hora
char alarma[16]; //Mostrar la alarma
char avisoAl[16]; //Mostrar aviso de la alarma
char borrado[] = "               "; //una linea entera de 15 espacios en
blanco
uint8_t linea = 0;
uint8_t estado=0;
uint8_t estado_anterior = 8;
uint32_t retraso = 500000;
int i = 0;
int temps = 0;
int temps2= 0;
int h = 0;
int min = 0;
int secs = 0;
int hAlarma = 0;
int minAlarma = 0;
int secsAlarma = 0;
int bool = 0;
```

```
/
*************************************************************************
*
 * INICIALIZACIÓN DEL CONTROLADOR DE INTERRUPCIONES (NVIC).
 *
 * Sin datos de entrada
 *
 * Sin datos de salida
 *

*************************************************************************
*/
void init_interrupciones(){
    // Configuracion al estilo MSP430 "clasico":
    // --> Enable Port 4 interrupt on the NVIC.
    // Segun el Datasheet (Tabla "6-39. NVIC Interrupts", apartado "6.7.2
Device-Level User Interrupts"),
    // la interrupcion del puerto 4 es la User ISR numero 38.
    // Segun el Technical Reference Manual, apartado "2.4.3 NVIC
Registers",
    // hay 2 registros de habilitacion ISER0 y ISER1, cada uno para 32
interrupciones (0..31, y 32..63, resp.),
    // accesibles mediante la estructura NVIC->ISER[x], con x = 0 o x =
1.
    // Asimismo, hay 2 registros para deshabilitarlas: ICERx, y dos
registros para limpiarlas: ICPRx.

    //Int. port 3 = 37 corresponde al bit 5 del segundo registro ISER1:
    NVIC->ICPR[1] |= BIT5; //Primero, me aseguro de que no quede ninguna
interrupcion residual pendiente para este puerto,
    NVIC->ISER[1] |= BIT5; //y habilito las interrupciones del puerto
    //Int. port 4 = 38 corresponde al bit 6 del segundo registro ISERx:
    NVIC->ICPR[1] |= BIT6; //Primero, me aseguro de que no quede ninguna
interrupcion residual pendiente para este puerto,
    NVIC->ISER[1] |= BIT6; //y habilito las interrupciones del puerto
    //Int. port 5 = 39 corresponde al bit 7 del segundo registro ISERx:
    NVIC->ICPR[1] |= BIT7; //Primero, me aseguro de que no quede ninguna
interrupcion residual pendiente para este puerto,
    NVIC->ISER[1] |= BIT7; //y habilito las interrupciones del puerto

    //Int del timer
    NVIC->ICPR[0] |= BIT8; //Primero, me aseguro de que no quede ninguna
interrupcion residual pendiente para este timer,
    NVIC->ISER[0] |= BIT8; //y habilito las interrupciones del timer

    NVIC->ICPR[0] |= BITA; //Primero, me aseguro de que no quede ninguna
interrupcion residual pendiente para este timer,
    NVIC->ISER[0] |= BITA; //y habilito las interrupciones del timer

    __enable_interrupt(); //Habilitamos las interrupciones a nivel global
del micro.
}
```

```
/
*************************************************************************
*
 * INICIALIZACIÓN DE LA PANTALLA LCD.
 *
 * Sin datos de entrada
 *
 * Sin datos de salida
 *

*************************************************************************
*/
void init_LCD(void)
{
    halLcdInit(); //Inicializar y configurar la pantallita
    halLcdClearScreenBkg(); //Borrar la pantalla, rellenando con el color
de fondo
}

/
*************************************************************************
*
 * BORRAR LINEA
 *
 * Datos de entrada: Linea, indica la linea a borrar
 *
 * Sin datos de salida
 *

*************************************************************************
*/
void borrar(uint8_t Linea)
{
    halLcdPrintLine(borrado, Linea, NORMAL_TEXT); //escribimos una linea
en blanco
}

/
*************************************************************************
*
 * ESCRIBIR LINEA
 *
 * Datos de entrada: Linea, indica la linea del LCD donde escribir
 *                   String, la cadena de caracteres que vamos a escribir
 *
 * Sin datos de salida
 *

*************************************************************************
*/
void escribir(char String[], uint8_t Linea)

{
```

```
    halLcdPrintLine(String, Linea, NORMAL_TEXT); //Enviamos la String al
LCD, sobreescribiendo la Linea indicada.
}

/
*************************************************************************
*
 * INICIALIZACIÓN DE LOS BOTONES & LEDS DEL BOOSTERPACK MK II.
 *
 * Sin datos de entrada
 *
 * Sin datos de salida
 *

*************************************************************************
*/
void init_botons(void)
{
    // Clock TA0
    // Seleccionamos la fuente del reloj
    TA0CTL |= TASSEL_2; // Escoger el clock SMCLK
    TA0CCR0 = 20000; // Limite superior del clock
    TA0CTL |= MC_0; //Empezamos con el clock en stop
    TA0CCTL0 |= CCIE; // Enable interrupt
    TA1CTL |= ID_2; // Dividimos el TA0 entre 4

    // Clock TA1
    // Seleccionamos la fuente del reloj
    TA1CTL |= TASSEL_2; // Escoger el clock SMCLK
    TA1CCR0 = 4500; // Limite superior del clock
    TA1CTL |= MC_0; //Empezamos con el clock en stop
    TA1CCTL0 |= CCIE; // Enable interrupt



    //Configuramos botones y leds
    //***************************

    //Leds RGB del MK II:
      P2DIR |= 0x50;  //Pines P2.4 (G), 2.6 (R) como salidas Led (RGB)
      P5DIR |= 0x40;  //Pin P5.6 (B)como salida Led (RGB)
      P2OUT &= 0xAF;  //Inicializamos Led RGB a 0 (apagados)
      P5OUT &= ~0x40; //Inicializamos Led RGB a 0 (apagados)

    //Boton S1 del MK II:
      P5SEL0 &= ~0x02;   //Pin P5.1 como I/O digital,
      P5SEL1 &= ~0x02;   //Pin P5.1 como I/O digital,
      P5DIR &= ~0x02; //Pin P5.1 como entrada
      P5IES &= ~0x02;   // con transicion L->H
      P5IE |= 0x02;     //Interrupciones activadas en P5.1,
      P5IFG = 0;    //Limpiamos todos los flags de las interrupciones del
puerto 5
      //P5REN: Ya hay una resistencia de pullup en la placa MK II
```

```
    //Boton S2 del MK II:
      P3SEL0 &= ~0x20;   //Pin P3.5 como I/O digital,
      P3SEL1 &= ~0x20;   //Pin P3.5 como I/O digital,
      P3DIR &= ~0x20; //Pin P3.5 como entrada
      P3IES &= ~0x20;   // con transicion L->H
      P3IE |= 0x20;   //Interrupciones activadas en P3.5
      P3IFG = 0;  //Limpiamos todos los flags de las interrupciones del
puerto 3
      //P3REN: Ya hay una resistencia de pullup en la placa MK II

    //Configuramos los GPIOs del joystick del MK II:
      P4DIR &= ~(BIT1 + BIT5 + BIT7);   //Pines P4.1, 4.5 y 4.7 como
entrades,
      P4SEL0 &= ~(BIT1 + BIT5 + BIT7);  //Pines P4.1, 4.5 y 4.7 como I/O
digitales,
      P4SEL1 &= ~(BIT1 + BIT5 + BIT7);
      P4REN |= BIT1 + BIT5 + BIT7;  //con resistencia activada
      P4OUT |= BIT1 + BIT5 + BIT7;  // de pull-up
      P4IE |= BIT1 + BIT5 + BIT7;   //Interrupciones activadas en P4.1,
4.5 y 4.7,
      P4IES &= ~(BIT1 + BIT5 + BIT7);   //las interrupciones se generaran
con transicion L->H
      P4IFG = 0;    //Limpiamos todos los flags de las interrupciones del
puerto 4

      P5DIR &= ~(BIT4 + BIT5);  //Pines P5.4 y 5.5 como entrades,
      P5SEL0 &= ~(BIT4 + BIT5); //Pines P5.4 y 5.5 como I/O digitales,
      P5SEL1 &= ~(BIT4 + BIT5);
      P5IE |= BIT4 + BIT5;  //Interrupciones activadas en 5.4 y 5.5,
      P5IES &= ~(BIT4 + BIT5);  //las interrupciones se generaran con
transicion L->H
      P5IFG = 0;    //Limpiamos todos los flags de las interrupciones del
puerto 4
    // - Ya hay una resistencia de pullup en la placa MK II
}

/
**********************************************************************
*
 * DELAY - A CONFIGURAR POR EL ALUMNO - con bucle while
 *
 * Datos de entrada: Tiempo de retraso. 1 segundo equivale a un retraso
de 1000000 (aprox)
 *
 * Sin datos de salida
 *

**********************************************************************
*/
void delay_t (uint32_t time)
{
```

```
   /*************************
     * A RELLENAR POR EL ALUMNO
     *************************/
   TA1CTL |= MC_1; // Contar de 0 a TA1CCR0(mode UP)

   do {
   } while(temps2 < time); // Salir del bucle cuando el contador llegue
al tiempo
   TA1CTL |= MC_0; // Poner el TA1 en stop

}

void delay_t2 ()
{

   /*************************
     * A RELLENAR POR EL ALUMNO
     *************************/
   TA0CTL |= MC_1; //Contar de 0 a TA0CCR0(mode UP)

   do {
   } while(temps < 1000); // Salir del bucle cuando el contador llegue al
tiempo
   TA0CTL |= MC_0; //Poner el TA0 en stop

}

/
*****************************************************************************
****
 * CONFIGURACIÓN DEL PUERTO 7. A REALIZAR POR EL ALUMNO
 *
 * Sin datos de entrada
 *
 * Sin datos de salida
 *

*****************************************************************************
***/
void config_P7_LEDS (void)
{

    //A RELLENAR POR EL ALUMNO
    P7SEL0 = 0x00;
    P7SEL1 = 0x00; //Configurar todos los pines del puerto 7 como GPIO
    P7DIR |= 0xFF; //Configurar todos los pines del puerto 7 como salida
    P7OUT &= ~(0xFF); //Apagar los LEDs
}

void main(void)
{

    WDTCTL = WDTPW+WDTHOLD;         // Paramos el watchdog timer
```

```
    // Inicializaciones:
    init_ucs_24MHz();       // Ajustes del clock (Unified Clock System)
    init_botons();         // Configuramos botones y leds
    config_P7_LEDS();
    init_interrupciones();  // Configurar y activar las interrupciones de
los botones
    init_LCD();             // Inicializamos la pantalla

    halLcdPrintLine(saludo,linea, INVERT_TEXT); // Escribimos saludo en
la primera linea
    linea++;                    // Aumentamos el valor de linea y con
ello pasamos a la linea siguiente
    int tiempo = 1000;

    // Bucle principal (infinito):
    do
    {

    if (estado_anterior != estado)          // Dependiendo del valor del
estado se encenderá un LED u otro.
    {
        sprintf(cadena,"State %02d", estado);  // Guardamos en cadena la
siguiente frase: Estado "valor del estado",
                                               //con formato decimal, 2
cifras, rellenando con 0 a la izquierda.
        escribir(cadena,linea); // Escribimos la cadena al LCD
        estado_anterior = estado; // Actualizamos el valor de
estado_anterior, para que no esté siempre escribiendo.


        /*********************************************************+
            A RELLENAR POR EL ALUMNO BLOQUE switch ... case
        Para gestionar las acciones:
        Boton S1, estado = 1
        Boton S2, estado = 2
        Joystick left, estado = 3
        Joystick right, estado = 4
        Joystick up, estado = 5
        Joystick down, estado = 6
        Joystick center, estado = 7
        *********************************************************/
        switch(estado){
                    case Button_S1:
                        P2OUT |= 0x50;
                        P5OUT |= 0x40; // Encender los tres RGB
                        if(bool){ // Si bool es 1, aumentamos la hora del
reloj, sino aumentamos la hora de la alarma
                            h++;
                        }else{
                            if(hAlarma < 23){
                                hAlarma++;
```

```
                            }else{  // Cuando pasamos de 23 horas,
ponemos el valor de la hora a 0
                                hAlarma = 0;
                            }
                        }
                        break;
                    case Button_S2:
                        P2OUT &= ~(0x50);
                        P5OUT &= ~(0x40); // Apagar los tres RGB
                        if(bool){ // Si bool es 1, aumentamos los minutos
del reloj, sino aumentamos los minutos de la alarma
                            min++;
                        }else{
                            if(minAlarma < 59){
                                minAlarma++;
                            }else{ // Cuando pasamos de 59 minutos,
ponemos el valor de los minutos a 0
                                minAlarma = 0;
                            }
                        }
                        break;
                    case Jstick_Left:
                        P2OUT |= 0x50;
                        P5OUT |= 0x40;//Encender los tres RGB
                        for(i = 7; i>=0; i--){
                            P7OUT = 0x00;
                            if(i == 0){
                                P7OUT |= 0x01;
                            }else if(i == 1){
                                P7OUT |= 0x02;
                            }else if(i == 2){
                                P7OUT |= 0x04;
                            }else if(i == 3){
                                P7OUT |= 0x08;
                            }else if(i == 4){
                                P7OUT |= 0x10;
                            }else if(i == 5){
                                P7OUT |= 0x20;
                            }else if(i == 6){
                                P7OUT |= 0x40;
                            }else if(i == 7){
                                P7OUT |= 0x80;
                            }
                            delay_t(tiempo); // Encender los LEDs de
derecha a izquierda
                            temps2 = 0;
                        }
                        P7OUT = 0x00;
                        break;
                    case Jstick_Right:
                        P2OUT |= 0x50;
                        P5OUT &= ~(0x40); // Apagar el LED B i encender
el R y G
```

```
                        for(i = 0; i<8; i++){
                            P7OUT = 0x00;
                            if(i == 0){
                                P7OUT |= 0x01;
                            }else if(i == 1){
                                P7OUT |= 0x02;
                            }else if(i == 2){
                                P7OUT |= 0x04;
                            }else if(i == 3){
                                P7OUT |= 0x08;
                            }else if(i == 4){
                                P7OUT |= 0x10;
                            }else if(i == 5){
                                P7OUT |= 0x20;
                            }else if(i == 6){
                                P7OUT |= 0x40;
                            }else if(i == 7){
                                P7OUT |= 0x80;
                            }
                            delay_t(tiempo); // Encender los LEDs de
izquierda a derecha
                            temps2 = 0;
                        }
                        P7OUT = 0x00;
                        break;
                    case Jstick_Up:
                        borrar(linea+1);
                        P2OUT |= 0x40;
                        P2OUT &= ~(0x20);
                        P5OUT |= 0x40;  // Apagar el LED G i encender el
R y B
                        if(tiempo < 2000){
                            tiempo += 10; // Aumenta el timepo que estan
encendidos los LEDs
                            borrar(linea+2); // Borrar el aviso del
maximo
                        }else{
                            sprintf(limite,"Max Time"); // Escribir en la
cadena
                            escribir(limite, linea+2); // Escribimos un
aviso en la pantalla LCD que hemos llegado al tiempo limite
                        }
                        if(bool){ // Si bool es 1, aumentamos los
segundos del reloj, sino aumentamos los segundos de la alarma
                            secs++;
                        }else{
                            if(secsAlarma < 59){
                                secsAlarma++;
                            }else{ // Cuando pasamos de 59 segundos,
ponemos el valor de los segundos a 0
                                secsAlarma = 0;
                            }
                        }
```

```
                        break;
                    case Jstick_Down:
                        borrar(linea+1);
                        P2OUT |= 0x20;
                        P2OUT &= ~(0x40);
                        P5OUT |= 0x40; // Apagar el LED R i encender el G
y B
                        if (tiempo > 10){ // Definimos un tiempo mínimo
para que el tiempo no sea negativo
                            tiempo -= 10; // Disminuye el timepo que
estan encendidos los LEDs
                            borrar(linea+2); // Borrar el aviso del
maximo
                        }else{
                            sprintf(limite,"Min Time"); // Escribir en la
cadena
                            escribir(limite, linea+2); // Escribimos un
aviso en la pantalla LCD que hemos llegado al tiempo limite
                        }

                        break;
                    case Jstick_Center:
                        P2OUT ^= 0x50;
                        P5OUT ^= 0x40; // Invierte el estado de los LEDs
RGB
                        if(bool){ // Canviamos el valor de bool, que
sirve para seleccionar si aumentamos la hora o la alarma
                            bool = 0;
                        }else{
                            bool = 1;
                        }
                        break;
                    }
    }
    sprintf(milisecs,"Time LED:%02dms", tiempo); // Escribir en la cadena
el Time LED
    escribir(milisecs,linea+1); // Se muestra en la pantalla LCD el
tiempo actual que tarda los LEDs en encenderse

    sprintf(hora, "Hour: %02d:%02d:%02d", h, min, secs); // Escribir en
la cadena la hora
    escribir(hora, linea+3); // Se muestra en la pantalla LCD el reloj
del programa

    sprintf(alarma, "Alarm: %02d:%02d:%02d", hAlarma, minAlarma,
secsAlarma); // Escribir en la cadena la alarma
    escribir(alarma, linea+4); // Se muestra en la pantalla LCD la alarma

    if(h == hAlarma && min == minAlarma && secs == secsAlarma){
        sprintf(avisoAl, "ALARM!!!!");  // Escribir en la cadena el aviso
de la alarma
        escribir(avisoAl, linea+5); // Se muestra en la pantalla LCD el
aviso de la alarma
```

```
    }else{
        borrar(linea+5); // Borramos el aviso de la alarma
    }

    // Retraso de 1 segundo para ir aumentando la hora de nuestro reloj
    delay_t2();
    temps = 0; // Ponemos el contador del retraso del clock a 0
    if(secs < 59){ // Comprobamos si los segundos son menores a 59
        secs++; // Aumentamos en uno los segundos
    }else{
        secs = 0; // Reseteamos el valor de los segundos
        if(min < 59){ // Comprobamos si los minutos son menores a 59
            min++; // Aumentamos en uno los minutos
        }else{
            min = 0; // Reseteamos el valor de los segundos
            if(h < 23){ // Comprobamos si las horas son menores a 23
                h++; // Aumentamos en uno las horas
            }else{
                h = 0; // Reseteamos el valor de las horas
            }
        }
    }

    }while(1); //Condicion para que el bucle sea infinito
}


/
*****************************************************************************
*
 * RUTINAS DE GESTION DE LOS BOTONES:
 * Mediante estas rutinas, se detectará qué botón se ha pulsado
 *
 * Sin Datos de entrada
 *
 * Sin datos de salida
 *
 * Actualizar el valor de la variable global estado
 *

*****************************************************************************
*/

//ISR para las interrupciones del puerto 3:
void PORT3_IRQHandler(void){//interrupcion del pulsador S2
    uint8_t flag = P3IV; //guardamos el vector de interrupciones. De
paso, al acceder a este vector, se limpia automaticamente.
    P3IE &= 0xDF;  //interrupciones del boton S2 en port 3 desactivadas
    estado_anterior=0;
    switch(flag){
        case 0x0C:
            estado = Button_S2;
            break;
```

```
    }//Guardamos el estado de la interrupcion

    /*********************************************************+
        A RELLENAR POR EL ALUMNO
    Para gestionar los estados:
    Boton S1, estado = 1
    Boton S2, estado = 2
    Joystick left, estado = 3
    Joystick right, estado = 4
    Joystick up, estado = 5
    Joystick down, estado = 6
    Joystick center, estado = 7
    *********************************************************/


    P3IE |= 0x20;   //interrupciones S2 en port 3 reactivadas
}

//ISR para las interrupciones del puerto 4:
void PORT4_IRQHandler(void){  //interrupción de los botones. Actualiza el
valor de la variable global estado.
    uint8_t flag = P4IV; //guardamos el vector de interrupciones. De
paso, al acceder a este vector, se limpia automaticamente.
    P4IE &= 0x5D;   //interrupciones Joystick en port 4 desactivadas
    estado_anterior=0;
    switch(flag){
            case 0x04:
                estado = Jstick_Center;
                break;
            case 0x0C:
                estado = Jstick_Right;
                break;
            case 0x10:
                estado = Jstick_Left;
                break;
        }//Guardamos el estado de la interrupcion

    /*********************************************************+
        A RELLENAR POR EL ALUMNO BLOQUE switch ... case
    Para gestionar los estados:
    Boton S1, estado = 1
    Boton S2, estado = 2
    Joystick left, estado = 3
    Joystick right, estado = 4
    Joystick up, estado = 5
    Joystick down, estado = 6
    Joystick center, estado = 7
    *********************************************************/



    /*********************************************
     * HASTA AQUI BLOQUE CASE
```

```
     ********************************************/

    P4IE |= 0xA2;   //interrupciones Joystick en port 4 reactivadas
}

//ISR para las interrupciones del puerto 5:
void PORT5_IRQHandler(void){  //interrupción de los botones. Actualiza el
valor de la variable global estado.
    uint8_t flag = P5IV; //guardamos el vector de interrupciones. De
paso, al acceder a este vector, se limpia automaticamente.
    P5IE &= 0xCD;   //interrupciones Joystick y S1 en port 5 desactivadas
    estado_anterior=0;
    switch(flag){
            case 0x04:
                estado = Button_S1;
                break;
            case 0x0C:
                estado = Jstick_Down;
                break;
            case 0x0A:
                estado = Jstick_Up;
                break;
            }//Guardamos el estado de la interrupcion

    /*********************************************************+
        A RELLENAR POR EL ALUMNO BLOQUE switch ... case
    Para gestionar los estados:
    Boton S1, estado = 1
    Boton S2, estado = 2
    Joystick left, estado = 3
    Joystick right, estado = 4
    Joystick up, estado = 5
    Joystick down, estado = 6
    Joystick center, estado = 7
    *********************************************************/



    /*******************************************
     * HASTA AQUI BLOQUE CASE
     *******************************************/

    P5IE |= 0x32;   //interrupciones Joystick y S1 en port 5 reactivadas
}



void TA0_0_IRQHandler(void){
    uint16_t flag = TA0IV; // Guardamos el vector de interrupciones del
TA0 y se limpia automaticamente
    TA0CCTL0 &= ~CCIE; // Desactivamos temporalmente la interrupciones
del TA0
    temps++; // Aumentamos el contador del tiempo
```

```
    TA0CCTL0 &= ~CCIFG; // Hem de netejar el flag de la interrupció
    TA0CCTL0 |= CCIE; // S'ha d'habilitar la interrupció abans de sortir
}

void TA1_0_IRQHandler(void){
    uint16_t flag = TA1IV; // Guardamos el vector de interrupciones del
TA1 y se limpia automaticamente
    TA1CCTL0 &= ~CCIE; // Desactivamos temporalmente la interrupciones
del TA1
    temps2++; // Aumentamos el contador del tiempo
    TA1CCTL0 &= ~CCIFG; //Hem de netejar el flag de la interrupció
    TA1CCTL0 |= CCIE; //S'ha d'habilitar la interrupció abans de sortir

}
```

3. Diagrama de Flujo