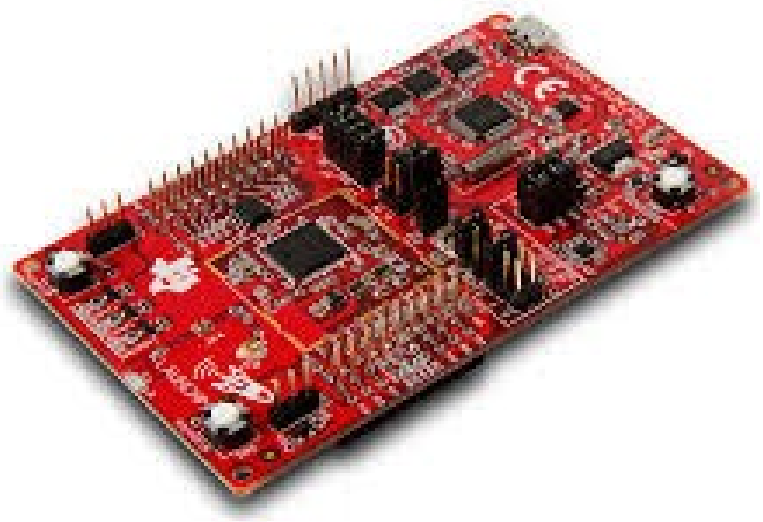
Programació d'Arquitectures Encastades

Pràctica 2 – Configuració De Ports

Semestre de Primavera 2023



David Fernàndez
Víctor Sort

David Fernàndez
Víctor Sort

ÍNDEX

David Fernàndez
Víctor Sort

1. OBJECTIUS

L'objectiu d'aquesta pràctica consisteix, per una part, a aprendre a configurar i a controlar els ports d'Entrada/Sortida de propòsit general (o GPIOs) del microcontrolador MSP432P401R usat a l'assignatura. Per altra part, aprendre a utilitzar en llenguatge C diverses instruccions de programació per controlar el flux del programa i fer salts condicionals i bucles.

Per això, després de seguir un guió on se'ns expliquen uns quants conceptes referents a l'anterior temari esmentat, se'ns proposa el següent exercici. Hem de programar un codi que permeti que un LED vermell pampalluguegi constantment i també permeti controlar tres LEDs RGB mitjançant 2 polsadors de la següent manera:

- Si premem el primer polsador un nombre imparell de cops, s'han d'encendre els 3 LEDs RGB, i si es prem un nombre parell de cops, s'inverteix l'estat d'ells.

- Si premem el segon polsador, l'estat antic del LED vermell passa a ser el del verd, el del verd passa a ser el del blau, i el vermell s'apaga. Però, si els tres LEDs RGB estaven apagats, simplement s'encén el vermell.

David Fernàndez
Víctor Sort

2. RECURSOS UTILITZATS

En aquesta pràctica, no s'han fet servir ni recursos de la placa d'experimentació ni del robot, només del microcontrolador. En concret, s'han utilitzat els 2 polsadors (S1 i S2), un LED vermell (LED1) i 3 LEDs RGB (LED2_RED, LED2_GREEN, LED2_BLUE). Els ports del microcontrolador als quals estàn connectats es mostren a la següent taula:

Recurs	Port.pin	Recurs	Port.pin
S1	P1.1	LED2_RED	P2.0
S2	P1.4	LED2_GREEN	P2.1
LED1	P1.0	LED2_BLUE	P2.2

Clarament, tots els LEDs són dispositius de sortida i els polsadors són dispositius d'entrada. Tots aquests dispositius funcionen tal com s'ha explicat en el punt anterior.

Com tots els dispositius són GPIOs, s'han de posar del port 1 i 2, dels pins anteriorment dits, SEL0 i SEL1 a 0, per això es fa servir una màscara amb el & i els bits que toquen negats. Això posa a 0 únicament aquests bits, sense tocar la resta:

```
P1SEL0 &= ~(BIT0 + BIT1 + BIT4);
P1SEL1 &= ~(BIT0 + BIT1 + BIT4);
P2SEL0 &= ~LEDS_RGB;    //LEDS_RGB està definit com (BIT0 + BIT1 + BIT2)
P2SEL1 &= ~LEDS_RGB;
```

Com els LEDs són dispositius de sortida, posem els bits corresponents a PXDIR a 0. Anàlogament, com els polsadors són d'entrada, els posem a 1.

```
P1DIR |= LED_V_BIT; //LED_V_BIT està definit com BIT 0
P2DIR |= LEDS_RGB;
P1DIR &= ~(SW1_BIT + SW2_BIT );  //SWX_BIT estan definits com BIT 1 i BIT 4
P1OUT &= ~LED_V_BIT;
P2OUT &= ~LEDS_RGB;
```

Les següents línies, serveixen per posar els polsadors com a pull-up, activar les interrupcions i netejar les anteriors.

```
P1REN |= (SW1_BIT + SW2_BIT );
P1OUT |= (SW1_BIT + SW2_BIT );
P1IE |= (SW1_BIT + SW2_BIT );
P1IES &= ~(SW1_BIT + SW2_BIT );
P1IFG = 0;
```

David Fernàndez
Víctor Sort

3. PROBLEMES

Els 2 principals problemes amb els quals ens hem trobat han tingut a veure amb l'estat 3.

El primer, i que és un problema més pràctic, ha sigut la "dificultat", en el cas que no tots els LEDs RGB estiguin apagats, en aconseguir passar de R → 0, G → R i B → G sense modificar els altres 5 bits de P2OUT.

Finalment hem aconseguit solucionar el problema utilitzant dues variables auxiliars i diverses màscares de la següent manera:

Enumerem el valor dels 8 bits de P2OUT (que poden ser, clarament, 0 o 1) com «7» «6» «5» «4» «3» «B» «G» «R». El que volem és aconseguir que passin a ser «7» «6» «5» «4» «3» «G» «R» 0.

```
operacio = P2OUT <<1;                 //operacio guarda «6» «5» «4» «3» «B» «G» «R» 0
operacio &= LEDS_RGB;                 //operacio guarda 0 0 0 0 0 «G» «R» 0
operacio2 = P2OUT & ~LEDS_RGB;         //operacio2 guarda «7» «6» «5» «4» «3» 0 0 0
P2OUT = operacio + operacio2;         //P2OUT passa a ser «7» «6» «5» «4» «3» «G» «R» 0
```

La segona dificultat que hem tingut, ha sigut la capacitat de poder repetir l'estat 3. És a dir, suposadament, cada cop que es premia el segon polsador s'havia de tornar a repetir el mateix canvi en els LEDs que hem esmentat abans. Però, hi havia una línia de codi que ja ens venia donada:

```
if (estado_anterior != estado)
```

Això comportava que si estàvem en l'estat 3 i es tornava a prendre el segon polsador, com l'estat tornava a ser el 3 i no hi havia canvi, no es produïa un nou canvi en els LEDs.

La solució que hem trobat ha consistit en un cop acabat de fer els canvis en els LEDs RGB, assignar tant a la variable "estado" com a la variable "estado_anterior" el valor «0», que no es correspon en cap estat definit, però ens permet que una nova premuda consecutiva del segon polsador posi la variable "estado" a "3", i consecuentment, com es compleix la condició escrita anteriorment, entri al switch per fer un nou canvi en els valors dels LEDs RGB.

Per finalitzar aquesta secció, volem comentar que no hem tingut en compte, ja que no s'especifica, que els cops que cal prémer el primer polsador siguin consecutius per veure la paritat. Per exemple, si estem a l'estat 1 i prenem el segon polsador passant a l'estat 3, quan tornem a prémer el primer polsador no tornarem a l'estat 1, sinó que passarem a l'estat 2.

David Fernàndez
Víctor Sort

4. CONCLUSIONS

Per començar, estem molt satisfets amb el nostre codi final, ja que satisfà l'objectiu de la pràctica en tots els casos possibles. El led vermell pampallugueja tota l'estona i els tres LEDs RGB van variant el seu estat d'acord amb l'especificat a l'apartat d'objectius. Hem comprovat, a més, que és possible crear els 8 colors possibles que generen els 3 LEDs RGB, jugant amb els dos polsadors.

Aquesta pràctica, que personalment ens ha suposat menys feina de programació de la qual ens esperàvem inicialment (a diferència del temps invertit en la redacció d'aquest informe), ens ha ajudat a entendre els ports GPIOs, les màscares i algunes instruccions, al veure com poden ser aplicades a situacions pràctiques.

Finalment, creiem que hem treballat molt bé en equip, que ens hem ajudat mútuament i esforçat per aprendre, anar més enllà del demanat i per optimitzar al màxim, dins les nostres capacitats, el codi a entregar. En resum, i en la nostra opinió, hem fet una molt bona pràctica.

David Fernàndez
Víctor Sort

5. ANNEXOS

5.1 Programa Comentat

```
/*****************************
 * Practica_02_PAE Programació de Ports
 * i pràctica de les instruccions de control de flux:
 * "do ... while", "switch ... case", "if" i "for"
 * UB, 02/2021.
 *****************************/

//Incloem les llibreries necessàries pel funcionament del codi
#include <msp432p401r.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>

//Definim etiquetes que guarden els bits del Port2 corresponents als leds, els
bits del Port1 corresponents als dos polsadors, també les IHR i el retard
empleat al codi
#define LED_V_BIT BIT0
#define LEDS_RGB (BIT0 + BIT1 + BIT2)
#define SW1_POS 1
#define SW2_POS 4
#define SW1_INT 0x04
#define SW2_INT 0x0A
#define SW1_BIT BIT(SW1_POS)
#define SW2_BIT BIT(SW2_POS)
#define RETRASO 300000

//Definim una variable global que guardarà l'estat en el que es troba el
microcontrolador i li assignem l'estat inicial 0, que no fa res.
volatile uint8_t estado = 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 1 es la User ISR numero 35.
    // 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 1 = 35 corresponde al bit 3 del segundo registro ISER1:
```

```
    NVIC->ICPR[1] |= BIT3; //Primero, me aseguro de que no quede ninguna
interrupcion residual pendiente para este puerto,
    NVIC->ISER[1] |= BIT3; //y habilito las interrupciones del puerto

}

/*************************************************************************
 * INICIALIZACIÓN DE LOS BOTONES & LEDS DEL BOOSTERPACK MK II.
 *
 * Sin datos de entrada
 *
 * Sin datos de salida
 *
 *************************************************************************/
void init_botons(void)
{
    //Configuramos botones i LED vermell
    //***************************
    P1SEL0 &= ~(BIT0 + BIT1 + BIT4 );    //Els polsadors i el led son GPIOs
    P1SEL1 &= ~(BIT0 + BIT1 + BIT4 );    //Els polsadors i el led son GPIOs

    //LED vermell = P1.0
    P1DIR |= LED_V_BIT;      //El LED es una sortida
    P1OUT &= ~LED_V_BIT;     //El estat inicial del LED es apagat

    //Botó S1 = P1.1 i S2 = P1.4
    P1DIR &= ~(SW1_BIT + SW2_BIT );    //Un polsador es una entrada
    P1REN |= (SW1_BIT + SW2_BIT );     //Pull-up/pull-down pel pulsador
    P1OUT |= (SW1_BIT + SW2_BIT ); //Donat que l'altra costat es GND, volem una
pull-up
    P1IE |= (SW1_BIT + SW2_BIT );      //Interrupcions activades
    P1IES &= ~(SW1_BIT + SW2_BIT );    // amb transicio L->H
    P1IFG = 0;                   // Netegem les interrupcions anteriors
    config_RGB_LEDS(); //Cridem al mètode per inicialitzar els LEDs RGB
}

/*************************************************************************
 * DELAY 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 temps)
{
    volatile uint32_t i;

    //El delay es basa en un bucle buit que es repeteix tants cops com es
vulgui, ja que per molt ràpid que s'executi, triga un cert temps en fer cada
iteració
    for(i = 0; i< temps; i++){
    }
}

/*************************************************************************
 * CONFIGURACIÓN DE LOS LEDs DEL PUERTO 2. A REALIZAR POR EL ALUMNO
```

```
 *
 * Sin datos de entrada
 *
 * Sin datos de salida
 *
 ******************************************************************************/
void config_RGB_LEDS(void)
{

    P2SEL0 &= ~LEDS_RGB;        //Els LEDs RGB són GPIOs
    P2SEL1 &= ~LEDS_RGB;        //Els LEDs RGB són GPIOs

    P2DIR |= LEDS_RGB;  //Són GPIOs de sortida
    P2OUT &= ~LEDS_RGB; //Els LEDS tenen estat inicial apagat

}

void main(void)
{
    //Guardem la variable estat anterior per poder saber quan canviem d'estat,
operació i operacio2 serveixen
    uint8_t estado_anterior = 0, operacio, operacio2;

    WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer

    //Inicializaciones:
    init_botons();         //Configuramos botones y leds

    init_interrupciones(); //Configurar y activar las interrupciones de los
botones

    __enable_interrupts();


    //Bucle principal (infinito):
    while (true)
    {

        if (estado_anterior != estado)  // Dependiendo del valor del estado se
encenderá un LED u otro.
        {
            estado_anterior = estado;   //Actualitzem estado_anterior pel pròxim
cop que entrem al bucle
            //La variable estado té el cas en el qual estem actualment
            switch (estado){
                //Encenem els LEDs RGB
                case 1:
                    P2OUT |= LEDS_RGB;
                    break;
                //Invertim els LEDs RGB
                case 2:
                    P2OUT ^= LEDS_RGB;
                    break;
                case 3:
                    //Si els tres LEDs RGB estan apagats, encenem el LED R (2.0)
                    if ((P2OUT & LEDS_RGB) == 0x00){
                        P2OUT |= BIT0;
                    }
```

```
                    //Considerem que P2OUT = <7><6><5><4><3><B><G><R>
                    else{
                        //Ens guardem a operacio P2OUT desplaçat a l'esquerra un
bit
                        operacio = P2OUT << 1;  //operacio = <6><5><4><B><G><R>0
                        //Li apliquem la màscara per quedar-nos amb 0000<G><R>0
                        operacio &= LEDS_RGB;
                        //Ara apliquem una segona màscara per quedar-nos amb
<7><6><5><4><3>000
                        operacio2 = P2OUT & ~LEDS_RGB;
                        //Aconseguim que P2OUT sigui <7><6><5><4><3><G><R>0
                        P2OUT = operacio + operacio2;
                    }
                    //Assignem l'estat neutre 0 a estado i estado_anterior
perquè no entri al if inicial i si tornem a prémer el polsador 2, que torni a
entrar
                    estado_anterior = 0;
                    estado = 0;
                default:
                    break;

            }
        }

        P1OUT ^= LED_V_BIT;     // Conmutamos el estado del LED R
        delay_t(RETRASO);       // periodo del parpadeo
    }

}

/*************************************************************************
 * 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 1:
void PORT1_IRQHandler(void)
{
    static bool impar = false;
    uint8_t flag = P1IV; //guardamos el vector de interrupciones. De paso, al
acceder a este vector, se limpia automaticamente.
    P1IE &= ~(SW1_BIT + SW2_BIT ); //interrupciones del boton S1 y S2 en port 1
desactivadas

    switch (flag)
    {
    //Si la interrupció ha estat causada pel polsador 1
    case SW1_INT:
        if (!impar){
            //Si havíem premut un nombre parell de cops el polsador 1, passarà a
senar i, per tant, estem en el cas 1
```

```
            estado = 1;
            impar = true;
        }
        else{
            //Si havíem premut un nombre senar de cops el polsador 1, passarà a
parell i, per tant, estem en el cas 2
            estado = 2;
            impar = false;
        }
        break;
    //Si la interrupció ha estat causada pel polsador 2, estem en el cas 3
    case SW2_INT:
        estado = 3;
        break;
    default:
        break;
    }

    P1IE |= (SW1_BIT + SW2_BIT );   //interrupciones S1 y S2 en port 1
reactivadas
}
```

David Fernàndez
Víctor Sort

5.2 Diagrama d'Estats

Estat 1: S'acaba de prémer el polsador 1, i en total s'ha premut un nombre imparell de vegades.

Estat 2: S'acaba de prémer el polsador 1, i en total s'ha premut un nombre parell de vegades.

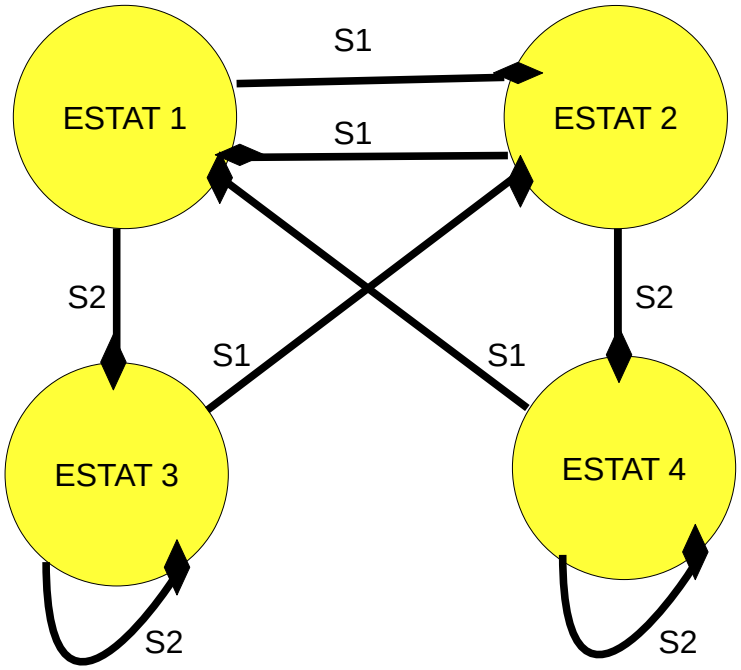
Estat 3: S'acaba de prémer el polsador 2, i en total el polsador 1 s'ha premut un nombre imparell de vegades.

Estat 4: S'acaba de prémer el polsador 2, i en total el polsador 1 s'ha premut un nombre parell de vegades.

(La diferència entre el tercer i quart estat no existeix com a tal en la implementació, però si que és important considerar-la per fer el diagrama d'estats.)

S1: Es prem el polsador 1.

S2: Es prem el polsador 2.



David Fernàndez
Víctor Sort

5.3 Diagrames de Flux

Diagrama de flux del main:

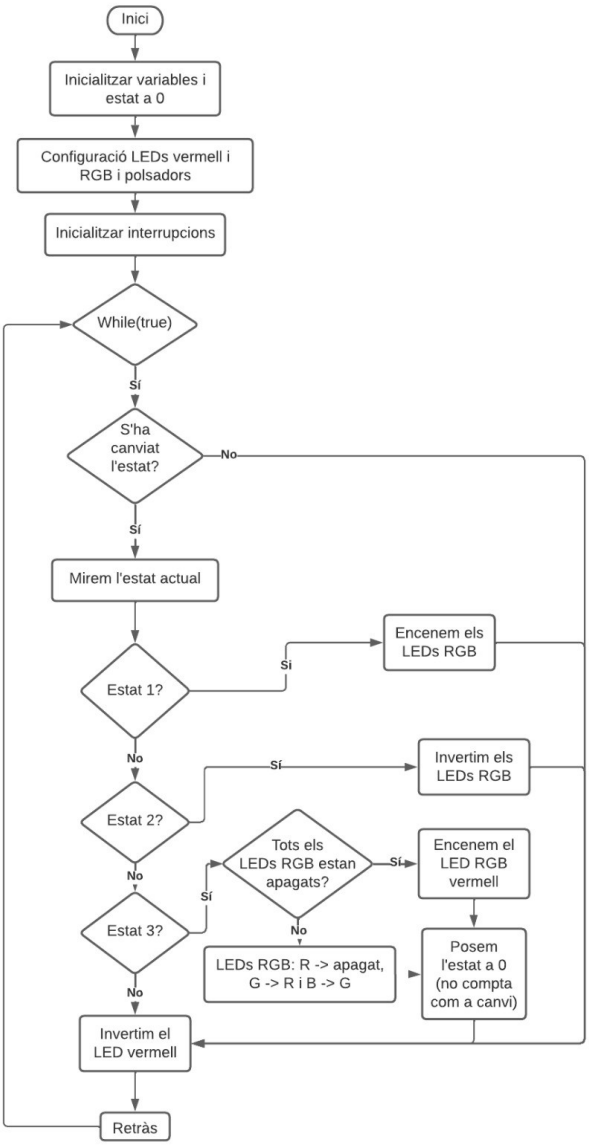


Diagrama de flux de les interrupcions: