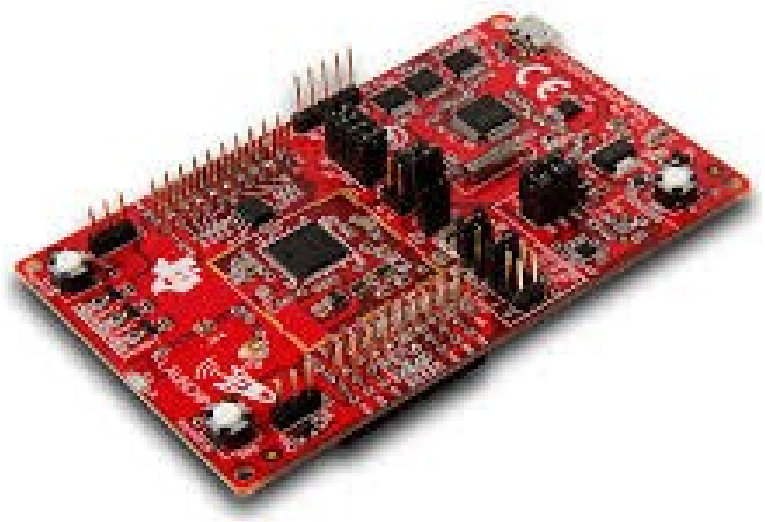
Programació d'Arquitectures Encastades

Pràctica 3 – Timers i interrupcions

Semestre de Primavera 2023



David Fernàndez
Víctor Sort

David Fernàndez
Víctor Sort

ÍNDEX

David Fernàndez
Víctor Sort

1. OBJECTIUS

L'objectiu principal d'aquesta pràctica consisteix, tal i com el seu nom indica, en aprendre a configurar i utilitzar timers i interrupcions (en tots els seus tres nivells) en el microcontrolador MSP432P401R usat a l'assignatura. També, al haver de crear un array d'estructures, es practiquen aquests conceptes.

A més a més, per resoldre el problema que se'ns planteja també s'han de configurar i controlar LEDs i altres recursos com els joysticks, utilitzant també doncs la placa del Boosterpack, a més de la placa Launchpad utilitzada a l'anterior pràctica. Per fer-ho, s'ha de treballar amb ports d'Entrada/Sortida de propòsit general (o GPIOs), així que s'ha de tornar a aplicar l'aprés a l'anterior pràctica.

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 controli un LED vermell a la placa Launchpad i tres altres LEDs RGB a la placa del Boosterpack de la següent manera:

1. Pel LED vermell de la placa Launchpad, es genera una base de temps de freqüència 10kHz, i es crea un codi perquè funcioni de la següent manera:

- Quan es produeixin "duty" interrupcions d'aquesta base de temps ("duty" sent un enter entre 0 i 100), el LED vermell s'apagarà.
- Quan el nombre d'interrupcions arribi a 100, el LED vermell s'encendrà i el comptador de les interrupcions tornarà a 0.

A més, també es configuren els joysticks de la placa del Boosterpack perquè actuïn de la següent manera:

- Joystick amunt: increment de "duty" en una quantitat "step" .
- Joystick avall: decrement de "duty" en una quantitat "step" .
- Joystick esquerra: la quantitat "duty" és veu decrementada en 5.
- Joystick dreta: la quantitat "duty" és veu incrementada en 5.

2. Els 3 LEDs RGB de la placa del Boosterpack seran configurats per anar seguint de manera cíclica la següent seqüència de colors:

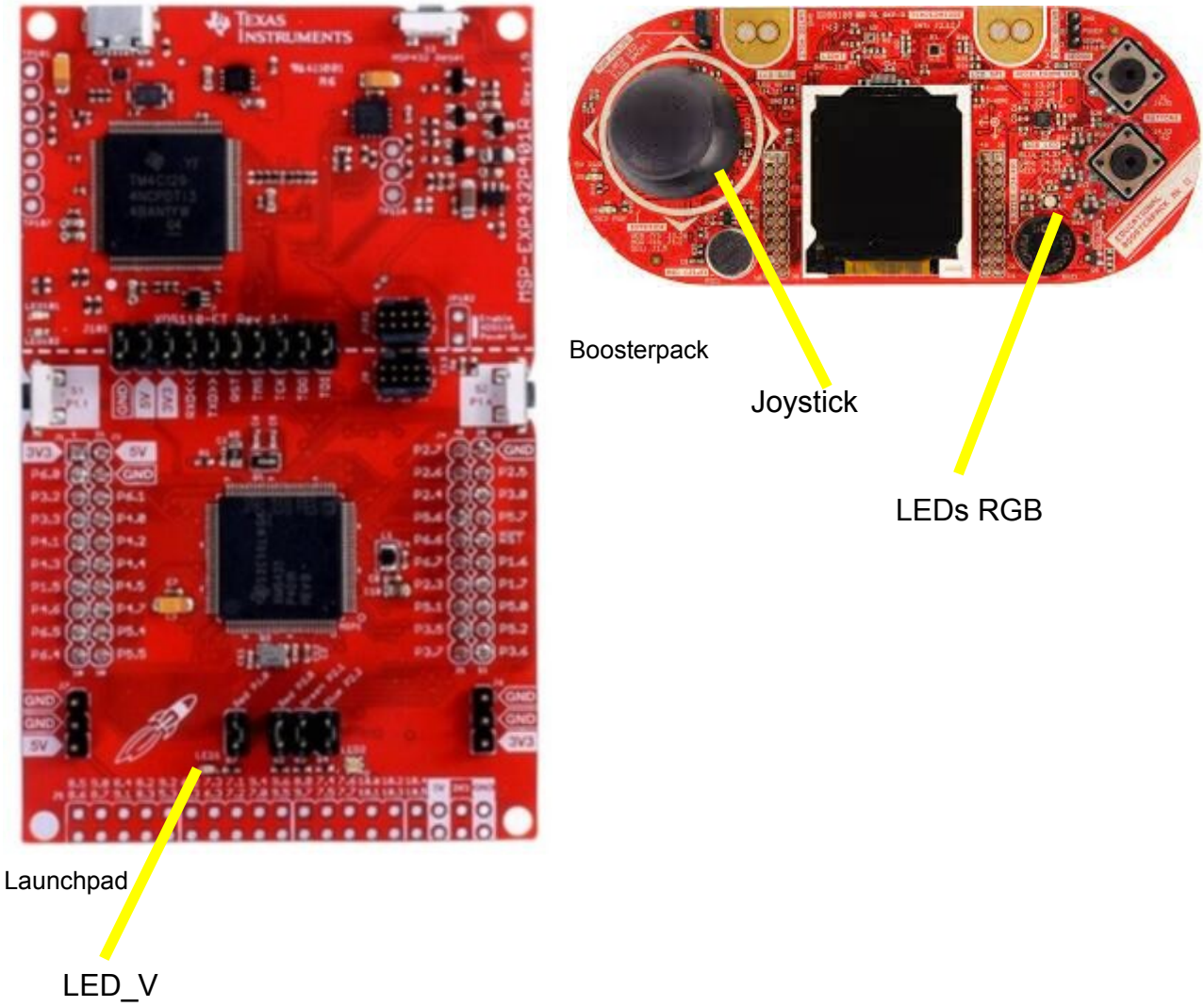
Color	Vermell	Groc	Verd	Blau	Blanc
Temps	2 segons	1 segon	3 segons	2 segons	1 segon

David Fernàndez
Víctor Sort

2. RECURSOS UTILITZATS

En aquesta pràctica, s'ha fet servir un únic recurs de la placa Launchpad. En concret, el LED vermell (LED_V). No s'han utilitzat ni els polsadors ni els LEDs RGB d'aquesta placa, a diferència de l'anterior pràctica.

De la placa d'experimentació (boosterpack) del robot s'han utilitzat els 3 LEDs RGB (LED_RGB_R, LED_RGB_G, LED_RGB_B) i el joystick (JOY_AMUNT, JOY_ABAIX, JOY_DRETA, JOY_ESQUERRA), tots menys el joystick central.



David Fernàndez
Víctor Sort

Els ports del microcontrolador als quals estan connectats tots els anteriors components utilitzats es mostren a la següent taula:

Recurs	Port.pin	Recurs	Port.pin
LED_V	P1.0	JOY_AMUNT	P5.4
LED_RGB_R	P2.6	JOY_AVALL	P5.5
LED_RGB_G	P2.4	JOY_DRETA	P4.5
LED_RGB_B	P5.6	JOY_ESQUERRA	P4.7

Tots els LEDs són díodes que es controlen mitjançant les sortides del microcontrolador, mentre que el joystick proporciona una entrada. Tots aquests components s'han programat perquè funcionin tal com s'ha explicat en el punt anterior.

Explico a continuació com s'han configurat els components. Per no fer pesat el punt, només faré el cas dels LED_RGB_R i LED_RGB_G (ja que la resta de LEDs són anàlogs) i JOY_DRETA i JOY_ESQUERRA (ja que la resta d'opcions del joystick són anàlogues). Igualment, es pot veure detalladament tots els casos en l'annex 5.1. El mètode on es fa la configuració és init_joysticks_LEDS().

Inicialment, definim a quins bits correspón cada component, per facilitar la futura comprensió.

```
#define LED_RGB_R BIT6
#define LED_RGB_G BIT4
#define JOY_DRETA BIT5
#define JOY_ESQUERRA BIT7
```

Com tots els dispositius s'han de configurar com a GPIOs, s'han de posar del port 2 i 4, 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:

```
P2SEL0 &= ~(LED_RGB_R + LED_RGB_G);
P2SEL1 &= ~(LED_RGB_R + LED_RGB_G);
P4SEL0 &= ~(JOY_DRETA + JOY_ESQUERRA );
P4SEL1 &= ~(JOY_DRETA + JOY_ESQUERRA);
```

Com els LEDs són dispositius de sortida, posem els bits corresponents a PXDIR a 1, utilitzant una altra màscara. Anàlogament, com els joysticks són d'entrada, els posem a 0. A més, com l'estat inicial dels LEDS és apagat, posem P2OUT dels bits també a 0.

```
P2DIR |= (LED_RGB_R + LED_RGB_G); //Són GPIOs de sortida
P2OUT &= ~(LED_RGB_R + LED_RGB_G); //Els LEDS tenen estat inicial apagat
P4DIR &= ~(JOY_DRETA + JOY_ESQUERRA); //Un joystick es una entrada
```

David Fernàndez
Víctor Sort

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

```
P4REN |= (JOY_DRETA + JOY_ESQUERRA);
P4OUT |= (JOY_DRETA + JOY_ESQUERRA);
P4IE |= (JOY_DRETA + JOY_ESQUERRA);
P4IES &= ~(JOY_DRETA + JOY_ESQUERRA);
P4IFG = 0;
```

David Fernàndez
Víctor Sort

3. PROBLEMES

Realitzant aquesta pràctica, hem tingut 2 principals problemes que ens han suposat bastant temps a solucionar, o bé decidir la solució més òptima.

El primer problema ens va comportar "perdre" gairebé tota una classe de laboratori, doncs trobar l'error que teníem no va ser fàcil. La primera part de la pràctica no ens funcionava i al fer un debug del codi vam veure que es quedava atrapat en bucle a la rutina d'interrupció del timer0.

El motiu, la següent línia al final:

```
TA1CCTL0 |= TIMER_A_CCTLN_CCIFG; //Reactivate interrupt flag
```

Aquesta línia posava a 1 un altre cop el flag, fent que es tornés a detectar la interrupció, i així es cridés un altre cop la rutina d'interrupció, creant un bucle infinit. Vem solucionar-ho borrant l'anterior línia i afegint la següent:

```
TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0
```

El temps que vam invertir en detectar l'error ens va servir però per entendre molt bé com funcionen exactament les interrupcions.

El segon problema l'hem tingut a l'hora de gestionar la seqüència dels LEDs RGB, ja que de la manera en com havíem programat el codi (fent les permutacions únicament en el mètode de gestió de la interrupció), o bé el primer segon de vermell no es feia o bé els 3 LEDs RGB estaven apagats un segon abans de començar la seqüència.

La nostra primera solució va consistir en configurar la sortida dels 3 LEDs RGB perquè creessin vermell (només encés el LED_R) des del mètode init_joystick_LEDS(). Tot i que això solucionava el problema, creiem que no és una bona solució ja que si es volgués canviar, seria difícil per un futur programador d'entendre.

La següent solució va consistir en crear un mètode update_RGB() i una variable booleana "canvi" declarada a true. Des de la rutina d'interrupció del timer1, quan detectés que ja és moment de canviar el color que produeixen els LEDs RGB, posaria "canvi" a true. El bucle infinit cridaria al mètode update_RGB() constantment i aquest comprovaria en tot moment si "canvi" és true. En aquest cas, el posaria a false i executaria el canvi de colors. Si bé també funcionava, no creiem que fos una solució gaire òptima, ja que es feien infinites crides al mètode update_RGB().

Finalment vam decidir eliminar la variable "canvi". Així doncs, abans del bucle infinit cridem un cop al mètode update_RGB() (per solucionar el problema inicial que ens porta fins aquest punt) i el cridem també dintre de la rutina d'interrupció del timer1, quan aquest ho trobi convenient (quan ja sigui moment de canviar de color), en lloc del bucle infinit.

David Fernàndez
Víctor Sort

4. CONCLUSIONS

Per començar, estem molts satisfets amb el nostre codi final, ja que satisfà l'objectiu de la pràctica en tots els casos possibles:

El led vermell de la placa del Launchpad està tota l'estona encés, variant però la seva intensitat depenent d'on es trobi el "duty". Si "duty" és proper a 100, el LED es veu molt il·luminat, i decreix la seva lluminositat a mesura que "duty" baixa a 0. Això succeeix perquè al ser la base de temps seleccionada tan petita, l'ull humà no percep els canvis i percebem una lluminositat proporcional al temps que el LED està encés o apagat. També, els LEDs RGB de la placa del Boosterpack s'il·luminen iterant el patró especificat en els objectius correctament.

A diferència de la primera pràctica a entregar, al no haver avançat part del codi a casa prèviament a les classes de laboratori, hem anat una mica justos de temps. A més a més, el temps que vàrem dedicar a classe a solucionar el primer problema esmentat al punt anterior va fer que ens quedéssim pràcticament sense temps de classe per aprofitar.

Aquesta pràctica també ens ha ajudat a entendre com utilitzar timers, clocks (i les matemàtiques que tenen darrere) i com configurar les interrupcions, al veure com poden ser aplicades a situacions pràctiques. També hem repassat conceptes que vam veure en anterioritat.

Finalment, creiem que hem treballat molt bé en equip, que ens hem repartit les tasques equitativament, ajudat mútuament i esforçat per aprendre i 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

```
//David Fernández i Víctor Sort
/**********
 * Practica_03_PAE Timers i interrupcions
 * UB, 03/2021.
 ***********/

#include <msp432p401r.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <lib_PAE.h>

#define LED_V_BIT BIT0
#define LED_RGB_R BIT6
#define LED_RGB_G BIT4
#define LED_RGB_B BIT6
#define JOY_DRETA BIT5
#define JOY_ESQUERRA BIT7
#define JOY_AMUNT BIT4
#define JOY_AVALL BIT5
#define CNT_MAX 100

typedef struct
{
    bool r, g, b;
    uint8_t time;
} color_t;

volatile color_t color_sequence[] = { { .r = true, .g = false, .b = false, .time
= 2 },
                              { .r = true, .g = true, .b = false, .time = 1 },
                              { .r = false, .g = true, .b = false, .time = 3 },
                              { .r = false, .g = false, .b = true, .time = 2 },
                              { .r = true, .g = true, .b = true, .time = 1 } };

//guardem la longitud de l'array per saber quan tornar a la posició 0
volatile uint8_t mida_color_sequence =
sizeof(color_sequence)/sizeof(color_sequence[0]);
//quan comencem suposem que volem començar amb el color_sequence 0.
volatile int8_t index = 0;
//valor del comptador en el qual s'apaga el led vermell
volatile int8_t pwm_duty = 50;
//step per incrementar o decrementar el pwm_duty
```

```
volatile int8_t pwm_step = 20;

/*************************
 * 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] |= 1 << (PORT1_IRQn & 31); //Primero, me aseguro de que no
quede ninguna interrupcion residual pendiente para este puerto,
    //NVIC->ISER[1] |= 1 << (PORT1_IRQn & 31); //y habilito las interrupciones
del puerto
    //Int. port 4 = 38 corresponde al bit 6 del segundo registro ISER1:
    NVIC->ICPR[1] |= 1 << (PORT4_IRQn & 31); //Primero, me aseguro de que no
quede ninguna interrupcion residual pendiente para este puerto,
    NVIC->ISER[1] |= 1 << (PORT4_IRQn & 31); //y habilito las interrupciones del
puerto
    //Int. port 5 = 39 corresponde al bit 7 del segundo registro ISER1:
    NVIC->ICPR[1] |= 1 << (PORT5_IRQn & 31); //Primero, me aseguro de que no
quede ninguna interrupcion residual pendiente para este puerto,
    NVIC->ISER[1] |= 1 << (PORT5_IRQn & 31); //y habilito las interrupciones del
puert
    //Int. port TA0_0 = 8 corresponde al bit 8 del primer registro ISER0:
    NVIC->ICPR[0] |= 1 << (TA0_0_IRQn); //Primero, me aseguro de que no quede
ninguna interrupcion residual pendiente para este puerto,
    NVIC->ISER[0] |= 1 << (TA0_0_IRQn); //y habilito las interrupciones del
puerto
    //Int. port TA1_0 = 8 corresponde al bit 10 del primer registro ISER0:
    NVIC->ICPR[0] |= 1 << (TA1_0_IRQn); //Primero, me aseguro de que no quede
ninguna interrupcion residual pendiente para este puerto,
    NVIC->ISER[0] |= 1 << (TA1_0_IRQn); //y habilito las interrupciones del
puerto
}
```

David Fernàndez
Víctor Sort

```
/*************************
 * INICIALIZACIÓN DE LOS BOTONES & LEDS DEL BOOSTERPACK MK II.
 * Sin datos de entrada
 * Sin datos de salida
 *************************/
void init_joysticks_LEDS(void){
    //LED vermell = P1.0
    P1SEL0 &= ~(LED_V_BIT);    //El LED es configura com GPIO
    P1SEL1 &= ~(LED_V_BIT);    //El LED es configura com GPIO
    P1DIR |= LED_V_BIT;      //El LED es una sortida
    P1OUT &= ~LED_V_BIT;     //El estat inicial del LED es apagat

    //LEDs RGB = P2.6, P2.4, P5.6
    P2SEL0 &= ~(LED_RGB_R + LED_RGB_G); //Els LEDs RGB es configuren com  GPIOs
    P2SEL1 &= ~(LED_RGB_R + LED_RGB_G); //Els LEDs RGB es configuren com  GPIOs
    P5SEL0 &= ~LED_RGB_B; //Els LEDs RGB es configuren com  GPIOs
    P5SEL1 &= ~LED_RGB_B; //Els LEDs RGB es configuren com  GPIOs
    P2DIR |= (LED_RGB_R + LED_RGB_G); //Són GPIOs de sortida
    P2OUT &= ~(LED_RGB_R + LED_RGB_G); //Els LEDS tenen estat inicial apagat
    P5DIR |= LED_RGB_B; //Són GPIOs de sortida
    P5OUT &= ~LED_RGB_B; //Els LEDS tenen estat inicial apagat

    //Joystick = P4.5, P4.7, P5.4, P5.5
    P4SEL0 &= ~(JOY_DRETA + JOY_ESQUERRA );    //El joystick es configura com
GPIO
    P4SEL1 &= ~(JOY_DRETA + JOY_ESQUERRA);    //El joystick es configura com
GPIO
    P5SEL0 &= ~(JOY_AMUNT + JOY_AVALL);    //El joystick es configura com GPIO
    P5SEL1 &= ~(JOY_AMUNT + JOY_AVALL );    //El joystick es configura com GPIO

    P4DIR &= ~(JOY_DRETA + JOY_ESQUERRA); //Un joystick es una entrada
    P4REN |= (JOY_DRETA + JOY_ESQUERRA); //Pull-up/pull-down pel joystick
    P4OUT |= (JOY_DRETA + JOY_ESQUERRA); //Donat que l'altre costat es GND,
volem una pull-up
    P4IE |= (JOY_DRETA + JOY_ESQUERRA); //Interrupcions activades
    P4IES &= ~(JOY_DRETA + JOY_ESQUERRA); // amb transicio L->H
    P4IFG = 0; // Netegem les interrupcions anteriors


    P5DIR &= ~(JOY_AMUNT + JOY_AVALL); //Un joystick es una entrada
    P5REN |= (JOY_AMUNT + JOY_AVALL); //Pull-up/pull-down pel joystick
    P5OUT |= (JOY_AMUNT + JOY_AVALL); //Donat que l'altre costat es GND, volem
una pull-up
    P5IE |= (JOY_AMUNT + JOY_AVALL); //Interrupcions activades
    P5IES &= ~(JOY_AMUNT + JOY_AVALL); // amb transicio L->H
    P5IFG = 0; // Netegem les interrupcions anteriors
```

```
}

void init_timers(void)
{
    //Timer A0, used for red LED PWM
    //Divider = 1; CLK source is SMCLK; clear the counter; MODE is up
    TIMER_A0->CTL = TIMER_A_CTL_ID_1 | TIMER_A_CTL_SSEL_SMCLK | TIMER_A_CTL_CLR
            | TIMER_A_CTL_MC__UP;
    TIMER_A0->CCR[0] = 2399;     // 10 kHz
    TIMER_A0->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0

    //Timer A1, used for RGB LEDs
    //Divider = 1; CLK source is ACLK; clear the counter; MODE is up
    TIMER_A1->CTL = TIMER_A_CTL_ID_1 | TIMER_A_CTL_SSEL_ACLK | TIMER_A_CTL_CLR
            | TIMER_A_CTL_MC__UP;
    TIMER_A1->CCR[0] = (1 << 15) - 1;     // 1 Hz
    TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0
}

void update_RGB(){
        //P2OUT & LED_RGBX serà true si el LED_RGBX és 1 a P2OUT i false si és 0
        //Si el led RGB vermell és encés i no ha d'estar-ho, l'apaguem
        if((P2OUT & LED_RGB_R) && !(color_sequence[index].r)){
            P2OUT &= ~LED_RGB_R;
        }
        //Si el led RGB vermell és apagat i no ha d'estar-ho, l'encenem
        else if(!(P2OUT & LED_RGB_R) && (color_sequence[index].r)){
            P2OUT |= LED_RGB_R;
        }
        //Si el led RGB verd és encés i no ha d'estar-ho, l'apaguem
        if((P2OUT & LED_RGB_G) && !(color_sequence[index].g)){
            P2OUT &= ~LED_RGB_G;
        }
        //Si el led RGB verd és apagat i no ha d'estar-ho, l'encenem
        else if(!(P2OUT & LED_RGB_G) && (color_sequence[index].g)){
            P2OUT |= LED_RGB_G;
        }
        //Si el led RGB blau és encés i no ha d'estar-ho, l'apaguem
        if((P5OUT & LED_RGB_B) && !(color_sequence[index].b)){
            P5OUT &= ~LED_RGB_B;
        }
        //Si el led RGB blau és apagat i no ha d'estar-ho, l'encenem
        else if(!(P5OUT & LED_RGB_B) && (color_sequence[index].b)){
            P5OUT |= LED_RGB_B;
        }
}
```

```
void main(void) {
    WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer

    //Inicializaciones:
    init_ucs_24MHz();

    //config_RGB_LEDS();
    init_interrupciones();          //Configurar y activar las interrupciones de
los joysticks y los timers
    init_timers();                  //Configuramos los timers
    init_joysticks_LEDS();         //Configuramos joysticks y leds
    __enable_interrupts();          //tercer nivel de interrupciones activado
    update_RGB();                   //posem els LEDS RGB amb el primer color de
la seqüència

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

void TA0_0_IRQHandler(void){
    static uint8_t cnt = 0;

    TA0CCTL0 &= ~TIMER_A_CCTLN_CCIFG; //Clear interrupt flag
    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIE; //Interrupciones desactivadas en
CCR0
    cnt++;
    //Si el comptador arriba al seu màxim, el posem a 0 i encenem el led vermell
    if(cnt == CNT_MAX){
        cnt = 0;
        P1OUT |= LED_V_BIT;
    }
    //Si el comptador arriba al pwm_duty, apaguem el led vermell
    else if(cnt == pwm_duty){
        P1OUT &= ~LED_V_BIT;
    }
    TIMER_A0->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0
}

void TA1_0_IRQHandler(void){ //permutar colors
    static uint8_t cnt = 0;

    TA1CCTL0 &= ~TIMER_A_CCTLN_CCIFG; //Clear interrupt flag
```

```
    TIMER_A1->CCTL[0] &= ~TIMER_A_CCTLN_CCIE; //Interrupciones desactivadas en
CCR0
    cnt++;
    //si el comptador arriba al màxim del time del color actual, vol dir que
l'haurem de canviar
    if(cnt == color_sequence[index].time){
        //posem el comptador a 0 el comptador
        cnt = 0;
        //augmentem l'index
        index++;
        //si l'index ja supera les posicions possibles de l'array
color_sequence, el posem a 0
        if(index == mida_color_sequence){
            index = 0;
        }
        update_RGB();
    }

    TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0
}
/**************************
 * RUTINAS DE GESTION DE LOS BOTONES:
 * Mediante estas rutinas, se detectará que 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 4:
void PORT4_IRQHandler(void){ //joystick
    uint8_t flag = P4IV; //guardamos el vector de interrupciones. De paso, al
acceder a este vector, se limpia automaticamente.
    P4IE &= ~(JOY_DRETA + JOY_ESQUERRA); //interrupciones del joystick en port 4
desactivadas
    switch (flag)
    {
    //Si la interrupció ha estat generada per moure el joystick a l'esquerra
    case 0x10:
        //decrementem en 5 el pwm_step
        pwm_step -= 5;
        //pwm_step com a mínim pot ser 0
        if (pwm_step < 0){
            pwm_step = 0;
        }
        break;
    //Si la interrupció ha estat generada per moure el joystick a la dreta
    case 0x0C:
```

```
        //incrementem en 5 el pwm_step
        pwm_step += 5;
        //pwm_step com a màxim pot ser CNT_MAX
        if (pwm_step > CNT_MAX){
            pwm_step = CNT_MAX;
        }
        break;
    default:
        break;
    }
    P4IE |= (JOY_DRETA + JOY_ESQUERRA);   //interrupciones del joystick en port
4 reactivadas
}
//ISR para las interrupciones del puerto 5:
void PORT5_IRQHandler(void){ //joystick
    uint8_t flag = P5IV; //guardamos el vector de interrupciones. De paso, al
acceder a este vector, se limpia automaticamente.
    P5IE &= ~(JOY_AMUNT + JOY_AVALL); //interrupciones del joystick en port 5
desactivadas

    switch (flag)
    {
    //Si la interrupció ha estat generada per moure el joystick amunt
    case 0x0A:
        //incrementem en pwm_step el pwm_duty
        pwm_duty += pwm_step;
        //pwm_duty com a màxim pot ser CNT_MAX
        if (pwm_duty > CNT_MAX){
            pwm_duty = CNT_MAX;
        }
        break;
    //Si la interrupció ha estat generada per moure el joystick avall
    case 0x0C:
        //decrementem en pwm_step el pwm_duty
        pwm_duty -= pwm_step;
        //pwm_duty com a mínim pot ser 0
        if (pwm_duty < 0){
            pwm_duty = 0;
        }
        break;
    default:
        break;
    }
    P5IE |= (JOY_AMUNT + JOY_AVALL);   //interrupciones del joystick en port 5
reactivadas
}
```

5.2 Diagrames de Flux

Diagrama de flux del main:

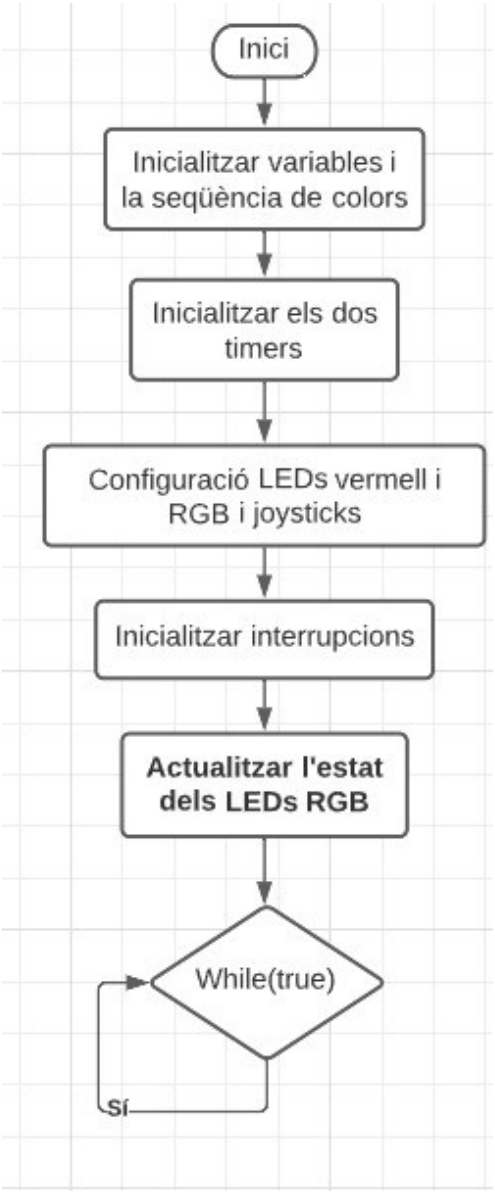
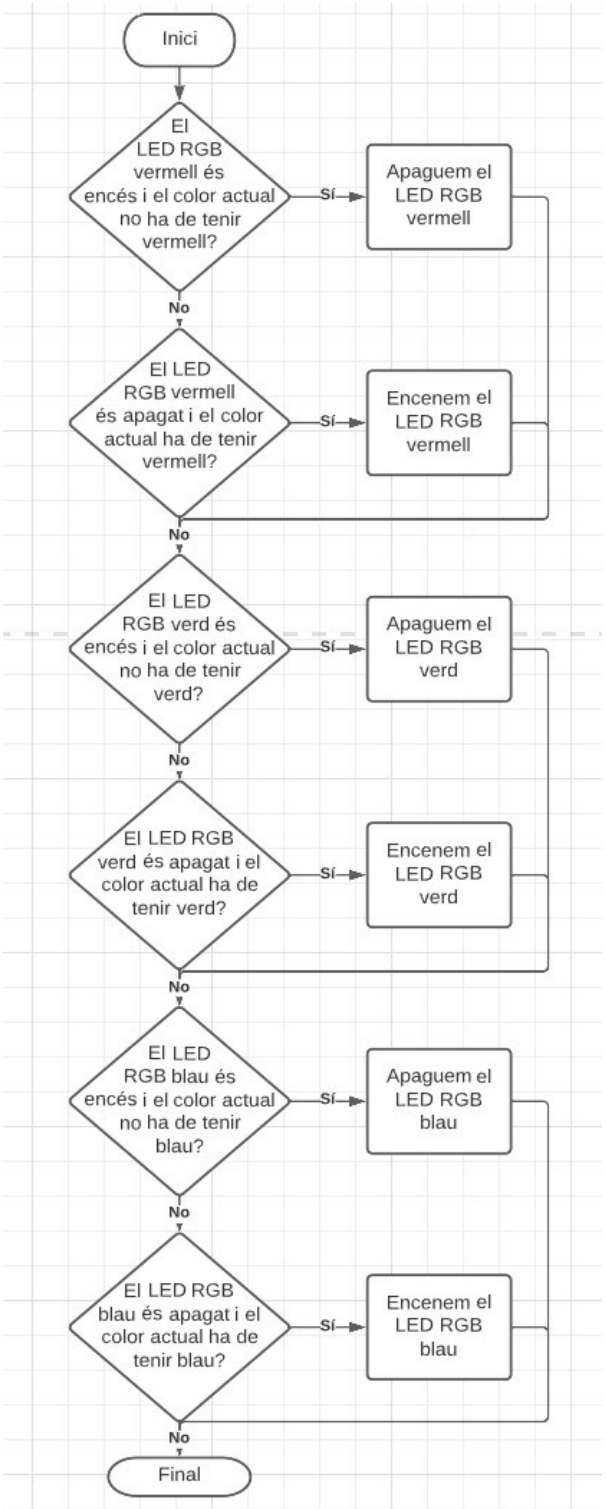
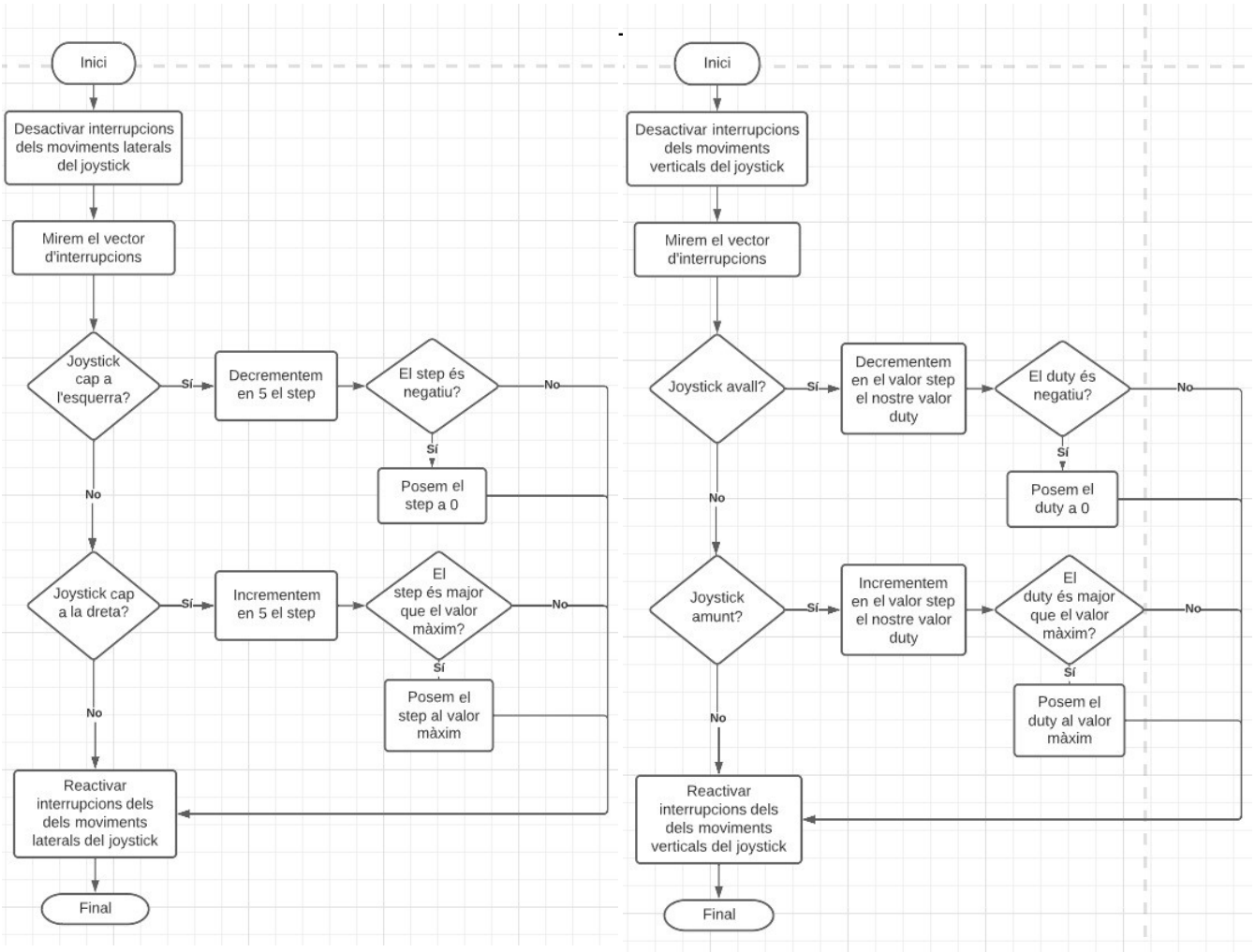


Diagrama de flux del mètode update_RGB():



David Fernàndez
Víctor Sort

Diagrames de flux de les rutines d'interrupció dels joysticks (referents al port 4, a l'esquerra, i al port 5, a la dreta):



David Fernàndez
Víctor Sort

Diagrames de flux de les rutines d'interrupció directes dels timers quan arriben al valor CCR0 (timer0, a la l'esquerra, i timer1, a la dreta):