



UNIVERSITAT<sup>DE</sup>  
BARCELONA

---

## Pràctica 3. Timers i Interrupcions

---

Noah Márquez Vara  
Alejandro Guzman Requena

29 març 2022

# ÍNDEX

<b>1</b>	<b>Introducció</b>	<b>3</b>
1	Què es vol fer a la pràctica . . . . .	3
2	Recursos utilitzats . . . . .	3
3	Configuració de recursos . . . . .	3
4	Funcions dels recursos . . . . .	7
5	Problemes . . . . .	8
5.1	1 <sup>a</sup> sessió . . . . .	8
5.2	2 <sup>a</sup> sessió . . . . .	8
6	Conclusions . . . . .	9
<b>2</b>	<b>Programa comentat</b>	<b>10</b>
<b>3</b>	<b>Diagrames de flux</b>	<b>17</b>

# 1 INTRODUCCIÓ

## 1 Què es vol fer a la pràctica

En aquesta pràctica l'objectiu principal es conèixer com es configuren i s'utilitzen correctament dos dels recursos més importants del microcontrolador: els timers i les interrupcions (tot i que ja vam fer servir les interrupcions en l'anterior pràctica). A més, per primer cop utilitzarem el *Boosterpack MKII* acoblat al robot enlloc de només a la placa *MSP-EXP432P401R*. Treballarem amb diverses instruccions de programació per tal de fer salts condicionals i bucles, a part de fer ús de màscares per facilitar la configuració dels pins utilitzats, tal i com es podrà veure a l'apartat 2.

Tractarem amb quatre ports del Boosterpack, concretament el port 1 (pel LED vermell), el port 2 (pels LEDs **RGB**) i els ports 4 i 5 (pels diversos moviments del *joystick*).

## 2 Recursos utilitzats

Primerament, del microcontrolador hem fet servir els següents recursos:

- Port 1: Concretament el pin 0 (**LED1**).
- Port 2: Concretament els pins 4 (**LED\_G** -> verd) i 6 (**LED\_R** -> vermell).
- Port 4: Concretament els pins 5 (*Joystick Dreta*) i 7 (*Joystick Esquerra*).
- Port 5: Concretament els pins 4 (*Joystick Amunt*), 5 (*Joystick Avall*) i 6 (**LED\_B** -> blau).

Els recursos utilitzats del Boosterpack es poden extreure del comentat fins ara, tot i així, s'indiquen a continuació:

- LED1 (LED vermell separat al que modificarem la intensitat lumínica)
- *Joystick Dreta*
- *Joystick Esquerra*
- *Joystick Amunt*
- *Joystick Avall*
- Els tres LEDs RGB (identificats com a LED\_X)

A continuació indiquem una taula resum dels recursos utilitzats amb els seus respectius ports i pins:

Recurs	Port.Pin	Recurs	Port.Pin
LED1	P1.0	<i>Joystick Esquerra</i>	P4.7
LED_G	P2.4	<i>Joystick Amunt</i>	P5.4
LED_R	P2.6	<i>Joystick Avall</i>	P5.5
<i>Joystick Dreta</i>	P4.5	LED_B	P5.6

Taula 1.1: Connexió dels recursos al Microcontrolador

## 3 Configuració de recursos

Tal i com hem vist a teoria, el primer que hem de dur a terme quan fem un programa per a un microcontrolador és configurar els ports de tal manera que treballin de la forma esperada.

Abans de procedir a la configuració dels recursos hem de tenir clar quins dispositius són d'entrada i quins de sortida, per tal de programar els seus respectius pins de manera correcta. A continuació mostrem una taula amb la distinció de quins dispositius són d'entrada i quins de sortida dels utilitzats en la pràctica:

ENTRADA	SORTIDA
Joystick Dreta	LED1
Joystick Esquerra	LED_R
Joystick Amunt	LED_G
Joystick Avall	LED_B


Taula 1.2: Dispositius d'entrada/sortida

Un cop feta la distinció de quins són els dispositius d'entrada i de sortida hem de procedir a la seva inicialització/configuració a nivell dels seus pins. Per tal de fer això hem d'escriure el contingut d'uns registres específics del microcontrolador. És molt important estudiar aquests registres abans de configurar res, ja que una mala configuració pot provocar un curt-circuit en algun pin.

Hem de tenir en compte també que si configurem un pin específic com a entrada/sortida digital hem d'especificar si és *entrada* o *sortida*. Un pin també pot treballar amb interrupcions, i això també ho hem hagut de configurar.

Totes aquestes configuracions inicials que fem les podrem anar canviant durant l'execució del programa segons ens interressi. És a dir, la configuració inicial no és restrictiva, la podem canviar durant el transcurs de l'execució del nostre programa.

Com haurem de fer servir les interrupcions en dos *timers* (A1 i A0), i en els ports 4 i 5 (pels *joysticks*), les hem de configurar a nivell del controlador d'interrupcions del processador, anomenat **NVIC** (*Nested Vectored Interrupt Controller*). Això ho fem de la següent forma:



```

1 // Port 4
2 NVIC->ICPR[1] |= 1 << (PORT4_IRQn & 31);
3 NVIC->ISER[1] |= 1 << (PORT4_IRQn & 31);
4
5 // Port 5
6 NVIC->ICPR[1] |= 1 << (PORT5_IRQn & 31);
7 NVIC->ISER[1] |= 1 << (PORT5_IRQn & 31);
8
9 // Timer A0
10 NVIC->ICPR[0] |= BIT8;
11 NVIC->ISER[0] |= BIT8;
12
13 // Timer A1
14 NVIC->ICPR[0] |= BITA;
15 NVIC->ISER[0] |= BITA;
```

Com hem d'habilitar les corresponents interrupcions, hem d'anar a la taula 6-39 del *Datasheet* (pàg. 116 i 117) on comprovem en primer lloc que el port 4 es igual a 38 i correspon al *BIT6* del segon registre *ISER1* i *ICPR1*, i com que els registres que fem servir són de 32 bits, realitzem la operació lògica  $PORT4\_IRQn \& 31 = 6$ , i desplaçem 1 sis cops a l'esquerra per obtenir el *BIT6*. Anàlogament ho realitzem per al port 5, que resulta ser el *BIT7* del segon registre *ICPR1* i *ISER1*.

Per a la configuració del NVIC (*Nested Vectored Interrupt Controller*) dels timers ho fem real-

itzant només la operació lògica *OR* per tal d'indicar que volem posar un 1 en el *BIT8* del primer registre *ISER0* i *ICPR0* (per al cas del *Timer A0*). De manera anàloga per a les següents configuracions dels dos *timers*, tal i com es pot veure en el codi mostrat anteriorment.

**Nota 1:** Amb la primera instrucció ens assegurem que no quedin interrupcions pendents als ports on estem configurant les interrupcions i amb la segona instrucció les habilitem.

**Nota 2:** La primera forma de configuració de l'*NVIC* pot semblar més enrevessada a simple vista però és més sofisticada ja que et neteja tot el registre i només activa les interrupcions que estàs configurant en aquell moment.

A continuació analitzarem els registres que hem hagut de configurar a l'inici del nostre programa per tal de configurar correctament els nostres recursos:

- **PxSEL0, PxSEL1:** Com volem que els pins d'aquests ports treballin com a GPIO (entrada/sortida digital), hem de configurar els seus respectius bits a 0 (fent ús d'operacions lògiques). Això ho fem amb les instruccions següents:



```

1 // LED vermell (P1.0)
2 P1SEL0 &= ~(BIT0);
3 P1SEL1 &= ~(BIT0);
4
5 // Joystick dreta (P4.5) i esquerra (P4.7); el joystick centre no ha estat
6 // utilitzat en aquesta practica.
7 P4SEL0 &= ~(BIT5 + BIT7);
8 P4SEL1 &= ~(BIT5 + BIT7);
9
10 // Joystick amunt (P5.4) i avall (P5.5)
11 P5SEL0 &= ~(BIT4 + BIT5);
12 P5SEL1 &= ~(BIT4 + BIT5);
13
14 // LEDs RGB vermell (P2.6) i verd (P2.4)
15 P2SEL0 &= ~(BIT4 + BIT6);
16 P2SEL1 &= ~(BIT4 + BIT6);
17
18 // LED RGB blau (P5.6)
19 P5SEL0 &= ~(BIT6);
20 P5SEL1 &= ~(BIT6);

```

- **PxDIR:** Aquest registre indica quin dels pins d'un port (dels que hem configurat com a entrada/sortida digital) seran d'entrada (bit = 0) i quins seran de sortida (bit = 1).

Nosaltres hem d'indicar que els moviments del *joystick* (esquerra, dreta, amunt i avall) seran entrades, el LED vermell serà sortida i els LEDs RGB (vermell, verd i blau) també seran sortides; això ho fem amb les següents instruccions:

```

1 // Joystick dreta (P4.5) i esquerra (P4.7); el joystick centre no ha estat
2 // utilitzat en aquesta practica.
3 P4DIR &= ~(BIT5 + BIT7);
4
5 // Joystick amunt (P5.4) i avall (P5.5).
6 P5DIR &= ~(BIT4 + BIT5);

```

També hem d'indicar que el LED vermell i els LEDs RGB seran sortides:

```

1 // LED vermell (P1.0)
2 P1DIR |= LED_V_BIT; // LED_V_BIT == BIT0
3

```

```

4 // LEDs RGB vermell (P2.6) i verd (P2.4)
5 P2DIR &= ~(BIT4 + BIT6);
6
7 // LED RGB blau (P5.6)
8 P5DIR &= ~(BIT6);
9

```

- **PxREN:** Aquest registre determina la resistència de l'entrada, fent que aquesta sigui *pull-up* o *pull-down* (Pàgina 678 del *Reference Manual*). Si no configuréssim aquest registre, l'estat del pin no estaria determinat i ens donaria resultats inesperats en el nostre programa.

Nosaltres hem de configurar aquest registre pels diversos moviments dels *joysticks* (els posarem un valor d'entrada igual a '1'), ja que els hem configurat com a entrades digitals; això es farà de la següent manera:

```

1 // Joystick dreta (P4.5) i esquerra (P4.7);
2 P4REN |= (BIT5 + BIT7);
3
4 // Joystick amunt (P5.4) i avall (P5.5).
5 P5REN |= (BIT4 + BIT5);
6

```

- **PxOUT:** És un registre destinat als pins que haguem configurat com a sortida digital (el nostre LED vermell i els LEDs RGB). Indicarà si escrivim l'estat del pin a la sortida o no.

Inicialment els LEDs estaran apagats (bit igual a 0) i això ho indiquem de la següent manera:

```

1 // LED vermell (P1.0)
2 P1OUT &= ~LED_V_BIT; // LED_V_BIT == BIT0
3
4 // LEDs RGB vermell (P2.6) i verd (P2.4)
5 P2OUT &= ~(BIT4 + BIT6);
6
7 // LED RGB blau (P5.6)
8 P5OUT &= ~(BIT6);
9

```

- **PxIE & PxIES:** Amb aquests dos registres habilitarem les interrupcions a nivell de dispositiu (perifèric). Modificarem aquests dos registres dels ports 4 i 5. Les instruccions utilitzades són les següents:

```

1 // Joystick dreta (P4.5) i esquerra (P4.7);
2 P4IE |= (BIT5 + BIT7);
3
4 // Joystick amunt (P5.4) i avall (P5.5).
5 P5IE |= (BIT4 + BIT5);
6
7 // Joystick dreta (P4.5) i esquerra (P4.7);
8 P4IES &= ~(BIT5 + BIT7);
9
10 // Joystick amunt (P5.4) i avall (P5.5).
11 P5IES &= ~(BIT4 + BIT5);
12

```

Amb la primera instrucció de cada port habilitem a nivell de dispositiu les interrupcions en els pins indicats.

Amb la segona instrucció de cada port volem que les interrupcions saltin al flanc de pujada L->H en els pins indicats.

- **TAxCCTLn**: Per tal d'activar les fonts d'interrupcions dels dos *timers* ho hem de fer configurant el seu registre *TaxCCTLn* (*BIT CCIE*; on **x** és el numero del *timer*, i **n** és la interrupció que volem configurar. Les instruccions utilitzades són les següents:

```

1  TIMER_A0->CCTL[0] |= TIMER_A_CCTLN_CCIE; // Capture & Compare enabled (
    interrupcions activades a CCR0)
2  TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; // Capture & Compare enabled (
    nterrupcions activades a CCR0)
3

```

- **TAxCTL**: Amb aquest registre configurem la forma de treballar del *timer* actual. Els bits MC s'utilitzen per a configurar els modes de funcionament (**Stop**: MC=00, **Up**: MC=01, **Continuous**: MC=10, **Up/Down**: MC=11).

```

1  // Registre TA0CTL; Timer A0 utilitzat pel LED vermell PWM
2  //Divider = 1; CLK source is SMCLK; clear the counter; MODE is up
3  TIMER_A0->CTL = TIMER_A_CTL_ID__1 | TIMER_A_CTL_SSEL__SMCLK | TIMER_A_CTL_CLR |
    TIMER_A_CTL_MC_UP;
4
5  // Registre TA1CTL; Timer A1 utilitzat pels LEDs RGB
6  //Divider = 1; CLK source is ACLK; clear the counter; MODE is up
7  TIMER_A1->CTL = TIMER_A_CTL_ID__1 | TIMER_A_CTL_SSEL__ACLK | TIMER_A_CTL_CLR |
    TIMER_A_CTL_MC_UP;
8

```

- **TAxCRR**: Aquest registre s'utilitza per aconseguir una freqüència determinada en el *timer*.

```

1  // Registre TA0CRR0 Donaria 1kHz. Es demanaven 10kHz, per tant, 24Mhz/2400=10kHz
2  TIMER_A0->CCR[0] = (.001 * 24000000) - 1; // 10 KHz (0.1 ms)
3
4  // Registre TA1CCR0
5  TIMER_A1->CCR[0] = (1 << 15) - 1; // 1 Hz (1 s)
6

```



#### 4 Funcions dels recursos

El funcionament dels recursos utilitzats en aquesta pràctica està descrit en l'enunciat, tot i així farem un resum:

- Generar una base de temps de l'ordre de 0'1 ms (10 kHz) per controlar la lluminositat del LED vermell. Aquesta funcionalitat la realitzem configurant el *Timer* A0 amb una font de rellotge *SMCLK* (*Sub-Main CLock*) i el *timer* en mode **UP**, és a dir, comptarà des de 0 fins el valor que l'assignem al registre *CCR0*, i tornarà a començar.

A més, la lluminositat del LED la podrem controlar canviant de valor una variable mitjançant els *joysticks* que farà que el LED s'apagui amb una major o menor freqüència (no perceptible a simple vista).

- Si movem el *joystick* amunt, s'ha d'incrementar la variable del *pwm\_duty* en una quantitat *step* (definida per la pròpia variable *step*).
- Si movem el *joystick* avall, s'ha de disminuir la variable del *pwm\_duty* en una quantitat *step* (definida per la pròpia variable *step*).
- Si movem el *joystick* a l'esquerra, s'ha de disminuir el valor de la variable *step* en 5.
- Si movem el *joystick* a la dreta, s'ha d'augmentar el valor de la variable *step* en 5.
- El *Timer* A1 generarà una base de temps d'1s (1 Hz) per tal de realitzar una certa seqüència de colors mitjançant els LEDs RGB. La font de rellotge del timer serà una *ACLK* (*Auxiliary Clock*) i el mode del *timer* serà **UP**, és a dir, comptarà des de 0 fins el valor que l'assignem al registre *CCR0*, i tornarà a començar.

Per tal d'aclarir les funcionalitats i com s'han de comportar els recursos, recomanem consultar l'apartat 3, on adjuntarem els diagrames de flux de tot el codi i de les diferents situacions en què ens podem trobar.

## 5 Problemes

### 5.1 1<sup>a</sup> sessió

- **Seleccionar font *timer*:** Tal i com es comenta a l'enunciat de la pràctica, disposem de dues fonts de rellotge pels *timers*: *ACLK* (*Auxiliary Clock*) a 37268 Hz i *SMCLK* (*Sub-Main CLock*) a 24MHz.

Per tal de configurar el *timer A0* vam haver de pensar quina font de rellotge escollíem per tal de generar la nostra base de temps de 0'1 ms.

La base de temps d'1 s pel *timer A1* ja ens venia donada amb el codi de la pràctica. Podíem comprovar com s'havia fet servir la font de rellotge *ACLK* (37268 Hz), nombre molt convenient quan es vol fer una base de temps d'1 s ja que al fer l'operació  $1 \ll 15$  se'ns desplaça l'1 quinze posicions a l'esquerra i el nombre binari que ens queda és: 1000000000000000 (37268).

Nosaltres però, vam creure convenient fer servir la font de rellotge *SMCLK* (*Sub-Main CLock*) per tal de generar la base de temps de 0'1 ms (10 KHz). Ja que al fer el càlcul de  $0,001 \cdot 24MHz$  obtenim 24.000. Això vol dir que necessitem un comptador de 15 bits (24.000 es representa amb 15 bits). És totalment acceptable ja que els *timers* del **MSP432** són de 16 bits.

- **No comprensió d'alguns apartats de l'enunciat:** A l'hora de programar el codi per tal de dur a terme la primera tasca en la que volíem poder variar la lluminositat del LED vermell, no ens quedava gaire clar l'enunciat i tampoc algunes de les variables que se'ns van entregar ja declarades en el codi.

Després de consultar amb els professors de pràctiques i llegir varis cops l'enunciat, vam entendre finalment el que es demanava.

Volíem entendre 100% l'enunciat ja que si no després no podríem dur a terme la pràctica correctament.

Vam veure llavors que la part interessant de la primera tasca era configurar una variable per tal de que actués com a **PWM** (*Pulse With Modulation*), per tal de permetre'ns modificar la lluminositat del LED.

### 5.2 2<sup>a</sup> sessió

- **No haver configurat els NVIC:** Un altre problema que vam experimentar durant el transcurs de la pràctica va ser que no vam activar les interrupcions dels *timers* al segon nivell, és a dir, a nivell de perifèric. És per això que el codi no estava funcionant de la manera que esperàvem.

Un cop repassat l'enunciat i la teoria ens vam adonar que no havíem configurat les interrupcions en tots els nivells. Es va solucionar ràpidament i vam comprovar com llavors el codi ja començava a funcionar.



- **No funcionava el LED\_B:** Un vegada pensàvem que havíem acabat el nostre codi per tal de generar el patró de color amb els LEDS RGB, vam poder comprovar com el LED blau no funcionava, ja que tots realitzaven el patró de manera correcta mentre que el blau no feia res.

Vam assegurar-nos de que no funcionava provant el codi en un altre robot i veient que funcionava correctament.

- **Instrucció `sizeof` no utilitzada correctament:** En un apartat del nostre codi en el que volíem comprovar si havíem arribat al final de l'*array* de *color\_sequence* vam fer ús de la instrucció *sizeof*. Aquesta però, ens retorna la mida en *bytes* del vector, i nosaltres volíem la mida en nombre d'elements.

Per tal de solucionar-ho vam haver de escriure la instrucció de la següent manera:

```
sizeof(color_sequence)/sizeof(color_sequence[0])
```

- **Mala configuració dels segons per la generació del patró:** Un cop ja ens funcionaven tots els recursos i el codi, vam poder comprovar que no havíem configurat bé els segons per a cada color del patró que s'havia de generar.

Va ser un problema ínfim que no afectava pas en el desenvolupament de la pràctica. No obstant, es va solucionar per tal d'adherir-nos completament al que demanava l'enunciat.

## 6 Conclusions

Aquesta segona pràctica ens ha servit per tal d'aprofundir en varis àmbits. Principalment hem tingut un primer contacte amb els *timers* que ofereix el microcontrolador, els qual hem utilitzat per programar en C les tasques demanades (i.e. generar una base de temps per controlar la lluminositat del LED vermell & generar una seqüència de colors amb els LEDs RGB amb un altre base de temps).

En la primera sessió de la pràctica vam aprendre a configurar correctament els ports que necessitàvem de la placa, així com a gestionar les interrupcions que havíem d'activar per tal de que el programa funcionés correctament. Tot i així, ens van sorgir uns quants problemes relacionats amb la comprensió de l'enunciat i a l'hora de seleccionar les diferents bases de temps per les dues tasques. No obstant, vam poder resoldre quasi tots els dubtes abans de finalitzar la sessió.


Durant la segona sessió de la pràctica vam resoldre alguns problemes que teníem amb la execució del programa i, a arrel d'això, vam entendre molt millor com funcionaven els recursos que estàvem fent servir, les seves interrupcions i sobretot els *timers*. Vam tenir alguns problemes degut a que el LED blau no funcionava a la placa del nostre robot, i això ens va fer perdre una mica de temps.

En general ha resultat ser una pràctica molt útil on hem après a relacionar hardware i software d'una manera interessant i intuïtiva. A més, el temps atorgat en la realització tant de la pràctica com de l'informe ha estat bastant encertat.

## 2 PROGRAMA COMENTAT

En aquesta secció s'inclourà el programa realitzat al *Code Composer* per la realització de la present pràctica. El codi inclourà comentaris detallats del seu funcionament per tal de demostrar la compressió del que s'està fent en tot moment.


A continuació adjuntem el codi de la pràctica degudament comentat:




```

1 #include <msp432p401r.h>
2 #include <stdio.h>
3 #include <stdint.h>
4 #include <stdbool.h>
5
6 #include "lib_PAE.h"
7
8 #define LED_V_BIT BIT0
9 #define LED_RGB_R BIT0
10 #define LED_RGB_G BIT1
11 #define LED_RGB_B BIT2
12
13 #define SW1_POS 1
14 #define SW2_POS 4
15 #define SW1_INT 0x04
16 #define SW2_INT 0x0A
17 #define SW1_BIT BIT(SW1_POS)
18 #define SW2_BIT BIT(SW2_POS)
19
20 /* Variables per saber el flag d'interrupcio dels joysticks */
21 #define JSTK5_INT 0x0C /* Joystick dreta i avall */
22 #define JSTK4_INT 0x0A /* Joystick amunt */
23 #define JSTK7_INT 0x10 /* Joystick esquerra */
24
25 /* Estructura que ens definira el patro a seguir pels LEDS RGB:
26  * -> 'r' fa referencia a RED (true: encès, false: apagat)
27  * -> 'g' fa referencia a GREEN (true: encès, false: apagat)
28  * -> 'b' fa referencia a BLUE (true: encès, false: apagat)
29  * -> 'time' fa referencia al temps que s'haurà de mantenir en el present
30  * color
31  * */
32 typedef struct
33 {
34     bool r, g, b;
35     uint8_t time;
36 } color_t;
37
38 /* Fent us de l'estructura creem la sequencia de colors que hauran de seguir
39  * els LEDS RGB de la placa */
40 color_t color_sequence[] = { { .r = true, .g = false, .b = false, .time = 2 },
41                             { .r = true, .g = true, .b = false, .time = 1 },
42                             { .r = false, .g = true, .b = false, .time = 3 },
43                             { .r = false, .g = false, .b = true, .time = 2 },
44                             { .r = true, .g = true, .b = true, .time = 1 }
45                             };
46
47
48 /*****
49  * INICIALIZACION DEL CONTROLADOR DE INTERRUPCIONES (NVIC).
50  *
51  * Sin datos de entrada
52  *
53  * Sin datos de salida
54  */

```






```

55  *****/
56 void init_interrupciones ()
57 {
58     // Configuracion al estilo MSP430 "clasico":
59     // --> Enable Port 4 interrupt on the NVIC.
60     // Segun el Datasheet (Tabla "6-39. NVIC Interrupts", apartado "6.7.2 Device-Level
        User Interrupts"),
61     // la interrupcion del puerto 1 es la User ISR numero 35.
62     // Segun el Technical Reference Manual, apartado "2.4.3 NVIC Registers",
63     // hay 2 registros de habilitacion ISER0 y ISER1, cada uno para 32 interrupciones
        (0..31, y 32..63, resp.),
64     // accesibles mediante la estructura NVIC->ISER[x], con x = 0 o x = 1.
65     // Asimismo, hay 2 registros para deshabilitarlas: ICERx, y dos registros para
        limpiarlas: ICPRx.
66
67     //Int. port 4 = 38 correspon al bit 6 del segon registre ISER1 (Joysticks dreta i
        esquerra):
68     NVIC->ICPR[1] |= 1 << (PORT4_IRQn & 31); //Primero, me aseguro de que no quede
        ninguna interrupcion residual pendiente para este puerto,
69     NVIC->ISER[1] |= 1 << (PORT4_IRQn & 31); //y habilito las interrupciones del puerto
70
71     //Int. port 5 = 39 correspon al bit 7 del segon registre ISER1 (Joysticks amunt i
        avall):
72     NVIC->ICPR[1] |= 1 << (PORT5_IRQn & 31); //Primero, me aseguro de que no quede
        ninguna interrupcion residual pendiente para este puerto,
73     NVIC->ISER[1] |= 1 << (PORT5_IRQn & 31); //y habilito las interrupciones del puerto
74
75     //Int. Timer_A0 correspon al bit 8 del primer registre ISER0:
76     NVIC->ICPR[0] |= BIT8; //Primero, me aseguro de que no quede ninguna interrupcion
        residual pendiente para este puerto,
77     NVIC->ISER[0] |= BIT8; //y habilito las interrupciones del puerto
78
79     //Int. Timer_A1 correspon al bit 10 del primer registre ISER0:
80     NVIC->ICPR[0] |= BITA; //Primero, me aseguro de que no quede ninguna interrupcion
        residual pendiente para este puerto,
81     NVIC->ISER[0] |= BITA; //y habilito las interrupciones del puerto
82 }
83
84 /*****
85  * INICIALIZACIO DEL LED DEL BOOSTERPACK MK II.
86  *
87  * Sin datos de entrada
88  *
89  * Sin datos de salida
90  *
91  *****/
92 void init_botons(void)
93 {
94     /* Configurem el LED vermell (els polsadors S1 i S2 no s'han inicialitzat
        * ja que en aquesta practica no es fan servir */
95     // *****/
96     // El LED vermell es GPIOs
97     P1SEL0 &= ~(BIT0);
98     P1SEL1 &= ~(BIT0);
99
100
101     //LED vermell = P1.0
102     P1DIR |= LED_V_BIT; //El LED es una sortida
103     P1OUT &= ~LED_V_BIT; //El estat inicial del LED es apagat
104
105 }
106
107 /*****

```



```

108 * INICIALIZACIO DELS JOYSTICKS DEL BOOSTERPACK MK II.
109 *
110 * Sin datos de entrada
111 *
112 * Sin datos de salida
113 *
114 *****/
115 void init_joysticks(void){
116     // Els Joysticks son GPIOs (el joystick centre no s'ha utilitzat en aquesta practica)
117     P4SEL0 &= ~(BIT5 + BIT7);
118     P4SEL1 &= ~(BIT5 + BIT7);
119     P5SEL0 &= ~(BIT4 + BIT5);
120     P5SEL1 &= ~(BIT4 + BIT5);
121
122     // Els joysticks son entrades
123     P4DIR &= ~(BIT5 + BIT7);
124     P5DIR &= ~(BIT4 + BIT5);
125
126     // Pull-up/Pull-down pels joysticks
127     P4REN |= (BIT5 + BIT7);
128     P5REN |= (BIT4 + BIT5);
129
130     // Donat que l'altre costat es GND, volem una pull-up
131     P4OUT |= (BIT5 + BIT7);
132     P5OUT |= (BIT4 + BIT5);
133
134     /* Interrupcions */
135     // Activem les interrupcions
136     P4IE |= (BIT5 + BIT7);
137     P5IE |= (BIT4 + BIT5);
138
139     // Amb transicio L->H
140     P4IES &= ~(BIT5 + BIT7);
141     P5IES &= ~(BIT4 + BIT5);
142
143     // Netegem les interrupcions anteriors
144     P4IFG = 0;
145     P5IFG = 0;
146 }
147
148 /*****
149 * CONFIGURACION DE LOS LEDs RGB. A REALIZAR POR EL ALUMNO
150 *
151 * Sin datos de entrada
152 *
153 * Sin datos de salida
154 *
155 *****/
156 void config_RGB_LEDS(void)
157 {
158     //LEDs RGB = P2.4, P2.6, P5.6
159     P2SEL0 &= ~(BIT4 + BIT6); // P2.4, P2.6 son GPIOs
160     P2SEL1 &= ~(BIT4 + BIT6); // P2.4, P2.6 son GPIOs
161     P5SEL0 &= ~(BIT6); // P5.6 es GPIO
162     P5SEL1 &= ~(BIT6); // P5.6 es GPIO
163
164
165     P2DIR |= (BIT4 + BIT6); //Els LEDs son sortides
166     P5DIR |= (BIT6); // Els LEDs son sortides
167     P2OUT &= ~(BIT4 + BIT6); //El seu estat inicial sera apagat
168     P5OUT &= ~(BIT6); //El seu estat inicial sera apagat
169


```

```

170 }
171
172 /*****
173  * INICIALIZACIO DELS TIMERS DEL BOOSTERPACK MK II .
174  *
175  * Sin datos de entrada
176  *
177  * Sin datos de salida
178  *
179  *****/
180 void init_timers(void)
181 {
182     //Timer A0, used for red LED PWM
183     //Divider = 1; CLK source is SMCLK; clear the counter; MODE is up
184     TIMER_A0->CTL = TIMER_A_CTL_ID_1 | TIMER_A_CTL_SSEL_SMCLK | TIMER_A_CTL_CLR |
185     TIMER_A_CTL_MC_UP;
186     TIMER_A0->CCR[0] = (.001 * 24000000) - 1; // 10 KHz (0.1 ms)
187     TIMER_A0->CCTL[0] |= TIMER_A_CCTLN_CCIE; // Capture & Compare enabled (interrupcions
188     activades a CCR0)
189
190     //Timer A1, used for RGB LEDs
191     //Divider = 1; CLK source is ACLK; clear the counter; MODE is up
192     TIMER_A1->CTL = TIMER_A_CTL_ID_1 | TIMER_A_CTL_SSEL_ACLK | TIMER_A_CTL_CLR
193     | TIMER_A_CTL_MC_UP;
194     TIMER_A1->CCR[0] = (1 << 15) - 1; // 1 Hz (1 s)
195     TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0
196 }
197
198 void main(void)
199 {
200     WDTCIL = WDIPW + WDIHOLD; // Stop watchdog timer
201
202     //Inicializaciones:
203     init_ucs_24MHz(); // Amb aquesta funcio, l'ACLK ens queda a 37268 Hz i SMCLK a 24 MHz
204     init_botons(); // Configurem el LED vermell
205     init_joysticks(); // Configurem els Joysticks que farem servir
206     config_RGB_LEDS(); // Configuracio dels LEDS RGB que farem servir
207     init_interrupciones(); //Configurar y activar les interrupcions dels botons
208     init_timers(); // Inicialitzem els timers (fonamentals en aquesta prÀctica)
209
210     __enable_interrupts(); // Habilitem les interrupcions a nivell global del
211     microcontrolador.
212
213     //Bucle principal (infinit):
214     while (true)
215     {
216         ;
217     }
218 }
219
220 /* Valors maxims i minims que pot valdre la nostra variable comptador */
221 #define CNT_MAX 100
222 #define CNT_MIN 0
223
224 /* Variable que marcara el valor on s'apaga el LED vermell */
225 volatile int8_t pwm_duty = 50;
226
227 /* Variable que anira guardant el temps/cops que entrem a la interrupcio pel timer (cada
228 1 segon pel timer A1) per tal de comprovar si hem de canviar al següent color del
229 patró o si l'hem de mantenir */
230 volatile int8_t aux_TA1 = 1;

```






```

227
228 /* Variable que ens servira per recorre l'array color_sequence que hem declarat al
    principi del programa */
229 volatile int8_t index_TA1 = 0;
230
231 /* Variable que podra incrementar o disminuir la variable pwm_duty segons les accions que
    fem amb els joysticks (tambe podra incrementar o decrementar el seu valor en 5) */
232 static uint8_t step = 0;
233
234
235 /*****
236  * RUTINES DE GESTIO DELS TIMERS:
237  * Mitjancant les rutines següents realitzarem les accions requerides pel timer A0 i el
    timer A1.
238  *
239  * Sense dades d'entrada
240  * Sense dades de sortida
241  *
242  * Timer A0 -> Modificar la intensitat de la llum del LED vermell.
243  * Timer A1 -> Generar un patró de colors amb els LEDs RGB.
244  *****/
245
246 // Interrupcio al timer A0 (rutina d'atencio a la interrupcio directa)
247 void TA0_0_IRQHandler(void)
248 {
249     // Variable comptador que anira de 0 a 100
250     static uint8_t cnt = 0;
251
252     TA0CCTL0 &= ~TIMER_A_CCTLN_CCIFG; //Clear interrupt flag
253
254     // Si el comptador ha arribat al maxím (100) encenem el LED vermell i tornem el
    comptador a 0
255     if (cnt == CNT_MAX) {
256         P1OUT |= LED_V_BIT;
257         cnt = 0;
258     }
259     /* Si el comptador val el mateix que el pwm_duty, vol dir que hem d'apagar el LED
    vermell i seguir comptant. */
260     else if (cnt == pwm_duty) {
261         P1OUT &= ~LED_V_BIT;
262         cnt++;
263     }
264     // Altrament seguim comptant
265     else {
266         cnt++;
267     }
268 }
269
270 // Interrupcio al timer A1 (rutina d'atencio a la interrupcio directa)
271 void TA1_0_IRQHandler(void)
272 {
273     TA1CCTL0 &= ~TIMER_A_CCTLN_CCIFG; //Clear interrupt flag
274
275     /* Variable de tipus color_t (struct) on anirem guardan els diferents vectors que es
    troben dins l'array color_sequence, on a dins de cada vector es troba la informacio
    per tal de generar el patró de colors demanat. */
276     color_t iter = color_sequence[index_TA1];
277
278     /* Per cada color comprovem si la seva variable es true o false, en cas de que sigui
    true encenem el LED, sino l'apaguem. */
279     if (iter.r) {
280         P2OUT |= BIT6;


```




```

281     } else {
282         P2OUT &= ~(BIT6);
283     }
284     if (iter.g) {
285         P2OUT |= BIT4;
286     } else {
287         P2OUT &= ~(BIT4);
288     }
289     if (iter.b) {
290         P5OUT |= BIT6;
291     } else {
292         P5OUT &= ~(BIT6);
293     }
294
295     /* Si el time del vector es diferent al comptador pel time que hem declarat, voldra
296     dir que encara hem de seguir amb el present color en el patro, per tant nomes
297     augmentem la nostra variable auxiliar i continuem amb el programa. */
298     if (iter.time != aux_TA1) {
299         aux_TA1++;
300     }
301     /* En cas de que ja s'hagi complert el temps pel present color en el patro, tornem a
302     situar la variable auxiliar al seu valor inicial. */
303     else {
304         aux_TA1 = 1;
305
306         /* Si hem arribat al final de la sequencia, tornem a situar l'index a 0 per tal
307         de començar un altre cop. */
308         if (index_TA1 == (sizeof(color_sequence) / sizeof(color_sequence[0]))) {
309             index_TA1 = 0;
310         }
311         // Si encara no ens trobem al final de la sequencia, augmentem l'index per tal de
312         seguir amb el següent color del patro
313         else {
314             index_TA1++;
315         }
316     }
317 }
318
319 /*****
320 * RUTINES DE GESTIO DELS BOTONS:
321 * Mitjancant aquestes rutines detectem quin boto s'ha polsat
322 *
323 * Sin Datos de entrada
324 * Sin datos de salida
325 *****/
326
327 // ISR per les interrupcions del port 4:
328 void PORT4_IRQHandler(void)
329 {
330     uint8_t flag = P4IV; // Guardem el vector d'interrupcions. A mes, a l'accedir a
331     aquest vector, es neteja automaticament
332     P4IE &= ~(BIT5 + BIT7); // Interrupcions dels joysticks dreta i esquerra desactivades
333
334     switch (flag) // El joystick centre no s'ha utilitzat en aquesta practica
335     {
336         /* Joystick dreta -> Incrementa la variable step en 5 */
337         case JSTK5_INT:
338             step += 5;
339             break;
340
341         /* Joystick esquerra -> Decrementa la variable step en 5 */
342         case JSTK7_INT:

```



```
337     step -= 5;
338     break;
339 }
340
341 // Interrupcions activades un altre cop
342 P4IE |= (BIT5 + BIT7);
343 }
344
345 //ISR para las interrupciones del puerto 5:
346 void PORT5_IRQHandler(void)
347 {
348     uint8_t flag = P5IV; // Guardem el vector d'interrupcions. A mes, a l'accedir a
349     aquest vector, es neteja automaticament
350     P5IE &= ~(BIT4 + BIT5); // Interrupcions dels joysticks amunt i avall desactivades
351
352     switch (flag)
353     {
354     /* Joystick amunt -> pwm_duty es veu incrementat una quantitat donada per la variable
355     step */
356     case JSTK4_INT:
357         pwm_duty += step;
358
359         /* Hem de controlar que pwm_duty no sigui mes gran que el maxim (100)*/
360         if (pwm_duty > CNT_MAX) {
361             pwm_duty = CNT_MAX;
362         }
363         break;
364
365     /* Joystick avall -> pwm_duty es veu disminuïda una quantitat donada per la variable
366     step */
367     case JSTK5_INT:
368         pwm_duty -= step;
369
370         /* Hem de controlar que pwm_duty no sigui mes petita que el minim (0)*/
371         if (pwm_duty < CNT_MIN) {
372             pwm_duty = CNT_MIN;
373         }
374         break;
375     }
376
377     // Interrupcions activades un altre cop
378     P5IE |= (BIT4 + BIT5);
379 }
```



### 3 DIAGRAMES DE FLUX

Els diagrames de flux representen un conjunt d'instruccions que es relacionen entre si formant una seqüència de passos que representen una certa operació que realitza, en aquest cas, un programa informàtic.

A continuació adjuntem 5 diagrames de flux:

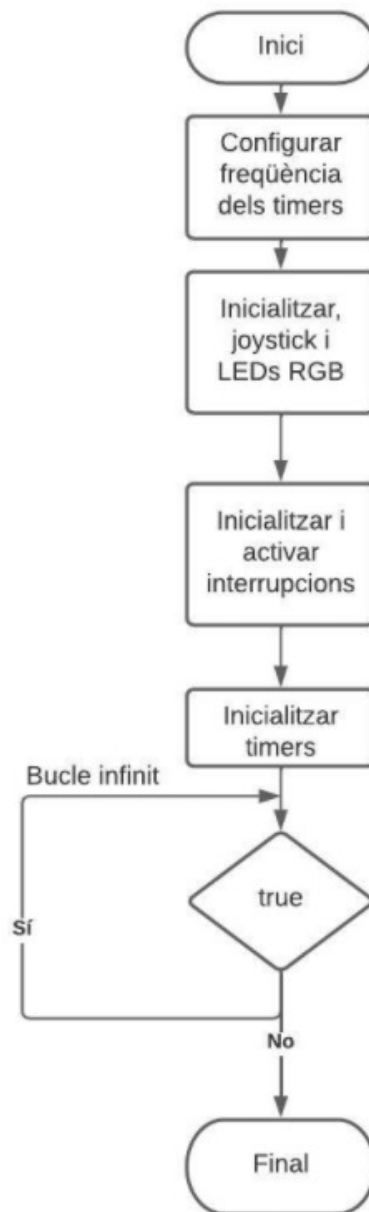
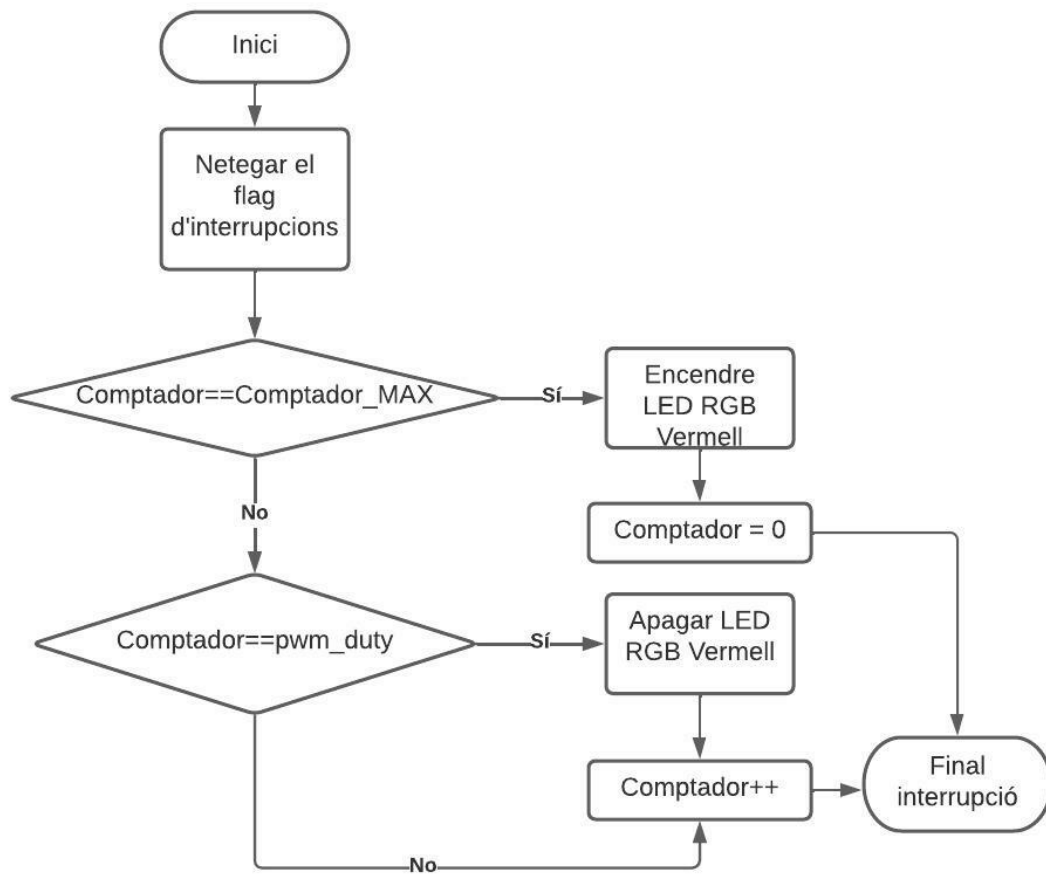
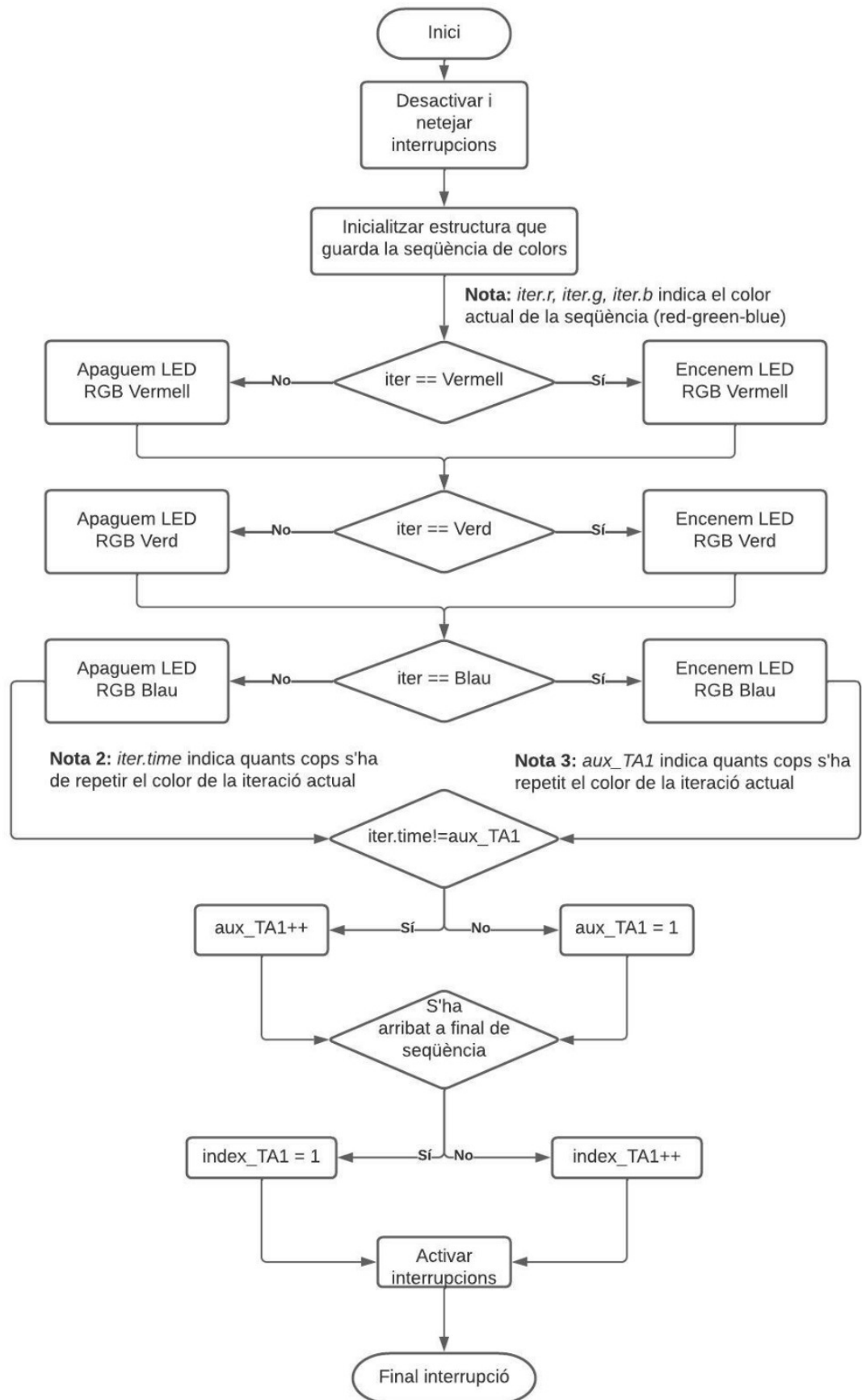


Figura 3.1: Diagrama de flux del *main* del programa

Figura 3.2: Diagrama de flux del *Timer A0*

Figura 3.3: Diagrama de flux del *Timer A1*

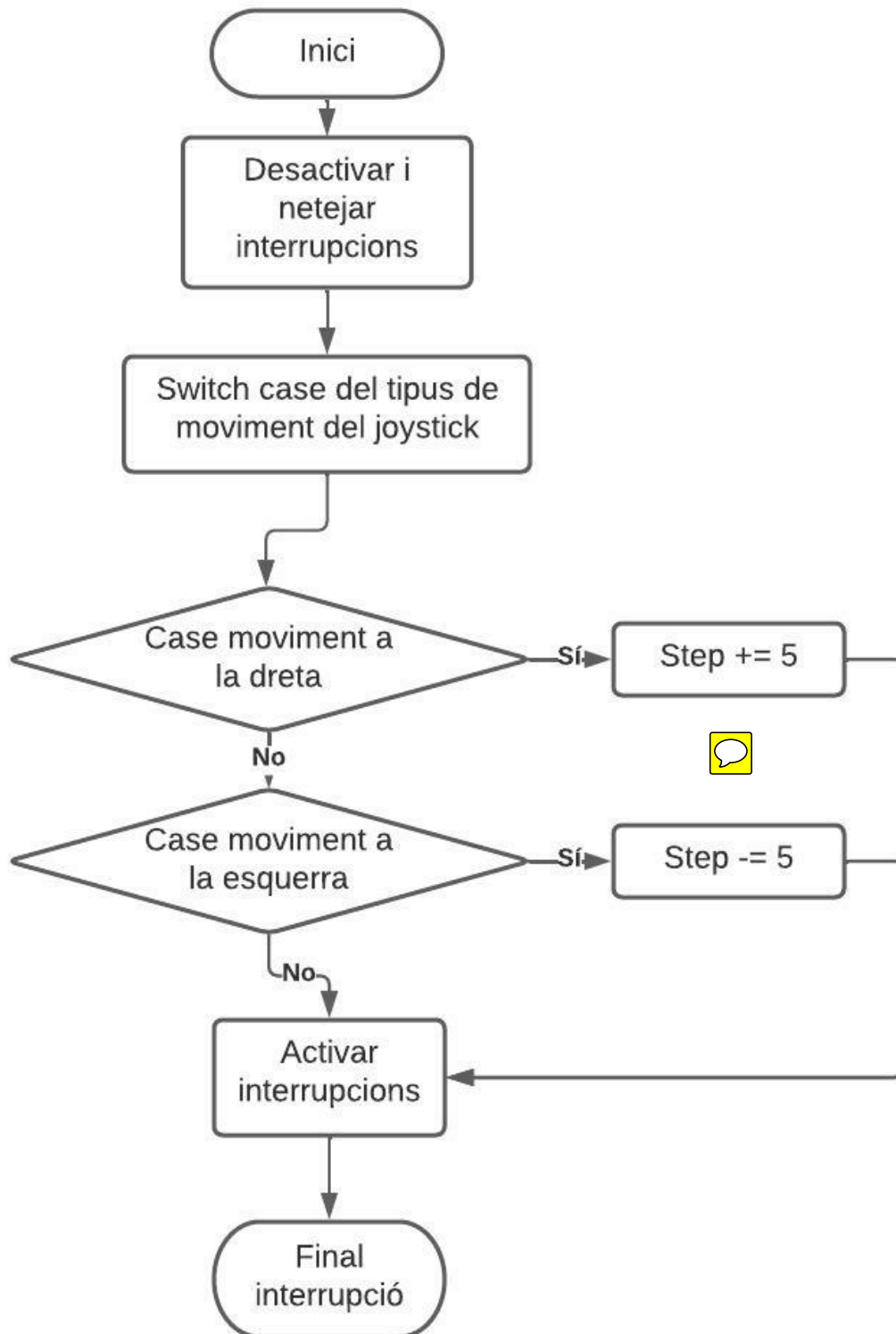
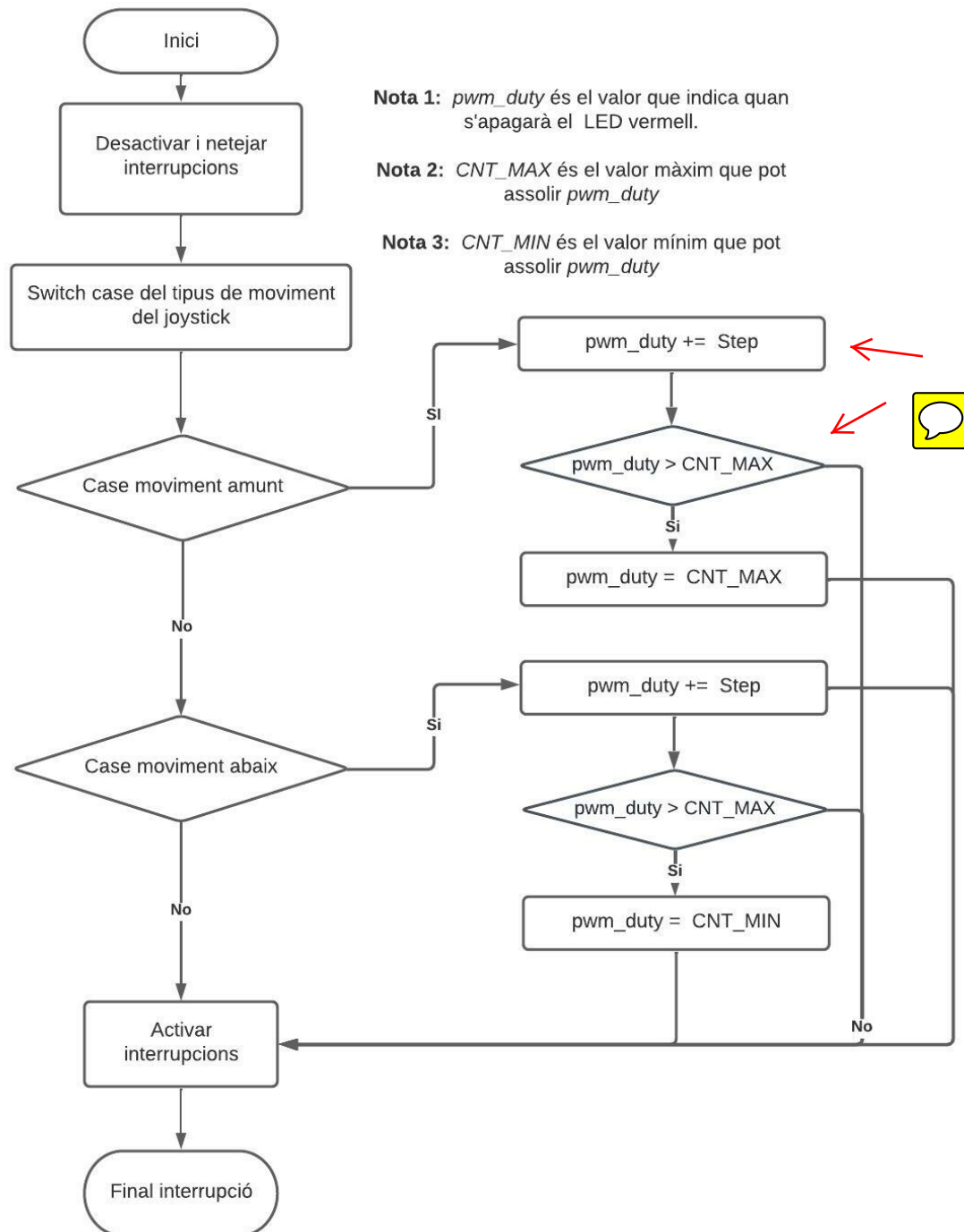


Figura 3.4: Diagrama de flux dels moviments dreta i esquerra del *joystick*

Figura 3.5: Diagrama de flux dels moviments amunt i abaix del *joystick*