



UNIVERSITAT^{DE}
BARCELONA

Pràctica 2. Configuració de ports

Noah Márquez Vara
Alejandro Guzman Requena

15 març 2022

ÍNDEX


1	Introducció	3
1	Què es vol fer a la pràctica	3
2	Recursos utilitzats	3
3	Configuració de recursos	3
4	Funcions dels recursos	5
5	Problemes	6
5.1	1 ^a sessió	6
5.2	2 ^a sessió	6
6	Conclusions	7
2	Programa comentat	8
3	Diagrames de flux	15

1 INTRODUCCIÓ

1 Què es vol fer a la pràctica

En aquesta pràctica l'objectiu principal és conèixer com es realitzen les configuracions dels ports d'entrada/sortida de propòsit general i com es gestionen les diverses interrupcions que poden sorgir donats uns certs events (en aquest cas, pulsar dos botons de la Placa d'Experimentació).

Treballarem amb diverses instruccions de programació per tal de fer salts condicionals i bucles, tal i com es podrà veure a l'apartat 2.

Tractarem amb dos ports de la nostra placa, concretament el port 1 (pels botons **S1** i **S2** i pel **LED1** vermell que **ha de fer pampallugues a un ritme fix**) i també treballarem amb el port 2, per tal de gestionar els LEDS RGB (Red-Green-Blue). 

2 Recursos utilitzats

Primerament, del microcontrolador hem fet servir els següents recursos:

- Port 1: Concretament els pins 0 (**LED1**), 1 (**botó S1**) i 4 (**botó S2**).
- Port 2: Concretament els pins 0 (**LED2_RED**), 1 (**LED2_GREEN**), i 2 (**LED2_BLUE**).

Els recursos utilitzats de la Placa d'Experimentació es poden extreure del comentat fins ara, tot i així, s'indiquen a continuació:

- Botó S1
- Botó S2
- LED1 (LED vermell separat que farà pampallugues a un ritme fix)
- Els tres LEDs RGB (identificats com a LED2)

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

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

Taula 1.1: Connexió dels recursos al Microcontrolador

El robot no s'ha fet servir per tal de dur a terme la realització d'aquesta pràctica.

3 Configuració de recursos

Tal i com indica l'enunciat (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
Botó S1	LED1
Botó S2	LED2_RED
	LED2_GREEN
	LED2_BLUE

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 els nostres dos botons (**S1** i **S2**), les hem de configurar a nivell del controlador d'interrupcions del processador, anomenat **NVIC**. Això ho fem de la següent forma:



```
NVIC->ICPR[1] |= BIT3;
NVIC->ISER[1] |= BIT3;
```

Com hem d'habilitar les interrupcions del port 1, hem d'anar a la taula 6-39 del *Datasheet*. (pàg. 116 i 117) i comprovem que el port 1 = 35, i com que els registres que fem servir són de 32 bits, fem la resta (35-32) i per això hem indicat *BIT3* (correspon al segon registre *ISER1*) a la instrucció que hem realitzat.

Amb la primera instrucció ens assegurem que no quedin interrupcions pendents al port 1.

Amb la segona instrucció habilitem les interrupcions al port 1.

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. Això ho fem amb les instruccions següents:

```
P1SEL0 &= ~(BIT0 + BIT1 + BIT4);
P1SEL1 &= ~(BIT0 + BIT1 + BIT4);
```

Com podem comprovar, posem a 0 els bits de les posicions 0, 1 i 4 del port 1, que corresponen als dos botons (**S1** **S2**) i al LED vermell que farà pampallugues.

- **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 el LED vermell serà una sortida i els botons seran entrades; això ho fem amb les següents instruccions:

```
P1DIR |= LED_V_BIT;
P1DIR &= ~(SW1_BIT + SW2_BIT);
```



També hem d'indicar que els LEDs RGB seran sortides:

```
P2DIR |= LED_V_BIT; // RED
P2DIR |= BIT1; // GREEN
P2DIR |= BIT2; // BLUE
```

- **PxREN:** Aquest registre ens ajuda a tenir en compte la situació en que un pin està configurat com a entrada digital i no rep res a l'entrada. 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 nostres dos botons (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:

```
P1REN |= (SW1_BIT + SW2_BIT);
```

- **PxOUT:** És un registre destinat als pins que haguem configurat com a sortida digital (els nostres LEDs). 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:

```
P1OUT &= ~LED_V_BIT;
P2OUT &= ~LED_V_BIT;
P2OUT &= ~BIT1
P2OUT &= ~BIT2
```

- **PxIE & PxIES:** Amb aquests dos registres habilitarem les interrupcions a nivell de dispositiu. Les instruccions utilitzades són les següents:

```
P1IE |= (SW1_BIT + SW2_BIT);
P1IES &= ~(SW1_BIT + SW2_BIT);
```

Amb la primera instrucció habilitem a nivell de dispositiu les interrupcions als 2 pins corresponents als botons **S1** i **S2**.

Amb la segona instrucció volem que les interrupcions saltin al flanc de pujada L->H.

4 Funcions dels recursos

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

- Si polsem **S1**, un nombre imparell de vegades, s'han d'encendre els 3 LEDs RGB.
- Si polsem **S1**, un nombre parell de vegades, s'ha d'invertir l'estat dels 3 LEDs RGB.
- Si polsem **S2**, s'han de desplaçar els LEDs encesos de tal forma que el nou estat del LED RGB verd correspon a l'anterior del vermell (**R** -> **G**) i el nou estat del LED RGB blau correspon a l'anterior del verd (**G** -> **B**). El nou valor del LED RGB vermell passarà a ser 0. Pulsacions consecutives del botó **S2** han de produir aquest efecte.

Per tal d'aclarir les funcionalitats i com s'han de comportar els recursos, mostrem una taula on s'indica clarament els diferents estats en que ens podem trobar i com passem d'un estat actual a un estat futur segons quin botó polsem:

ESTAT ACTUAL	POLSADOR	ESTAT FUTUR
1	S1	2
2	S1	1
X	S2	3

Taula 1.3: Taula d'estats

Tots els casos possibles en que ens trobarem segons les diferents combinacions de botons polsats s'indicaran de forma més clara en l'apartat 3, on s'adjutaran els diagrames de flux de tot el codi i de les diferents situacions en que ens podem trobar.

5 Problemes

5.1 1^a sessió

- **Els LED's RGB no s'encenien:**

En la primera sessió vam realitzar gairebé tot el codi necessari per a executar el programa, però a l'hora d'executar-lo els LEDs no feien el que esperàvem.

És per això que la última part de la primera sessió la vam dedicar a intentar esbrinar quin era el problema; ràpidament vam veure que havíem configurat el mètode *config_RGB_LEDS* del codi però no el cridàvem a l'iniciar el programa, és a dir, no els estàvem configurant. Això feia que els pins corresponents als LEDs romanguessin en el seu estat per defecte, i aquest era apagat.

5.2 2^a sessió

- **Els LEDs RGB no responien als dispositius d'entrada:**

En la segona sessió ens vam fixar que els LEDs no feien el que esperàvem segons la entrada dels botons, és a dir, no responien correctament als canvis d'estats després d'haver premut algun dels dos botons.

Això era degut a un error en el codi, que compilava correctament, però no feia el que volíem. Per tal de trobar on estava l'error vam haver de fer debug, i vam detectar que no construïem correctament la instrucció de canvi d'estat de cadascun dels LEDs. Tot era degut a que la variable que feiem servir per distingir si havíem premut un nombre parell o imparell de vegades no estava configurada correctament, impeding el correcte funcionament del nostre codi.

- **Warning d'inicialització de les variables *estado* i *estado_anterior* a -1:**

Al voler assignar un nombre negatiu a dues variables de tipus **unsigned int**, és a dir, sense signe, el *Code Composer* ens feia saltar un *Warning*.

Per tal d'evitar problemes, la variable *estado* la inicialitzem a *NULL* i la variable *estado_anterior* la inicialitzem a 4 (valor escollit arbitràriament) per tal de no fer-la coincidir amb cap

valor de la variable *estado* a l'hora d'inicialitzar el programa per primera vegada.

- **Per què no funcionava S2?**

Quan vam començar a programar el codi vam entendre que no podíem polsar dos cops seguits el botó **S2**, però a l'hora de repasar l'enunciat ens vam adonar que això no era així, que si que havíem de permetre polsar varies vegades seguides **S2**.

Vam aconseguir fer aquesta funcionalitat fent uns quants canvis al nostre codi per tal de que tot i que la nostre variable *estado* tingués el mateix valor (ja que hauríem premut dos cops seguits **S2**), poguéssim entrar al *switch* de dins el condicional *if* i realitzar la funció de **S2**.

6 Conclusions

Aquesta primera pràctica ens ha servit com a introducció en molts àmbits. Primerament hem tingut un primer contacte amb el programari *Code Composer*, el qual hem utilitzat per programar en C l'aplicació executada a la placa. També ens hem introduït a la programació d'arquitectures físiques, ja que fins ara havíem programat només amb software.

En la primera sessió de la pràctica vam aprendre a configurar correctament els ports de la placa, així com a gestionar interrupcions bàsiques dins del programa. Com que era el primer cop que ho feiem ens van sorgir errors de compilació i execució. Tot i així, els vam poder resoldre quasi tots abans de la finalització de 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 ports, els seus pins i les respectives configuracions que havíem plasmat en les instruccions.

En general ha resultat ser una pràctica molt útil on hem après a relacionar hardware i software d'una manera interessant i intuïtiva. A part, el temps otorgat 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 #define LED_V_BIT BIT0
7
8 #define SW1_POS 1
9 #define SW2_POS 4
10 #define SW1_INT 0x04
11 #define SW2_INT 0x0A
12 #define SW1_BIT BIT(SW1_POS)
13 #define SW2_BIT BIT(SW2_POS)
14
15 #define RETRASO 300000
16
17 /* Inicialització de les variables que necessitem durant l'execució del programa:
18 * --> estado: guardará l'estat en que ens trobem a cada moment per tal de fer una acció
19 * o un altre (l'hem inicialitzat a NULL per distingir quan polsem per primera vegada).
20 * --> check_S2: variable que farem servir per tal de poder polsar vÃries vegades S2 i
21 * aconseguir transferir els LEDS RGB tal i com demana l'enunciat (l'hem inicialitzat a
22 * false ja que encara no hem premut S2).
23 */
24 volatile uint8_t estado = NULL;
25 static bool check_S2 = false;
26
27 /*****
28 * INICIALIZACIÃN DEL CONTROLADOR DE INTERRUPCIONES (NVIC).
29 *
30 * Sin datos de entrada
31 *
32 * Sin datos de salida
33 *
34 *****/
35 void init_interrupciones()
36 {
37     // Configuracion al estilo MSP430 "clásico":
38     // --> Enable Port 4 interrupt on the NVIC.
39     // Segun el Datasheet (Tabla "6-39. NVIC Interrupts", apartado "6.7.2 Device-Level
40     // User Interrupts"),
41     // la interrupcion del puerto 1 es la User ISR numero 35.
42     // Segun el Technical Reference Manual, apartado "2.4.3 NVIC Registers",
43     // hay 2 registros de habilitacion ISER0 y ISER1, cada uno para 32 interrupciones
44     // (0..31, y 32..63, resp.),
45     // accesibles mediante la estructura NVIC->ISER[x], con x = 0 o x = 1.
46     // Asimismo, hay 2 registros para deshabilitarlas: ICERx, y dos registros para
47     // limpiarlas: ICPRx.
48
49     //Int. port 1 = 35 corresponde al bit 3 del segundo registro ISER1:
50     NVIC->ICPR[1] |= BIT3; //Primero, me aseguro de que no quede ninguna interrupcion
51     residual pendiente para este puerto,
52     NVIC->ISER[1] |= BIT3; //y habilito las interrupciones del puerto
53 }

```



```

48
49 /*****
50 * INICIALIZACI3N DE LOS BOTONES & LEDS DEL BOOSTERPACK MK II .
51 *
52 * Sin datos de entrada
53 *
54 * Sin datos de salida
55 *
56 *****/
57 void init_botons(void)
58 {
59     //Configuramos botones i LED vermell
60     // *****/
61     P1SEL0 &= ~(BIT0 + BIT1 + BIT4 );    //Els polsadors son GPIOs
62     P1SEL1 &= ~(BIT0 + BIT1 + BIT4 );    //Els polsadors son GPIOs
63
64     //LED vermell = P1.0
65     P1DIR |= LED_V_BIT;    //El LED es una sortida
66     P1OUT &= ~LED_V_BIT;    //El estat inicial del LED es apagat
67
68     //Bot3 S1 = P1.1 i S2 = P1.4
69     P1DIR &= ~(SW1_BIT + SW2_BIT );    //Un polsador es una entrada
70     P1REN |= (SW1_BIT + SW2_BIT );    //Pull-up/pull-down pel pulsador
71     P1OUT |= (SW1_BIT + SW2_BIT );    //Donat que l'altra costat es GND, volem una pull-up
72     P1IE |= (SW1_BIT + SW2_BIT );    //Interrupcions activades
73     P1IES &= ~(SW1_BIT + SW2_BIT );    // amb transicio L->H
74     P1IFG = 0;    // Netegem les interrupcions anteriors
75 }
76
77 /*****
78 * DELAY – A CONFIGURAR POR EL ALUMNO – con bucle for (com demana l'enunciat de la
79   pr3ctica)
80 *
81 * Datos de entrada: Tiempo de retraso. 1 segundo equivale a un retraso de 1000000 (aprox
82   )
83 *
84 * Sin datos de salida
85 *
86 *****/
87 void delay_t(uint32_t temps)
88 {
89     volatile uint32_t i;
90
91     /*****
92     * TODO PER PART DEL ALUMNE AMB UN BUCLE FOR
93     * Un cop implementat, comenteu o elimineu la seg3ent funció
94     *****/
95
96     //__delay_cycles(RETRASO);
97
98     // Soluci3
99
100     /* Simplement hem fet un bucle for que compti des de 0 fins al numero indicat ,
101      * per tal de generar un cert delay en el LED vermell que fa pampallugues.*/
102     for(i = 0; i < temps; i++){
103     }
104 }
105
106 /*****
107 * CONFIGURACI3N DE LOS LEDs DEL PUERTO 2. A REALIZAR POR EL ALUMNO
108 *
109 * Sin datos de entrada

```

```

108 *
109 * Sin datos de salida
110 *
111 *****/
112 void config_RGB_LEDS(void)
113 {
114     /* *****
115      * TODO PER PART DEL ALUMNE
116      * ***** */
117
118     /* En aquest mètode es realitza la inicialització dels LEDs del port 2 (RGB).
119      * Primerament s'indica que els LEDs actuaran com a sortida (emeten llum) i
120      * posteriorment s'indica explícitament que volem que tots els LEDs comencin
121      * apagats, per tal d'assegurar-nos que fan el que volem. */
122
123     //LED2_RED = P2.0
124     P2DIR |= LED_V_BIT;    //El LED es una sortida
125     P2OUT &= ~LED_V_BIT;   //El estat inicial del LED es apagat
126
127     //LED2_GREEN = P2.1
128     P2DIR |= BIT1;        //El LED es una sortida
129     P2OUT &= ~BIT1;       //El estat inicial del LED es apagat
130
131     //LED2_BLUE = P2.2
132     P2DIR |= BIT2;        //El LED es una sortida
133     P2OUT &= ~BIT2;       //El estat inicial del LED es apagat
134 }
135
136 void main(void)
137 {
138     /* Variable on guardarem l'estat anterior per tal de decidir si entrarem al
139      * condicional if. */
140     static uint8_t estado_anterior;
141
142     /* Variables que utilitzarem a l'hora de gestionar l'estat 3 (pulsació del botó S2)
143      * per tal de guardar els estats dels LEDS RGB (encesos o apagats) i fer-los servir per
144      * tal de realitzar la transició que demana l'enunciat de la practica. */
145     static uint8_t puerto_azul;
146     static uint8_t puerto_rojo;
147     static uint8_t puerto_verde;
148
149     /* Per tal d'evitar que entrem al condicional if al primer moment d'iniciar el nostre
150      * programa, inicialitzem la variable 'estado_anterior' a un nombre diferent dels que
151      * pot tenir la variable 'estado' (tot i que aquesta començarà a NULL). Li hem posat un
152      * valor de 4 (aleatoriament) i no el de '-1' que venia amb el codi de la practica ja
153      * que la variable 'estado_anterior' és de tipus unsigned int (així ens evitem un
154      * Warning). */
155     estado_anterior = 4;
156
157     /* El proposit del watchdog timer és reiniciar el microcontrolador obligatoriament si
158      * el codi es penja per algun motiu. Té un període de temps d'espera i, si no l'
159      * indiquem que tot està funcionant correctament, aquest reiniciarà el microcontrolador.
160      * En el nostre micro el watchdog timer està enés per defecte, és per aixó que a l'
161      * iniciar tots els programes l'hem de desactivar. */
162     WDTCIL = WDIPW + WDIHOLD;
163
164     /* Cridem als mètodes per inicialitzar els botons i configurar els LEDS RGB (mètodes
165      * indicats anteriorment). */
166     init_botons();
167     config_RGB_LEDS();

```

```

157  /* Cridem al étode que configura i inicialitza el controlador de les interrupcions
    dels botons. */
158  init_interrupciones();
159
160  /* Habilem les interrupcions a nivell global al registre de â statusâ del
    processador. */
161  __enable_interrupts();
162
163  //Bucle principal (infinito):
164  while (true)
165  {
166      /* Segons el valor de l'estat i de si em polsat anteriorment el botó S2 o no,
        realitzarem una acció o un altre. */
167      if (estado_anterior != estado || check_S2)
168      {
169          estado_anterior = estado;
170
171          /* *****+
            A RELLENAR POR EL ALUMNO BLOQUE switch (estado) ... case
            Para gestionar las acciones:
            Boton S1 presionado un número impar de veces, estado = 1
            Boton S1 presionado un número par de veces, estado = 2
            Boton S2, estado = 3
            ******/
172
173          /* Switch on segons l'estat en que ens trobem farem una acció o un altre ,
            seguint l'esquema d'estats i accions de l'enunciat. */
174          switch(estado) {
175              /* Estat = 1: s'han d'encendre tots els LEDs RGB, per tal de fer això fem
            ús d'una porta OR (|) per tal de fer una màscara i l'indiquem que volem posar un 1
            en els bits indicats (0, 1 i 2).*/
176              case 1:
177                  P2OUT |= (LED_V_BIT + BIT1 + BIT2);
178                  break;
179
180              /* Estat = 2: s'ha d'invertir l'estat dels LEDs RGB, per tal de realitzar
            -ho fem servir una porta XOR (^) per tal de fer una màscara i l'indiquem que volem
            invertir els bits indicats (e.g. si hi ha un '1' passa a ser un '0'). */
181              case 2:
182                  P2OUT ^= (LED_V_BIT + BIT1 + BIT2);
183                  break;
184
185              /* Estat = 3: hem de moure els estats dels LEDs RGB seguint les
            indicacions de l'enunciat de la pràctica; el valor del LED blau passa a ser el valor
            del LED verd, el valor del LED verd passa a ser el valor del LED vermell i el LED
            * vermell s'apaga.*/
186              case 3:
187                  /* Guardem en les variables l'estat del LED verd i del LED vermell (
            apagat o encés). */
188                  puerto_verde = P2OUT & BIT1;
189                  puerto_rojo = P2OUT & LED_V_BIT;
190
191                  /* Fem les transicions indicades anteriorment i a l'enunciat de la pr
            àctica fent un shift left en els dos casos, ja que en les variables anteriors tindrem
            guardat el valor per exemple del 'puerto_verde' de la següent forma:
            * 00000010 (LED verd encés) i si volem que el 'puerto_azul' tingui
            aquest estat, hem de desplaçar a l'esquerra de tal forma que el bit del verd (1)
            estigui en la posició del bit blau (2), de tal forma que ens quedaria així:
            * 00000100. */
192                  puerto_azul = puerto_verde << 1;
193                  puerto_verde = puerto_rojo << 1;
194
195
196
197
198
199
200
201
202
203

```

```

204      /* En cas que després de fer el desplaçament el bit verd s'hagi d'
encendre, fem ús d'una porta OR per tal de posar un 1 en la seva posició (BIT1).
Altrament, apaguem el LED verd com hem fet a l'hora de configurar-lo. */
205      if(puerto_verde == 0x02){
206          P2OUT |= puerto_verde;
207      } else {
208          P2OUT &= ~BIT1;
209      }
210
211      /* El mateix que pel LED verd pero ara per tal de modificar el bit
del LED blau (BIT2). */
212      if(puerto_azul == 0x04){
213          P2OUT |= puerto_azul;
214      } else {
215          P2OUT &= ~BIT2;
216      }
217
218      /* Sigui quin sigui la transformació realitzada, hem d'apagar el LED
vermell posant un 0 a la seva posició (BIT0 o LED_V_BIT). Per fer això fem ús d'una
porta AND i neguem el contingut de LED_V_BIT (00000001) de tal forma que al
219      * negar-ho quedaria de la següent forma: 11111110, i al fer una
porta AND amb P2OUT, ens quedarem amb els seus valors anteriors i posarem un 0 a la
posició 0. */
220      P2OUT &= ~LED_V_BIT;
221
222      /* En cas de que haguem entrat al switch per haver premut dos cops
seguits el botó S2, desactivem la variable que ens permet saber si l'hem premut dos
cops seguits o no. */
223      if(check_S2){
224          check_S2 = false;
225      }
226      break;
227  }
228  }
229
230      /* Commutem l'estat del LED vermell que fa pampallugues. */
231      P1OUT ^= LED_V_BIT;
232
233      /* Cridem a la funció que hem realitzat anteriorment per tal de donar un cert per
íode de delay al LED. */
234      delay_t(RETRASO);
235  }
236  }
237
238  /* *****
239  * RUTINAS DE GESTION DE LOS BOTONES:
240  * Mediante estas rutinas, se detectará qué botón se ha pulsado
241  *
242  * Sin Datos de entrada
243  *
244  * Sin datos de salida
245  *
246  * Actualizar el valor de la variable global estado
247  *
248  * *****/
249
250  //ISR para las interrupciones del puerto 1:
251  void PORT1_IRQHandler(void)
252  {
253      /* Variable que ens permetra saber en tot moment si hem premut un nombre imparell o
parell de vegades el botó S1. */
254      static bool impar;

```

```

255
256  /* Guardem el vector d'interrupcions del port 1. Alhora d'accedir-hi, es neteja
    automaticament, és a dir, es desactiven els flags que han fet saltar la interrupció.
    */
257  uint8_t flag = P1IV;
258
259  /* Per tal d'evitar que salti un altre interrupció mentre encara estem gestionant la
    present, les desactivem. */
260  P1IE &= ~(SW1_BIT + SW2_BIT );
261
262  /*****
263   A RELLENAR POR EL ALUMNO
264   Para gestionar los estados:
265       Boton S1 presionado un número impar de veces, estado = 1
266       Boton S1 presionado un número par de veces, estado = 2
267       Boton S2, estado = 3
268   *****/
269
270  switch (flag)
271  {
272  /* S'ha premut el botó S1. */
273  case SW1_INT:
274      /* En cas que el primer botó que polsem sigui S1, l'hauem premut un nombre
        imparell de vegades (1), i per tant hem de posar l'estat a 1 i la variable impar a
        false, ja que el següent cop que premem el botó S1, ja será parell. */
275      if (estado == NULL) {
276          estado = 1;
277          impar = false;
278      }
279      /* En cas que ja haguem premut algun dels dos botons (S1 o S2) anteriorment, hem
        de distingir el cas en que haguem premut un nombre imparell/parell de vegades S1 per
        tal de decidir quin valor assignar a la variable 'estado'. */
280      else {
281          if (impar) {
282              estado = 1;
283          } else {
284              estado = 2;
285          }
286
287          /* Un cop assignat el valor a la variable 'estado', invertim el valor de la
            variable 'impar'; si havíem premut un nombre imparell de vegades, doncs ara 'impar'
            será true per tal d'indicar que el següent cop que premem el botó S1, será un nombre
            imparell i viceversa. */
288          impar = !impar;
289      }
290      break;
291
292  /* S'ha premut el botó S2. */
293  case SW2_INT:
294      /* Si l'estat és igual a 3, vol dir que el botó que s'ha polsat anteriorment ha
        estat S2, per tant posem a true la variable 'check-S2' que ens permetrà entrar a la
        condició de l'if del main per tal de poder prémer varis cops el botó S2. */
295      if (estado == 3) {
296          check_S2 = true;
297      }
298
299      /* Si S2 és el primer botó que premem, posem l'estat a 3 i iniciem la variable '
        impar' a true per tal d'indicar que quan polsem S1 per primera vegada, será imparell.
        */
300      else if (estado == NULL) {
301          estado = 3;
302          impar = true;

```

```
303     }
304     /* En qualsevol altre cas, indiquem que l'estat és igual a 3. */
305     else {
306         estado = 3;
307     }
308     break;
309 }
310
311 /* Un cop que hem gestionat la interrupció, tornem a activar les interrupcions dels
312    nostres botons (S1 i S2) en el port 1. */
312 P1IE |= (SW1_BIT + SW2_BIT);
313 }
```

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 dos diagrames de flux, el del *main* del programa i el de les gestions d'interrupció:

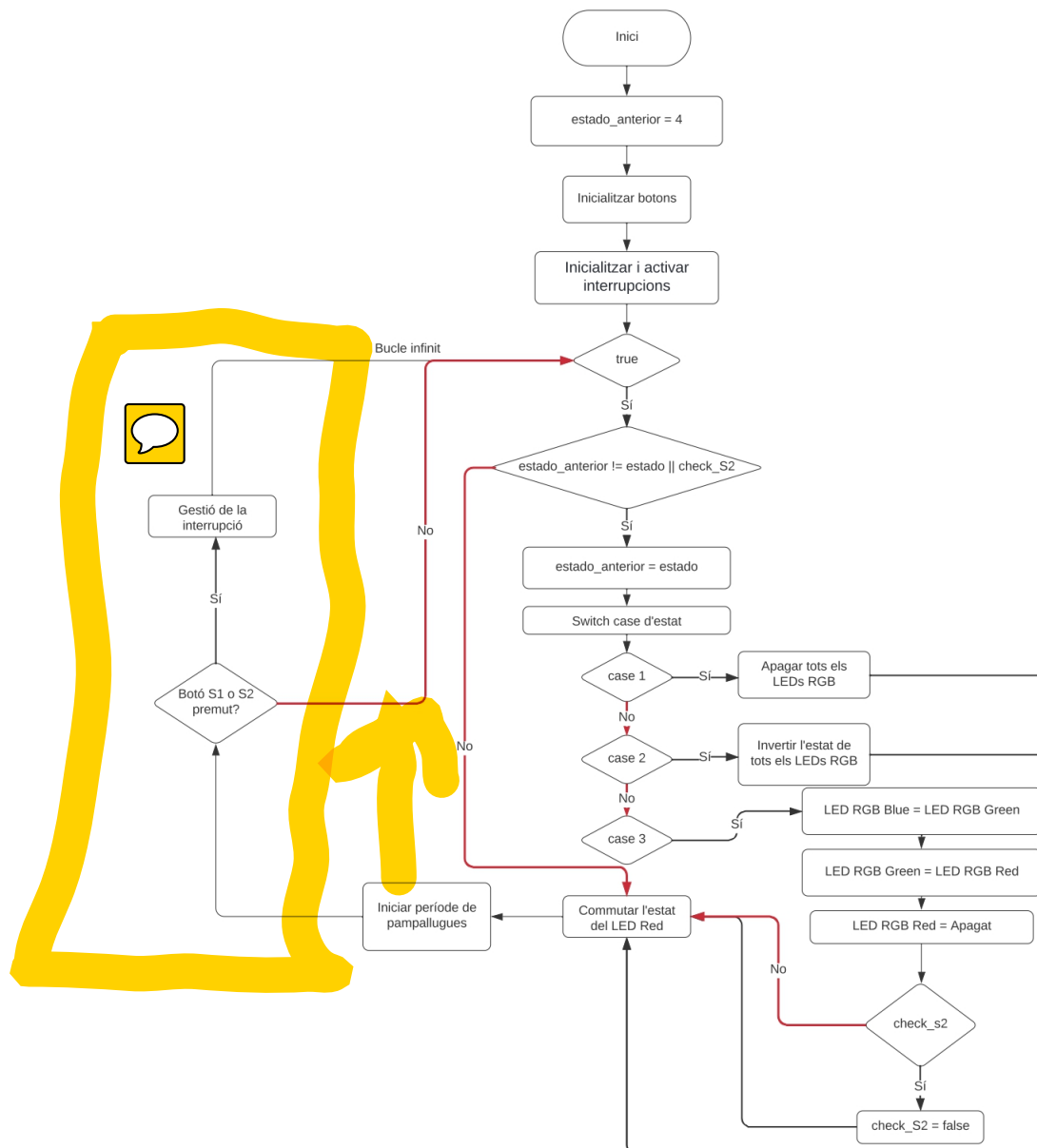


Figura 3.1: Diagrama de flux del *main* del programa

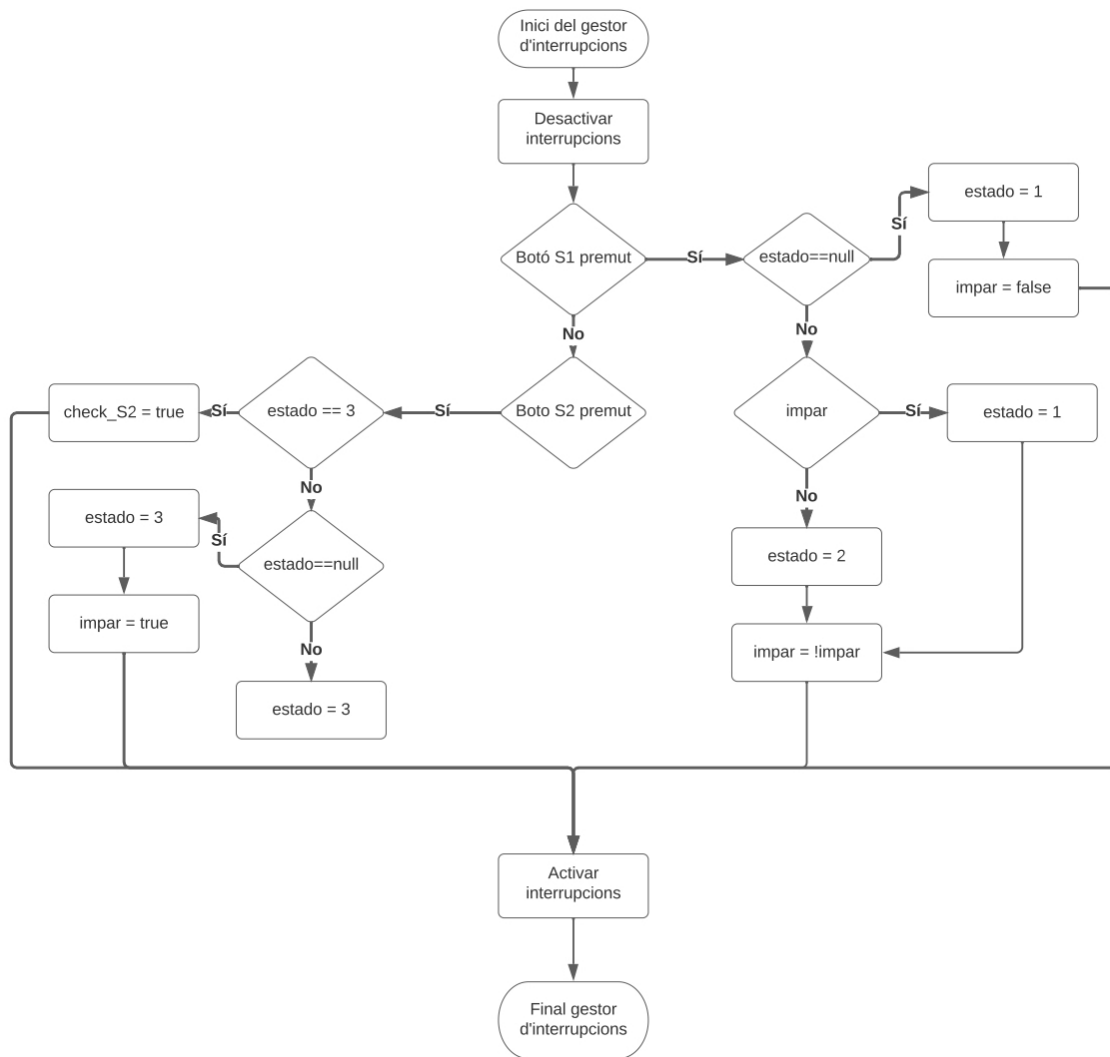


Figura 3.2: Diagrama de flux de la gestió d'interrupcions