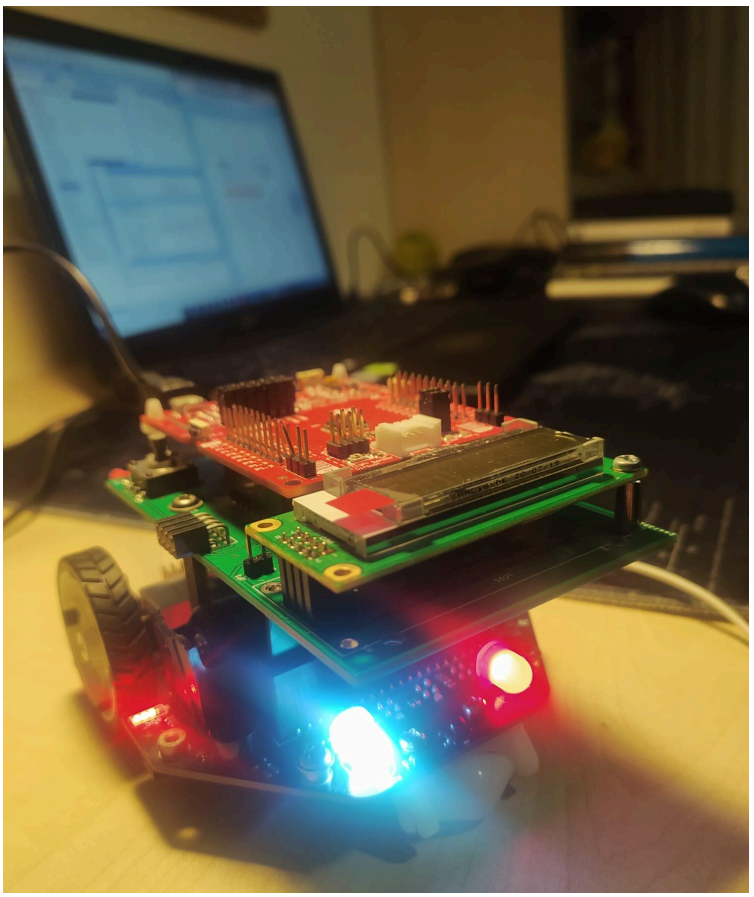
UNIVERSITAT DE BARCELONA

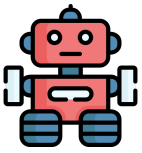
Desenvolupament del software del robot

Enginyeria Electrònica de Telecomunicació
Microcontroladors i Sistemes Empotrats Curs 2024/2025



Aidar Iglesias **David García**

ÍNDEX



MOTIVACIÓ I OBJECTIUS

En aquest projecte, s'ha implementat un conjunt de funcionalitats a fi de controlar diferents perifèrics com LEDs RGB, motors, una pantalla LCD i un seguidor de línia, tots interconnectats mitjançant el protocol de comunicació I2C. L'objectiu principal ha estat programar aquest sistema per tal que pugui executar tasques senzilles, mitjançant interrupcions, temporitzadors i rutines d'espera.

Els diferents objectius particulars d'aquest informe son:

- ➢ Desenvolupar el software de manera escalada, mitjançant diferents fitxers C i generar els fitxers de capçalera.
- ➢ Crear una funció d'espera per controlar els retards que es troben dins del codi.
- ➢ Controlar diferents perifèrics mitjançant I2C. Entre ells, controlar els LEDs del robot, els actuadors de les rodes, la pantalla LCD i el seguidor de línia

ESTRUCTURA GENERAL DEL SOFTWARE

El software del projecte s'organitza de manera modular per facilitar la llegibilitat, la reutilització i el manteniment del codi. Cada funcionalitat del robot relacionada amb la comunicació I2C o amb l'ús dels perifèrics digitals del microcontrolador es desenvolupa en fitxers de tipus C independents, acompanyats dels seus corresponents fitxers de capçalera, que defineixen les funcions globals accessibles des del programa main.

Aquesta modalitat permet encapsular les funcionalitats específiques com el control dels LEDs RGB, la gestió dels motors, la comunicació amb el display LCD o la configuració de rellotges i temporitzadors de manera individual.

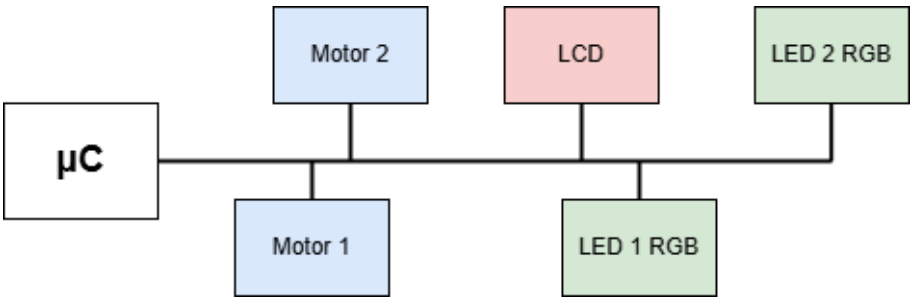


Figura 1: *Visualització de diagrama I2C amb els diferents perifèrics.*

RECURSOS DE HARDWARE

Els models comercials que s'implementaran, juntament amb els seus corresponents **Datasheets**, es presenten a la **Taula 1**. A la **Taula 2**, es proporciona una breu descripció de cadascun.

Bloc	Recurs comercial	datasheet
Microcontrolador	MSP430FR2355TPT	MSP430FR2355TPT
Robot	DFRobot Mcqueen Plus	DFRobot Mcqueen Plus
LCD	MD21605B6W-FPTLW3	MD21605B6W-FPTLW3
Ultrasons	URM10	SEN0307 URM09
Oscil·lador	LFXTAL002997BULK	LFXTAL002997BULK
Joystick	THB001P	THB001P
LDRs	NSL-19M51	NSL-19M51
Mòdul wifi	ESP01	ESP01
Lògica joystick	LMV324	LMV324

Taula 1: *Visualització dels recursos amb els respectius datasheets.*

Recurs comercial	Característiques
MSP430FR2355TPT	Baix consum, memòria FRAM ràpida i múltiples protocols de comunicació
DFRobot Mcqueen Plus	Robot amb múltiples motors, sensors i funcionalitats.
MD21605B6W-FPTLW3	Interfaç I2C senzilla, baix consum, més econòmic que una pantalla LED.
URM10	Sensor d'alta precisió, però és més car que altres mòduls d'ultrasons.
LFXTAL002997BULK	Bona estabilitat, però requereix condensadors de càrrega per funcionar correctament.
THB001P	Joystick analògic amb un control precís dels dos eixos i un polsador integrat.
NSL-19M51	Té una resposta no lineal que depèn molt de la il·luminació ambiental.
ESP01	Aquest model concret de mòdul wifi té un baix cost, ocupa poca àrea i és programable amb altres entorns com arduino.
LMV324	Operacional LM324 amb baix consum.

Taula 2: *Visualització dels recursos amb una descripció.*

TAULA AMB PINS DEL µC

Per poder desenvolupar el software i saber quins pins s'han d'inicialitzar com a GPIOs així com tenir una idea de quins pins s'associen a cada funció del robot, s'ha adjuntat la taula amb les connexions del microcontrolador, indicant el nom del pin, el mòdul que es connecta a aquests pins i una petita descripció de la seva funció, especificant si es tracta d'una entrada o una sortida digital, una línia de comunicació (I2C, UART), una entrada analògica o un senyal específic.

Pin μC	Etiqueta	Component	Port/Pin amb funcionalitats	Descripció
2	$ACLK_{TP}$	Conn pinheader	**P1.1**/UCB0CLK/ACLK/OA0O/COMP0.1/A1	Output. Test point del ACLK
3	$SMCLK_{TP}$	Conn_pinheader	**P1.0**/UCB0STE/SMCLK/COMP0.0/A0/Veref+	Output. Test point del SMCLK
4	SBW_{TCK}	Spy By Wide	TEST/SBWTCK	I/O Pin de dades del SPW
5	SBW_{TDIO}	Spy By Wide	RST/NMI/SBWTDIO	I/O Pin de rellotge del Spy By Wire
6	VCC	VCC	DVCC	Alimentació VCC
7	GND	GND	DVSS	Gnd, punt de referència
8	$XT1_{IN}$	CRISTALL	**P2.7**/TB0CLK/XIN	Input. Pin 1 del cristall
9	$XT1_{OUT}$	CRISTALL	**P2.6**/MCLK/XOUT	Input. Pin 2 del cristall
12	$I2C_{SCL}$	LCD	**P4.7**/UCB1SOMI/UCB1SCL	SCL del I2C para conn.robot y LCD
13	$I2C_{SDA}$	LCD	**P4.6**/UCB1SIMO/UCB1SDA	SDA del I2C para conn. robot y LCD
14	$\overline{RST_{LCD}}$	LCD	**P4.5**/UCB1CLK	Reset del LCD
20	$P0$	CONN_ROBOT	**P6.2**/TB3.3	I/O. Pin digital per connexió robot
21	$P1$	CONN_ROBOT	**P6.1**/TB3.2	I/O Pin digital per connexió robot
22	$P2$	CONN_ROBOT	**P6.0**/TB3.1	I/O Pin digital per connexió robot
27	$P8$	CONN_ROBOT	**P2.3**/TB1TRG	I/O Pin digital per connexió robot
31	RX_{ESP}	MÒDUL WIFI	**P1.7**/UCA0TXD/UCA0SIMO/TB0.2/TDO/OA1+/A7/VREF+	Input. Connexió de recepció del mòdul wifi tipus UART.
32	TX_{ESP}	MÒDUL WIFI	**P1.6**/UCA0RXD/UCA0SOMI/TB0.1/TDI/TCLK/OA1-/A6	Output. Connexió transmissió del mòdul wifi tipus UART.
37	JTK_R	joystick	**P3.5**/OA3O	Output. Pin digital del port 3 del joystick que determina direcció dreta (right).
38	JTK_F	joystick	**P3.4**/SMCLK	Output. Pin digital del port 3 del joystick que determina direcció endavant (forward)
42	LDR_R	LDR	**P5.1**/TB2.2/MFM.TX/A9	Pin analògic del port 5 per LDR dret (right)
43	LDR_L	LDR	**P5.0**/TB2.1/MFM.RX/A8	Pin analògic del port 5 per LDR esquerra (left)
44	JTK_{SEL}	joystick	**P3.3**/OA2+	Ouput. Pin digital del pulsador del joystick per seleccionar al LCD.
45	JTK_B	joystick	**P3.2**/OA2-	Output. Pin digital del port 3 del joystick que determina direcció abaix (backward).
46	JTK_L	joystick	**P3.1**/OA2O	Output. Pin digital del port 3 del joystick que determina direcció esquerra (left).

Taula 3: *Taula amb els diversos components de la placa i els pins corresponents del microcontrolador.*

DESCRIPCIÓ DE FUNCIONS DEL ROBOT

Com anteriorment s'ha esmentat, el robot integra diferents funcionalitats, proporcionades pels diferents perifèrics. Aquestes són controlades per una placa de desenvolupament MSP-EXP430FR2355, la qual té com a microcontrolador principal el MSP430FR2355 de Texas Instruments.

El desenvolupament de l'entorn de software del robot inclou funcions per al control de temporitzadors mitjançant l'ús de rellotges i funcions per al control de perifèrics, tals com LEDs RGB, actuadors i un display LCD.

El control dels LEDs RGB es realitza via protocol de comunicació I2C. S'han definit funcions per al control de l'encesa i l'apagada dels esmentats, així com per al color.

A més, el robot també té la capacitat de desplaçar-se en totes les direccions del pla sobre el qual reposa gràcies a un conjunt de dos actuadors programables per usuari. Per a aquest perifèric s'han desenvolupat un seguit de funcions que permeten el control del moviment del robot en diverses direccions. La comunicació entre la placa i el controlador dels actuadors també es fa a través d'I2C.

Addicionalment, el robot està dotat d'una petita pantalla de tipus LCD de dues línies de 16 caràcters per línia, també controlada mitjançant el protocol I2C. En el projecte s'ha creat funcions per a la representació de caràcters a través de la pantalla LCD.

Per la mateixa naturalesa del protocol de comunicació, per a cadascun dels mòduls de software que implementen el control de cadascun dels perifèrics s'ha establert un format de trama específic que inclou l'adreça del perifèric a controlar, el conjunt de dades a enviar i la longitud de la trama.

Cal destacar que totes les funcions esmentades anteriorment tenen la seva contrapart d'inicialització, que és necessària per al correcte funcionament dels perifèrics del nostre robot i, en general, per a qualsevol sistema encastat.

Rellotges

En aquest projecte s'han utilitzat dues fonts de rellotge: una de 16 MHz proporcionada pel DCO (*Digitally Controlled Oscillator*) i una altra de 32 kHz generada pel cristall extern XT1. A través dels pins de sortida configurats al LaunchPad, s'han pogut visualitzar amb l'oscil·loscopi els dos senyals corresponents. A les **Figures 2** i **3**, respectivament, s'observa clarament que el senyal de 16 MHz presenta una forma d'ona més distorsionada en comparació amb la de 32 kHz. Aquesta distorsió és causada per diversos factors, com ara el soroll introduït per l'elevada freqüència, la capacitat paràsita del circuit, o limitacions de la instrumentació utilitzada. En canvi, el senyal de 32 kHz es mostra molt més estable i net, ja que treballa a una freqüència inferior i prové d'un oscil·lador més estable com és el cristall.

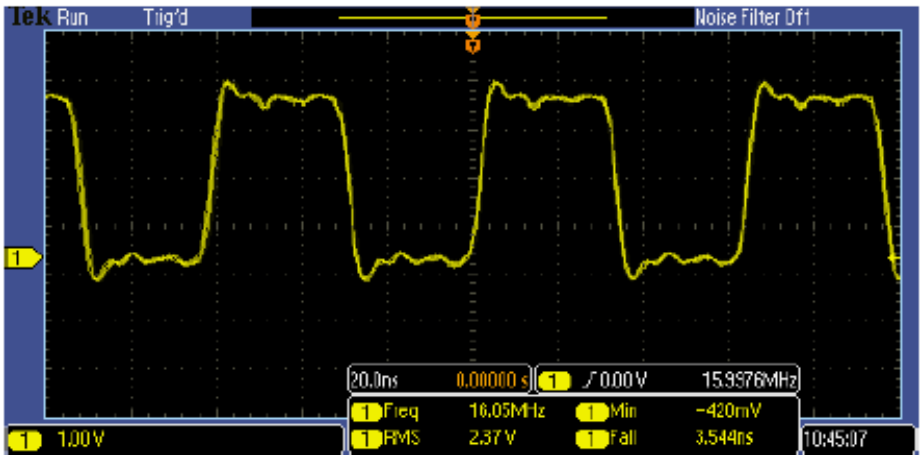


Figura 2: *Visualització de la font de rellotge de 16 MHz.*

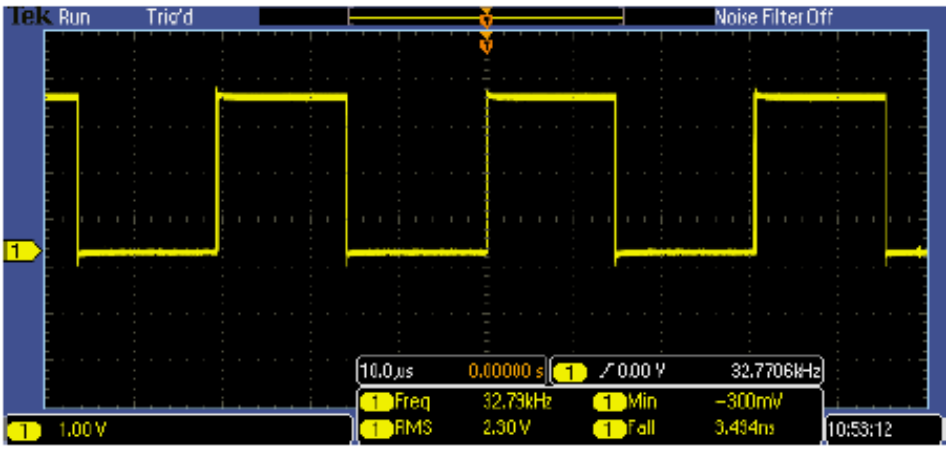


Figura 3: *Visualització de la font de rellotge de 32 kHz.*

Funció `delay_ms()`

Per implementar retards precisos al sistema, s'ha desenvolupat la funció `delay_ms()`, que permet generar una espera en mil·lisegons. Aquesta funció inicialitza un comptador global `count_ms` i utilitza el Timer B3 del microcontrolador per incrementar aquest valor mitjançant una interrupció generada cada 1 ms. Quan es crida la funció, s'entra en un bucle d'espera que no finalitza fins que el comptador arriba al valor indicat. L'increment del comptador es produeix a través de la subrutina d'interrupció `TimerB0_ISR()`. A la **Figura 4** es pot observar el diagrama de flux corresponent al funcionament de la funció `delay_ms()`. A la **Figura 5** es pot visualitzar el pols generat per un GPIO configurat com a sortida amb període de 200 ms (retard de 100 ms entre commutacions).

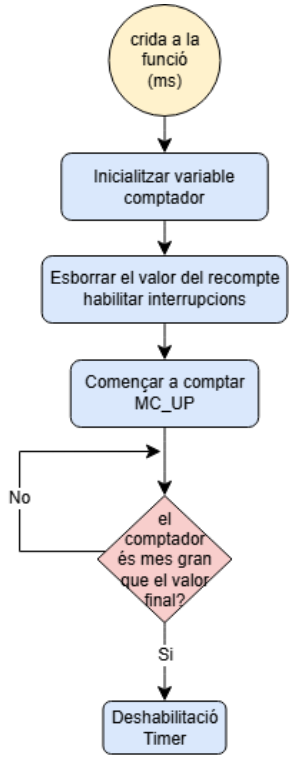


Figura 4: *Diagrama de flux de la funció delay.*

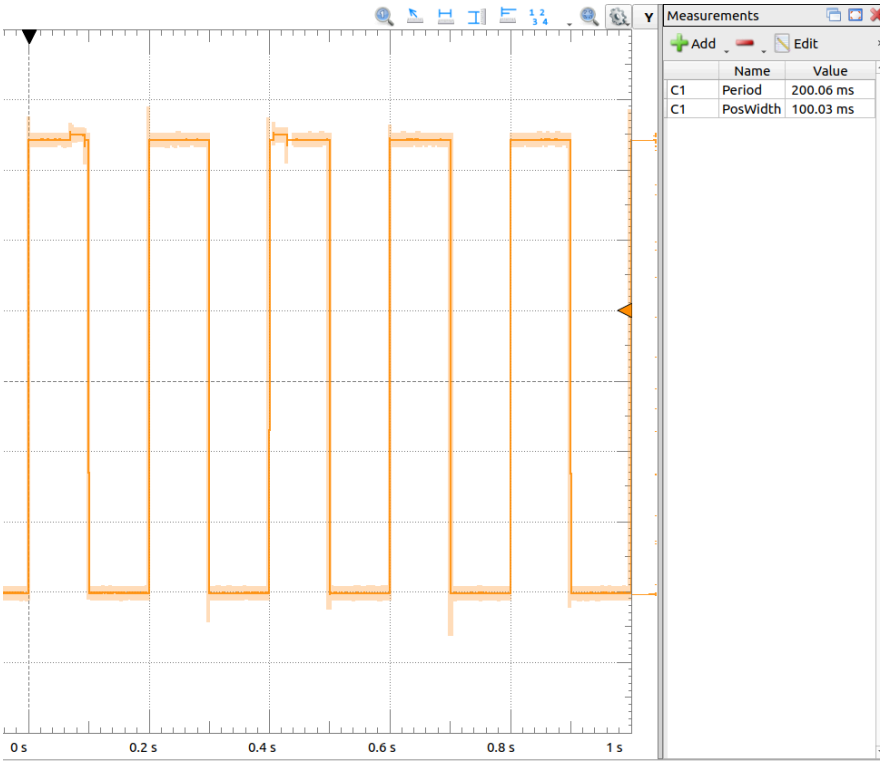


Figura 5*: Visualització del retard generat per la funció delay amb l'oscil·loscopi. S'aprecia a la finestra de mesures que el període d'espera és de 100 ms.*

Timers

Per tal de generar temporitzacions precises en el sistema, s'ha utilitzat el mòdul TimerB3 del microcontrolador. A la funció `init_TimerB()`, es configura aquest temporitzador amb el rellotge SMCLK com a font, funcionant a 16 MHz. S'estableix un valor de recompte de 16000 cicles, que equival a un període d'1 mil·lisegon (tenint en compte que 1 cicle = 1/16 MHz). Aquest temporitzador s'utilitza principalment com a base per a la funció `delay_ms()`, la qual permet generar retards amb precisió en mil·lisegons.

LEDs RGB

Els LEDs frontals del robot es controlen mitjançant trames enviades a través de l'I2C. Cada trama enviada consta de tres bytes: una adreça funcional que identifica el mòdul LED, i dos valors corresponents als colors dels LEDs esquerre i dret. A la **Taula 4** es visualitza la trama enviada al robot i a la **Figura 6** es veu el resultat obtingut a l'analitzador de protocols de l'Analog Discovery 2. Aquest sistema permet encendre els LEDs en diferents combinacions de color. Cada color es codifica amb un número de l'1 al 7, on cada valor representa un color específic com vermell, verd, blau, etc. A la **Taula 5** es recopilen els diferents colors possibles. A la **Figura 7** es presenta un exemple d'aplicació de la funció.

Adreça Robot	Adreça LED	Color1	Color2	Mida de la trama
0x10	0x0B	color LED esq.	color LED dret	3

Taula 4: *Taula amb la trama que s'envia al robot.*

Color	Identificador
Vermell	1
Verd	2
Groc	3
Blau	4
Magenta	5
Cyan	6
Blanc	7

Taula 5: *Taula amb els colors disponibles i els seus identificadors.*

```
Start, h7C [ h3E | WR ], h00, h39, h14, h74, h54, h6F, h0C, h01, Stop
Start, h20 [ h10 | WR ], h0B, h01, h01, Stop
Start, h20 [ h10 | WR ], h0B, h02, h02, Stop
Start, h20 [ h10 | WR ], h0B, h03, h03, Stop
```

Figura 6: *Visualització de la trama I2C dels LEDs amb ajut de l'analitzador de protocols de l'Analog Discovery 2.*

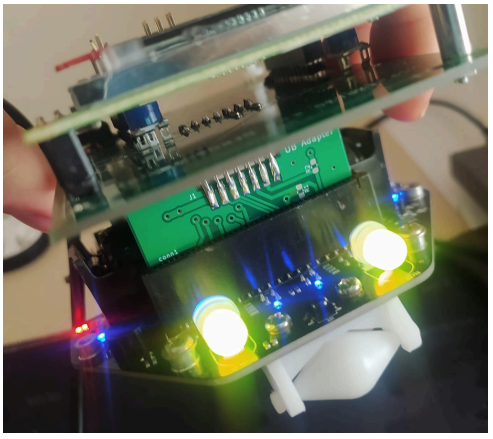


Figura 7: *Visualització dels LEDs de color groc al robot.*

Motors/Actuadors

Els motors del robot es controlen també mitjançant comunicació I2C enviant trames més llargues, de cinc bytes. Aquesta trama que es visualitza a la **Taula 7**, conté una adreça del mòdul motor, la direcció i la velocitat de cada motor (esquerre i dret). La direcció pot ser endavant o enrere, i la velocitat s'expressa amb un valor entre 0 i 255.

Per facilitar el control, s'han implementat funcions que encapsulen aquestes trames. A la **Taula 6** es recopilen les esmentades, junt amb una petita descripció.

Funció	Descripció
`move_forward(uint8_t fvel)`	Ambdós motors actuen en la direcció posterior.
`move_backward(uint8_t bvel)`	Ambdós motors actuen en la direcció anterior.
`fleft(uint8_t fleft_vel)`	Únicament el motor dret actua en la direcció posterior.
`bleft(uint8_t bleft_vel)`	Únicament el motor dret actua en la direcció anterior.
`left_rot(uint8_t lrot_vel)`	Ambdós motors actuen en direccions oposades (motor esquerre enrere, motor dret endavant).
`fright(uint8_t fright_vel)`	Únicament el motor esquerre actua en la direcció posterior.
`bright(uint8_t bright_vel)`	Únicament el motor esquerre actua en la direcció anterior.
`right_rot(uint8_t rrot_vel)`	Ambdós motors actuen en direccions oposades (motor esquerre endavant, motor dret enrere).
`stop(void)`	Deshabilitar ambdós motors (posar la velocitat dels dos a zero).

Taula 6: *Taula amb les funcions desenvolupades pels actuadors amb una descripció*

ID_robot	Adreça controlador motors	Moviment motor esquerre [1]	velocitat motor esquerre	Moviment motor dret [1]	velocitat motor dret	Mida de la trama
0x10	0x00	1 o 2	0-255	1 o 2	0-255	5

Taula 7: *Visualització de la trama que s'envia als motors.*

A la **Figura 8** s'aprecia l'enviament d'instruccions als actuadors via I2C, mitjançant l'analitzador de protocols de l'Analog Discovery 2. Es pot veure que les dades enviades segueixen el mateix ordre que l'especificat a la **Taula 7** anteriorment: identificador del robot, adreça del controlador de motors, direcció de moviment del motor esquerre, velocitat del motor esquerre, direcció del motor dret i velocitat del motor dret.

[1] La direcció del moviment està definida com: direcció posterior = 1, direcció anterior = 2.

```
Start, h20 [ h10 | WR ], h00, h01, h3C, h01, h3C, Stop
Start, h20 [ h10 | WR ], h00, h01, h00, h01, h00, Stop
```

Figura 8: *Visualització de la trama dels motors amb l'Analog Discovery 2.*
(1ª línia: comanda de moure endavant, 2ª línia: comanda d'aturar els motors)

Tanmateix, a les **Figures 9** i **10** es mostren el diagrama de flux del funcionament general de les funcions que componen el mòdul dels actuadors i un exemple d'implementació en software d'una de les funcions de control dels motors, respectivament.

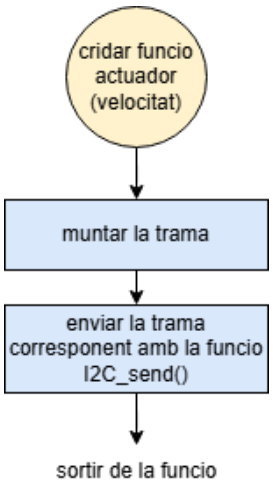


Figura 9: *Visualització del diagrama de flux que segueixen les funcions*

```
void move_forward(uint8_t fvel){
    actuator_buf[0] =
LOC_ACTUATOR_ADDR;
    actuator_buf[1] = FDIR;
    actuator_buf[2] = fvel;
    actuator_buf[3] = FDIR;
    actuator_buf[4] = fvel;
    I2C_send(LOC_ID_ROBOT,
actuator_buf, actuator_blen);
}
```

Figura 10: *Implementació en software de la funció per moure el robot endavant (vegeu **Taula 6**).* [2]

[2] Tant els diferents camps del búffer de dades de la trama `actuator_buf[]` com els arguments de la funció mare i la instrucció `I2C_send()` estan prèviament definits/inicialitzats.

LCD Display

Per controlar el display LCD s'implementen missatges codificats en ASCII. Per fer-ho, s'utilitza un búffer on s'emmagatzema el text mitjançant la funció `sprintf()`, que converteix la cadena de text en caràcters ASCII. El primer byte del buffer és sempre 0x40 ⇒ @, indicant que el que segueix són dades a mostrar. Un cop preparat, el missatge es transmet a través de la funció `I2C_send()`. Quan aquesta transmissió es realitza, és possible observar les trames corresponents de les línies SCL i SDA amb l'oscil·loscopi, mostrant els impulsos de rellotge i les dades del text enviat bit a bit. A la **Figura 8** es mostra una trama genèrica per a enviar missatges al display LCD via el protocol I2C.

Adreça LCD	Búffer amb els caràcters del missatge	Mida del búffer
0x3E	Array de tipus **`uint8_t`**	Depèn del missatge

Taula 8: *Visualització de la trama que s'envia a l'LCD.*

D'altra banda, existeixen també un cert seguit de funcions útils per al control de la pantalla LCD. Aquestes funcions es detallen a la **Figura 11** amb els corresponents diagrames de flux.

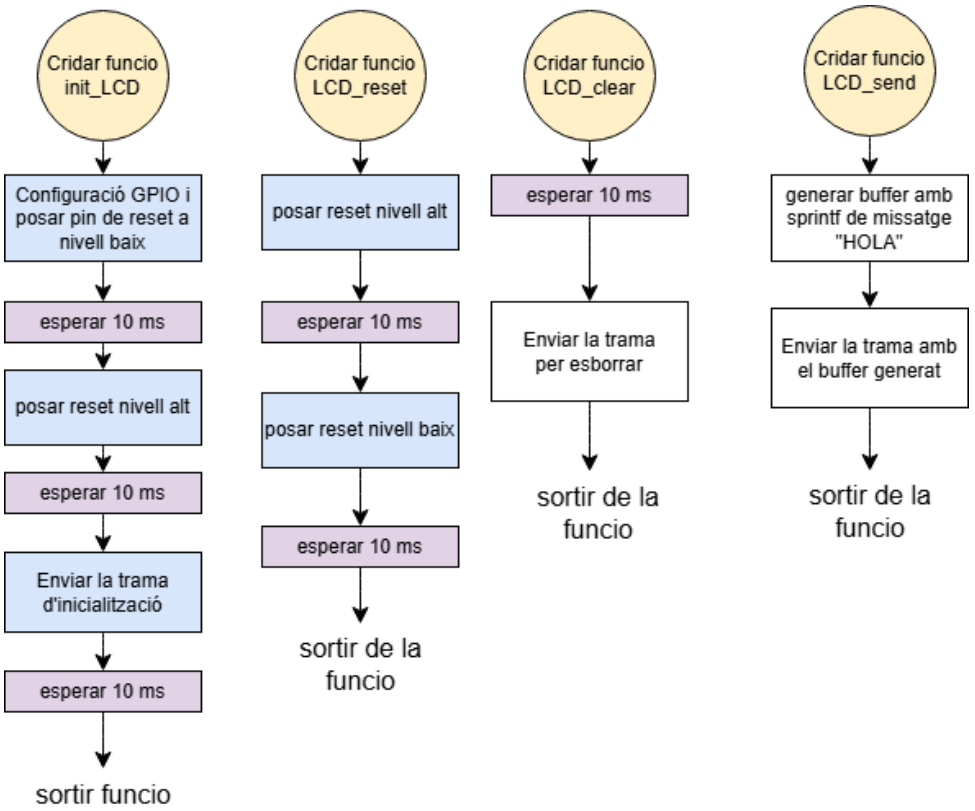


Figura 11: *Diagrames de flux de les diferents funcions per controlar l'LCD.* [3]

[3] La trama d'inicialització es compon dels caràcters 0x00, 0x39, 0x14, 0x74, 0x54, 0x6F, 0x0C i 0x01.

A la **Figura 12** s'analitza la trama d'enviament de dades al display LCD via I2C mitjançant l'analitzador de protocols. A les **Figures 13** i **14** es presenten una imatge de la visualització física de contingut a la pantalla i la implementació d'una de les funcions de control del display, respectivament.

```
Start, h7C [ h3E | WR ], NUL, 9, DC4, t, T, o, FF, SOH, Stop
Start, h7C [ h3E | WR ], @, H, e, l, l, o, , W, o, r, l, d, Stop
```

Figura 12: *Visualització de trama I2C via AD2.*
(1ª línia: inicialització de l'LCD, 2ª línia: missatge «Hello World»)

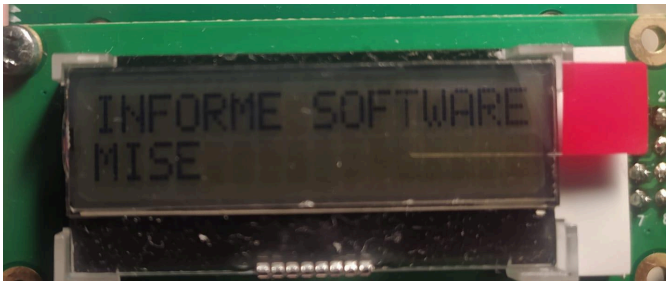


Figura 13: *Imatge amb text escrit a l'LCD.*

```
void LCD_send(){
    llarg = sprintf(cadena,"@INFORME ",llarg);
    I2C_send(0x3E,cadena,llarg);

    llarg = sprintf(cadena,"@SOFTWARE ",llarg);
    I2C_send(0x3E,cadena,llarg);

    uint8_t cursor_segona_linea[2] = {0x00,
0xC0};
    I2C_send(0x3E, cursor_segona_linea, 2);
    delay_ms(10);

    llarg = sprintf(cadena, "@MISE");
    I2C_send(0x3E, (uint8_t*)cadena, llarg);
}
```

Figura 14: *Implementació en software de la funció per a l'enviament de dades a la pantalla LCD.*

Line-Tracking

El mòdul *Line-Tracking* o seguiment de línia permet al robot detectar canvis d'intensitat lluminosa gràcies a uns fotodetectors instal·lats a la part inferior de la placa base del robot. Aquests, en detectar un canvi d'intensitat (per exemple, al passar d'una superfície clara a una superfície fosca) envien un nombre binari de 6 bits que representen quins fotodetectors han detectat un canvi de luminància. Aquests conformen els 6 bits de menor pes del registre d'estat dels fotodetectors. La **Taula 9** mostra el registre d'estat dels fotodetectors.

a7	a6	a5	a4	a3	a2	a1	a0
x	x	r	r	r	r	r	r

Taula 9: *Registre d'estat dels fotodetectors. Únicament de lectura. Útils són els 6 bits de menor pes.*

El diagrama de flux de la funció es mostra a la **Figura 16** i a la **Figura 15** es mostra el robot seguint la línia.

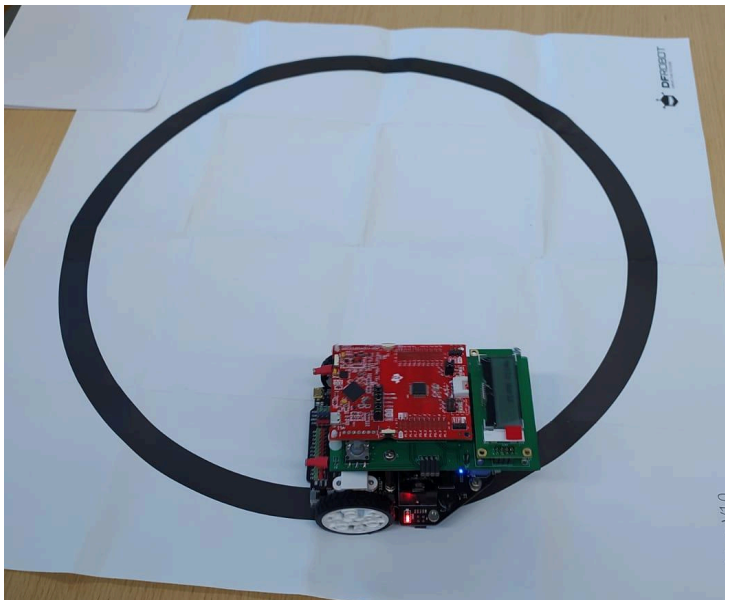


Figura 15: *Visualització del robot funcionant amb el linetracking*

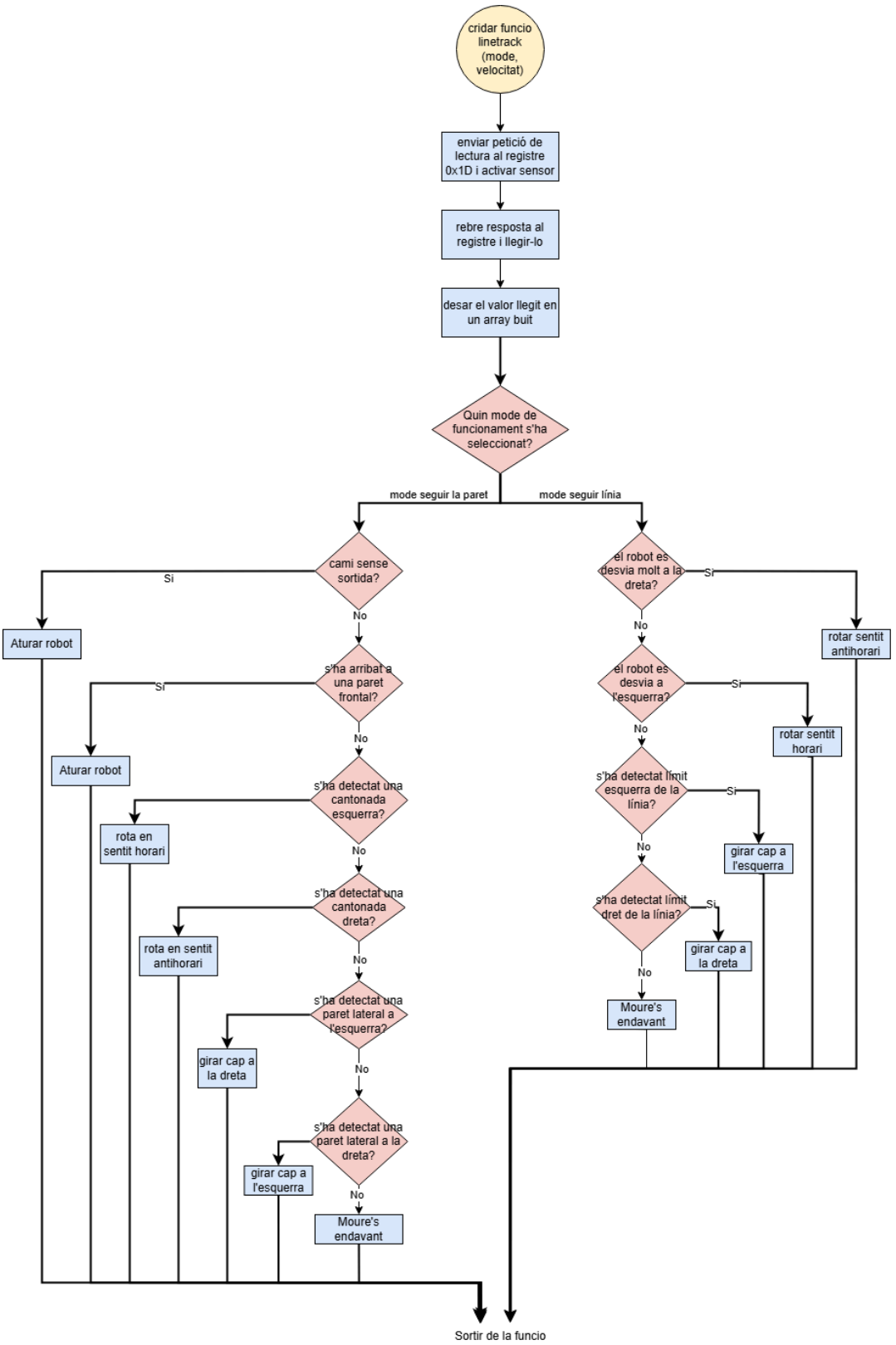


Figura 16: *Diagrama de flux de la funció linetracking()*

CONCLUSIONS

En aquest informe s'ha explicat el funcionament del codi de control implementat en el robot dissenyat anteriorment. Integra múltiples perifèrics que venien inclosos al DFRobot Mcqueen Plus V2 i recursos externs a través del bus I2C. El desenvolupament del software ha permès entendre el funcionament del BUS I2C a escala de capa d'enllaç, generant trames definides per usuari amb l'adreça de l'esclau, la direcció de comunicació (escriptura o lectura), les dades i bits de confirmació (NACK i ACK).

S'ha pogut comprovar l'ús del bus I2C per controlar dispositius externs amb trames específiques, i s'ha verificat que la resposta del sistema és correcte i coherent amb **l'oscil·loscopi i l'AD2**. Sempre que es tinguin en compte aspectes crítics com la sincronització i les esperes necessàries després de cada transmissió.

El projecte ha servit per aplicar els coneixements teòrics de programació en llenguatge C per a la placa dissenyada amb anterioritat. Finalment, l'estructura en blocs del codi afavoreix l'escalabilitat del projecte, cosa que permet afegir noves funcionalitats de manera senzilla.

ANNEXOS

DELAY

```
/*
 * delay_ms.c
 *
 * Microcontroladors i Sistemes Empotrats
 * Curs 2024-25
 * Universitat de Barcelona
 *
 * Autors: David Garcia, Aidar Iglesias
 */


/****************************** LLIBRERIES
*************************************/
#include <msp430fr2355.h>
#include <stdint.h>
#include "timers.h"
/*****************************************************************************
*********/


/************************** Variables de control
*******************************/
uint16_t count_ms = 0;
/*****************************************************************************
*********/


/*************************** Funció delay_ms
***********************************/
/* La funcio delay_ms accepta un unic argument de tipus enter de 8 bits
i no retorna
 * cap dada (tipus void). L'argument que se li passa a la funcio es la
quantitat de
 * temps d'espera (en mil·lisegons) que es desitja fer.
 *
 * La funcio explota el Timer B3 com a temporitzador per al recompte del
pas del temps.
 * A l'inici es fa la inicialitzacio del temporitzador: s'esborra el
valor emmagat-
```

```
 * zemat al registre de recompte, s'habiliten les interrupcions de
comptador i s'ha-
 * bilita el temporitzador. Un cop acabat el recompte, es deshabilita el
temporitzador.
 *
 * El recompte es controla mitjançant una variable de recompte que
s'incrementa mitjan-
 * çant una interrupcio, la qual es genera cada cop que el temporitzador
finalitza el
 * periode d'un mil·lisegon.
 */
void delay_ms(uint16_t ms){  // Funció d'espera de milisegons
    count_ms = 0;                       // Inicialitzem la variable
count_ms a 0
    TB3CTL |= TBCLR;                    // Esborrem el valor de recompte
    TB3CCTL0 |= CCIE;                   // Habilitem les interrupcions
del comptador del TimerB3
    TB3CTL |= MC__UP;                   // Comencem el recompte
    while(count_ms < ms){
        // cos del bucle buit (comptant...)
    }
    TB3CTL &= ~(MC__UP);                // Deshabilitem el TimerB3
}
/*********************************************************************
*********/


/********************** Inicialització ISR del TimerB3
************************/
#pragma vector = TIMER3_B0_VECTOR
__interrupt void TimerB0_ISR (void){
    count_ms += 1; // Incrementem la variable de recompte en 1 i
deshabilitem les interrupcions.
}
/*********************************************************************
*********/
```

LCD

```
/*
 * LCD_display.c
 *
 * Microcontroladors i Sistemes Empotrats
 * Curs 2024-25
 * Universitat de Barcelona
 *
 * Autors: David Garcia, Aidar Iglesias
 */


/******************************** LLIBRERIES
*************************************/
#include <msp430fr2355.h>
#include <stdint.h>
#include <stdio.h>
#include "i2c_master.h"
#include "delay_ms.h"
/*****************************************************************************
*************/


/*************************** Variables de control
*********************************/
#define LOC_ID_ROBOT 0x10  // Variable local de l'identificador del
robot
#define WAIT_10MS 10       // Variable de delay d'inicialització (10ms)
#define LOC_LCD_ADDR 0x3E  // Variable local adreça del LCD
uint8_t LCD_init_buf[8] = {0x00, 0x39, 0x14, 0x74, 0x54, 0x6F, 0x0C,
0x01};  // Búffer d'inicialització del LCD
const uint8_t LCD_init_blen = sizeof(LCD_init_buf);     // Longitud del
búffer d'inicialització
uint8_t LCD_clear_buf[2] = {0x00, 0x01};                // Búffer per a
esborrar el contingut en pantalla
const uint8_t LCD_clear_blen = sizeof(LCD_clear_buf);   // Longitud del
búffer d'esborrar contingut en pantalla
uint8_t LCD_cursor_reset_buf[2] = {0x00, 0x03};         // Búffer per a
posicionar el cursor a l'inici de línia
const uint8_t LCD_cursor_reset_blen = sizeof(LCD_cursor_reset_buf); //
Longitud del búffer per a posicionar el cursor a l'inici de línia
/*****************************************************************************
```

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


/******************* Funció d'inicialització del display LCD
************************/
/* La funcio 'init_LCD' s'encarrega d'inicialitzar els parametres del
display LCD
 * d'acord amb les seves caracteristiques fisiques (2x16 caracters) i
electriques
 * (3.3V d'alimentacio).
 *
 * En concret, s'habilita el senyal de reset de la pantalla LCD (GPIO
del microcon-
 * trolador) durant un cert temps especificat pel fabricant i despres de
deshabili-
 * tar-lo, s'envia una seqüencia de caracters per I2C que el configuren.
 */
void init_LCD(void){
    P5DIR |= BIT2;          // Configurem el pin GPIO del reset del LCD
com a sortida (actiu per nivell baix
    P5OUT &= ~(BIT2);       // Posem el pin de reset a nivell lògic baix
(habilitem reset)
    delay_ms(WAIT_10MS);    // Esperem un temps de configuració
(requeriment del fabricant)
    P5OUT |= BIT2;          // Posem el pin de reset a nivell lògic alt
(deshabilitem reset)
    delay_ms(WAIT_10MS);    // Esperem un temps de configuració
(requeriment del fabricant)
    I2C_send(LOC_LCD_ADDR, LCD_init_buf, LCD_init_blen);  // Enviem la
trama d'inicialització per I2C
    delay_ms(WAIT_10MS);    // Esperem un temps de configuració
(requeriment del fabricant)
}
/*************************************************************************
*************/


/************************ Funció de reset del display LCD
*************************/
/* La funcio 'LCD_reset' envia un senyal de reset al display LCD. La
utilitat de la
 * funcio es desfer-se de la tasca de treballar a nivell de GPIOs. Amb
nomes cridar
```

```
 * la funcio ja es fa el reset.
 */
void LCD_reset(void){
    P5OUT &= ~(BIT2);       // Habilitem reset
    delay_ms(WAIT_10MS);    // Esperem 10ms
    P5OUT |= BIT2;          // Deshabilitem reset
    delay_ms(WAIT_10MS);    // Esperem 10ms
}
/**************************************************************************
*************/


/****************** Funció per esborrar el contingut del display LCD
***************/
/* La funcio 'LCD_clear' te com a finalitat esborrar el contingut
presentat a la pan-
 * talla. La posicio del cursor tambe retorna al primer caracter de la
primera linia.
 */
void LCD_clear(void){
    delay_ms(WAIT_10MS);  // Esperem 10ms
    I2C_send(LOC_LCD_ADDR, LCD_clear_buf, LCD_clear_blen);  // Enviem
comanda d'esborrament en
        // pantalla per I2C
}
/**************************************************************************
*************/


/********** Funció per posar el cursor del display LCD a la posició
inicial**********/
/* La funcio 'LCD_cursor_reset' s'empra per, en el moment que es
desitgi, retornar el
 * cursor a la posicio inicial, es a dir, al primer caracter de la
primera linia.
 */
void LCD_cursor_reset(void){
    delay_ms(WAIT_10MS);  // Esperem 10ms
    I2C_send(LOC_LCD_ADDR, LCD_cursor_reset_buf, LCD_cursor_reset_blen);
// Enviem comanda de
        // posicionament del cursor a l'inici de la primera línia del
display
}
```

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


/********** Funcio sprintf per posar missatge "INFORME SOFTWARE MISE"
***************/
/* La funcio 'LCD_send' envia la cadena "INFORME SOFTWARE MISE" a la
pantalla LCD.
 * Per fer-ho, es construeixen les diferents cadenes corresponents a
cada paraula
 * de la frase i s'envien via I2C al display.
 */
void LCD_send(void){
    uint8_t cadena[10]; // Declarem un array per desar la cadena de
caracters
    uint8_t llarg;      // Declarem la mida de la cadena de caracters

    /* Construim la cadena de caracters mitjançant la funcio 'sprintf'
de la
     * llibreria <stdio.h> de C
     */
    llarg = sprintf(cadena,"@INFORME ",llarg);
    I2C_send(LOC_LCD_ADDR, cadena, llarg);      // Enviem la cadena per
I2C al LCD

    // Repetim el mateix per a la següent paraula
    llarg = sprintf(cadena,"@SOFTWARE ",llarg);
    I2C_send(LOC_LCD_ADDR, cadena, llarg);      // Enviem la cadena per
I2C al LCD

    /* Enviem una cadena de caracters que corresponen a un salt de linia
     * (vegeu app. note del fabricant per a les trames de control)
     */
    uint8_t cursor_segona_linea[2] = {0x00, 0xC0};
    I2C_send(LOC_LCD_ADDR, cursor_segona_linea, 2); // Enviem la cadena
per I2C al LCD
    delay_ms(WAIT_10MS);                            // Esperem 10ms

    // Construim la cadena per a la ultima paraula de la frase
    llarg = sprintf(cadena, "@MISE");
    I2C_send(LOC_LCD_ADDR, (uint8_t*)cadena, llarg); // Enviem la cadena
per I2C al LCD
}
```

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

LEDs

```
/*
 * LEDs.c
 *
 * Microcontroladors i Sistemes Empotrats
 * Curs 2024-25
 * Universitat de Barcelona
 *
 * Autors: David Garcia, Aidar Iglesias
 */


/******************************** LLIBRERIES
*************************************/
#include <msp430fr2355.h>
#include <stdint.h>
#include "timers.h"
#include "delay_ms.h"
#include "i2c_master.h"
/*********************************************************************
*************/


/*************************** Variables de control
*********************************/
// LEDs RGB robot
// colors: 1-->RED, 2-->GREEN, 3-->YELLOW, 4-->BLUE, 5-->MAGENTA,
6-->CYAN, 7-->WHITE
#define OFF 0
#define RED 1
#define GREEN 2
#define YELLOW 3
#define BLUE 4
#define MAGENTA 5
#define CYAN 6
#define WHITE 7
// Paràmetres I2C dels LEDs
#define LOC_ID_ROBOT 0x10   // Adreça del robot
#define LED_addr 0x0B       // Adreça del controlador dels LEDs RGB del
```

```
robot
#define LED_BLEN 3          // Longitud del buffer de dades dels LEDs
// Trames per a cada funció (el funcionament no es limita a les
presentades)
uint8_t LEDs_OFF_buffer[3] = {LED_addr, OFF, OFF};
uint8_t RED_LEDs_buffer[3] = {LED_addr, RED, RED};
uint8_t GREEN_LEDs_buffer[3] = {LED_addr, GREEN, GREEN};
uint8_t YELLOW_LEDs_buffer[3] = {LED_addr, YELLOW, YELLOW};
uint8_t BLUE_LEDs_buffer[3] = {LED_addr, BLUE, BLUE};
uint8_t MAGENTA_LEDs_buffer[3] = {LED_addr, MAGENTA, MAGENTA};
uint8_t CYAN_LEDs_buffer[3] = {LED_addr, CYAN, CYAN};
uint8_t WHITE_LEDs_buffer[3] = {LED_addr, WHITE, WHITE};
uint8_t CULE_LEDs_buffer[3] = {LED_addr, BLUE, RED};
uint8_t MERENGUE_LEDs_buffer[3] = {LED_addr, WHITE, WHITE};
uint8_t PERICO_LEDs_buffer[3] = {LED_addr, WHITE, BLUE};
uint8_t GIRONI_LEDs_buffer[3] = {LED_addr, RED, WHITE};
/*****************************************************************
*************/


/********************** Funcions de control dels LEDs
*****************************/
/* Funcions per a l'activació dels LEDs RGB del robot. Les funcions
envien una coman-
 * da I2C per encendre els LEDs desitjats amb el color especificat.
 *
 * Cal destacar que en aquesta llibreria es recullen unicament funcions
que encenen
 * ambdos LEDs RGB del robot del mateix color. En cas de desitjar una
altra combinacio
 * de colors, s'han de definir la trama i la funcio corresponents, les
quals permeten
 * aquest comportament. Alternativament, es pot fer l'enviament
d'instruccions al cos
 * del programa main.c directament.
 */
void reset_LEDs(void){  // Funció per a apagar els LEDs
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, LEDs_OFF_buffer, LED_BLEN);
}

void red_LEDs(void){    // Funció per a encendre els LEDs de color
vermell
```

```
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, RED_LEDs_buffer, LED_BLEN);
}

void green_LEDs(void){  // Funció per a encendre els LEDs de color verd
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, GREEN_LEDs_buffer, LED_BLEN);
}

void yellow_LEDs(void){ // Funció per a encendre els LEDs de color groc
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, YELLOW_LEDs_buffer, LED_BLEN);
}

void blue_LEDs(void){   // Funció per a encendre els LEDs de color blau
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, BLUE_LEDs_buffer, LED_BLEN);
}

void magenta_LEDs(void){ // Funció per a encendre els LEDs de color
magenta
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, MAGENTA_LEDs_buffer, LED_BLEN);
}

void cyan_LEDs(void){   // Funció per a encendre els LEDs de color cyan
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, CYAN_LEDs_buffer, LED_BLEN);
}

void white_LEDs(void){  // Funció per a encendre els LEDs de color blanc
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, WHITE_LEDs_buffer, LED_BLEN);
}

void cule_LEDs(void){  // Funció per a encendre els LEDs de color
blaugrana
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, CULE_LEDs_buffer, LED_BLEN);
}

void merengue_LEDs(void){  // Funció per a encendre els LEDs de color
merengue
```

```
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, MERENGUE_LEDs_buffer, LED_BLEN);
}

void perico_LEDs(void){  // Funció per a encendre els LEDs de color
blanc
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, PERICO_LEDs_buffer, LED_BLEN);
}

void gironi_LEDs(void){  // Funció per a encendre els LEDs de color
gironi
    delay_ms(1);
    I2C_send(LOC_ID_ROBOT, GIRONI_LEDs_buffer, LED_BLEN);
}
/*************************************************************************
*************/
```

ACTUADORS

```
/*
 * actuadors.c
 *
 * Microcontroladors i Sistemes Empotrats
 * Curs 2024-25
 * Universitat de Barcelona
 *
 * Autors: David Garcia, Aidar Iglesias
 */

/***************************** MACROS I LLIBRERIES
*********************************/
#include <msp430fr2355.h>
#include <stdint.h>
#include "i2c_master.h"
/***************************************************************************
*************/


/***************************** Variables de control
*********************************/
#define LOC_ID_ROBOT 0x10       // Identificador local del robot (a
nivell de llibreria)
#define LOC_ACTUATOR_ADDR 0x00  // Adreça local dels actuadors (a nivell
de llibreria)
#define FDIR 1                  // Direcció de moviment posterior
#define BDIR 2                  // Direcció de moviment anterior
#define STOP_WHEEL 0            // Variable que defineix l'estat de
rodes aturades
uint8_t actuator_buf[5];        // Búffer que emmagatzema l'estat dels
actuadors
uint8_t actuator_blen = sizeof(actuator_buf);  // Longitud del búffer
d'estat dels actuadors
/***************************************************************************
*************/


/************************ Funcions per al moviment del robot
***********************/
// Funció per a aturar el robot
void stop(void){
```

```
    actuator_buf[0] = LOC_ACTUATOR_ADDR;
    actuator_buf[1] = FDIR;
    actuator_buf[2] = STOP_WHEEL;
    actuator_buf[3] = FDIR;
    actuator_buf[4] = STOP_WHEEL;
    I2C_send(LOC_ID_ROBOT, actuator_buf, actuator_blen);
}

// Funció per a moure endavant el robot
void move_forward(uint8_t fvel){
    actuator_buf[0] = LOC_ACTUATOR_ADDR;
    actuator_buf[1] = FDIR;
    actuator_buf[2] = fvel;
    actuator_buf[3] = FDIR;
    actuator_buf[4] = fvel;
    I2C_send(LOC_ID_ROBOT, actuator_buf, actuator_blen);
}

// Funció per a moure enrere el robot
void move_backward(uint8_t bvel){
    actuator_buf[0] = LOC_ACTUATOR_ADDR;
    actuator_buf[1] = BDIR;
    actuator_buf[2] = bvel;
    actuator_buf[3] = BDIR;
    actuator_buf[4] = bvel;
    I2C_send(LOC_ID_ROBOT, actuator_buf, actuator_blen);
}

// Funció per a girar cap a l'esquerra endavant
void fleft(uint8_t fleft_vel){
    actuator_buf[0] = LOC_ACTUATOR_ADDR;
    actuator_buf[1] = FDIR;
    actuator_buf[2] = STOP_WHEEL;
    actuator_buf[3] = FDIR;
    actuator_buf[4] = fleft_vel;
    I2C_send(LOC_ID_ROBOT, actuator_buf, actuator_blen);
}

// Funció per a girar cap a l'esquerra enrere
void bleft(uint8_t bleft_vel){
    actuator_buf[0] = LOC_ACTUATOR_ADDR;
    actuator_buf[1] = FDIR;
    actuator_buf[2] = STOP_WHEEL;
```

```
    actuator_buf[3] = BDIR;
    actuator_buf[4] = bleft_vel;
    I2C_send(LOC_ID_ROBOT, actuator_buf, actuator_blen);
}

// Funció per a rotar en sentit antihorari
void left_rot(uint8_t lrot_vel){
    actuator_buf[0] = LOC_ACTUATOR_ADDR;
    actuator_buf[1] = BDIR;
    actuator_buf[2] = lrot_vel;
    actuator_buf[3] = FDIR;
    actuator_buf[4] = lrot_vel;
    I2C_send(LOC_ID_ROBOT, actuator_buf, actuator_blen);
}

// Funció per a girar cap a la dreta endavant
void fright(uint8_t fright_vel){
    actuator_buf[0] = LOC_ACTUATOR_ADDR;
    actuator_buf[1] = FDIR;
    actuator_buf[2] = fright_vel;
    actuator_buf[3] = FDIR;
    actuator_buf[4] = STOP_WHEEL;
    I2C_send(LOC_ID_ROBOT, actuator_buf, actuator_blen);
}

// Funció per a girar cap a la dreta enrere
void bright(uint8_t bright_vel){
    actuator_buf[0] = LOC_ACTUATOR_ADDR;
    actuator_buf[1] = BDIR;
    actuator_buf[2] = bright_vel;
    actuator_buf[3] = FDIR;
    actuator_buf[4] = STOP_WHEEL;
    I2C_send(LOC_ID_ROBOT, actuator_buf, actuator_blen);
}

// Funció per a rotar en sentit horari
void right_rot(uint8_t rrot_vel){
    actuator_buf[0] = LOC_ACTUATOR_ADDR;
    actuator_buf[1] = FDIR;
    actuator_buf[2] = rrot_vel;
    actuator_buf[3] = BDIR;
    actuator_buf[4] = rrot_vel;
    I2C_send(LOC_ID_ROBOT, actuator_buf, actuator_blen);
```

```
}
/**************************************************************************
*************/
```

LINETRACKING

```
/*
 * linetracking.c
 *
 * Microcontroladors i Sistemes Empotrats
 * Curs 2024-25
 * Universitat de Barcelona
 *
 * Autors: David Garcia, Aidar Iglesias
 */


/******************************** LLIBRERIES
*************************************/
#include <msp430fr2355.h>
#include <stdint.h>
#include "timers.h"
#include "LCD_display.h"
#include "actuadors.h"
#include "delay_ms.h"
#include "i2c_master.h"
/**************************************************************************
*************/


/****************************** VARIABLES DE CONTROL
*******************************/
#define LOC_ID_ROBOT 0x10       // Adreça I2C del robot
#define LINE_TRACK_ADDR 0x1D    // Adreça del controlador del modul de
seguiment de linia
#define WAIT_10MS 10            // Delay de 10ms entre la transmisio i
la rececpcio del line-track
uint8_t linetrack_reg[6];       // Variable que emmagatzema els bits
d'estat de cadascun dels fotodetectors
/**************************************************************************
*************/
```

```
/****************** Funcio d'execucio del mode de seguiment de linia
***************/
/* La funcio de seguiment de linia 'linetrack' accepta dos arguments. El
primer es
 * el mode de funcionament, el format del qual ve donat per un nombre
enter de 8 bits.
 * El segon argument es la velocitat del robot durant l'etapa de
seguiment de linia.
 * No retorna cap dada (tipus void).
 *
 * La funcio esta dividida en dos fases: la fase de mesura i la de presa
de decisio.
 * Durant la fase de presa de mesura s'envia la comanda de mesura via
I2C al contro-
 * lador dels fotodetectors. Un cop feta la mesura, les dades d'aquesta
es processen
 * i es preparen per a la següent fase. Durant la fase de presa de
decisio s'envien
 * comandes d'operacio als actuadors del robot en funcio del valor dels
fotodetectors.
 *
 * La funcio te dos modes principals de funcionament: seguiment de linia
i seguiment de
 * paret. En el mode de seguiment de linia, el robot tractara de seguir
la linia amb
 * la roda passiva davantera, si mes no mantenir la linia entre les dues
rodes actives.
 * En el mode de seguiment de paret, el robot avançara fins que es trobi
una paret, cas
 * en el qual rotara cap a la dreta fins que detecti que no col·lisiona,
aleshores
 * avançara mantenint'se al marge d'aquesta.
 *
 * Els punts febles del robot son els camins sense sortida (tots els
fotodetectors
 * detecten una col·lisio imminent) i que en el mode de seguiment de
paret, el robot
 * evita la col·lisio, pero no mante una distancia constant amb la
paret.
 */
void linetrack(uint8_t mode, uint8_t linetrack_vel) {
```

```
    /********************** Fase de mesura dels fotodetectors
**********************/
    // Inicialitzar els buffers de transmisio i recepcio
    uint8_t tx_buf[1] = {LINE_TRACK_ADDR};  // Byte a enviar per activar
el sensor
    uint8_t rx_buf[1] = {0x00};             // Byte de recepció
inicialitzat a 0
    uint8_t tx_buf_len = sizeof(tx_buf);    // Longitud del buffer de
transmissio
    uint8_t rx_buf_len = sizeof(rx_buf);    // Longitud del buffer de
recepcio

    // Enviar la peticio de mesura al registre del modul line-track
    I2C_send(LOC_ID_ROBOT, tx_buf, tx_buf_len);
    delay_ms(WAIT_10MS);

    // Rebre i desar la resposta
    I2C_receive(LOC_ID_ROBOT, rx_buf, rx_buf_len);
    linetrack_reg[0] = rx_buf[0] & ((uint8_t)BIT0);        // Extraiem i
desem els bits del registre
    linetrack_reg[1] = (rx_buf[0] & ((uint8_t)BIT1)) >> 1; // dels
fotodetectors mitjançant una mascara
    linetrack_reg[2] = (rx_buf[0] & ((uint8_t)BIT2)) >> 2; // de bit i
desplaçaments
    linetrack_reg[3] = (rx_buf[0] & ((uint8_t)BIT3)) >> 3;
    linetrack_reg[4] = (rx_buf[0] & ((uint8_t)BIT4)) >> 4;
    linetrack_reg[5] = (rx_buf[0] & ((uint8_t)BIT5)) >> 5;

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


    /************************** Fase de presa de decisio
***************************/
    switch(mode){

    case 0: // Mode de funcionament: seguiment de linia
(LINE_TRACK_MODE)
        if(linetrack_reg[0]){
            /* excrusio excessiva cap a la dreta --> rotar en sentit
antihorari */
            delay_ms(5);
            left_rot(linetrack_vel);
```

```
            delay_ms(5);
        } else if(linetrack_reg[5]){
            /* excursio excessiva cap a l'esquerra --> rotar en sentit
horari */
            delay_ms(5);
            right_rot(linetrack_vel);
            delay_ms(5);
        } else if(linetrack_reg[1] | linetrack_reg[2]){
            /* s'ha detectat limit esquerre de la linia --> girar cap a
l'esquerra */
            delay_ms(5);
            fleft(linetrack_vel);
            delay_ms(5);
        } else if(linetrack_reg[3] | linetrack_reg[4]){
            /* s'ha detectat limit dret de la linia --> girar cap a la
dreta */
            delay_ms(5);
            bright(linetrack_vel);
            delay_ms(5);
        } else{
            /* ens trobem dintre dels limits --> avançar */
            delay_ms(5);
            move_forward(linetrack_vel);
            delay_ms(5);
        }
        break;

    case 1: // Mode de funcionament seguiment de paret
(WALL_FOLLOWER_MODE)
        if(linetrack_reg[5] & linetrack_reg[4] & linetrack_reg[3] &
linetrack_reg[2] & linetrack_reg[1] & linetrack_reg[0]){
            /* s'ha arribat a un cami sense sortida --> aturem el
robot*/
            delay_ms(5);
            stop();
            delay_ms(5);
        } else if(linetrack_reg[4] & linetrack_reg[3] & linetrack_reg[2]
& linetrack_reg[1]){
            /* s'ha arribat a una paret frontal --> aturem el robot*/
            delay_ms(5);
            stop();
            delay_ms(5);
            right_rot(linetrack_vel);
```

```
            delay_ms(500);
            stop();
            delay_ms(5);
        } else if(linetrack_reg[1] & linetrack_reg[0]){
            /* s'ha detectat una cantonada esquerra --> rota en sentit
horari*/
            delay_ms(5);
            right_rot(linetrack_vel);
            delay_ms(5);
        } else if(linetrack_reg[5] & linetrack_reg[4]){
            /* s'ha detectat una cantonada dreta --> rota en sentit
antihorari*/
            delay_ms(5);
            left_rot(linetrack_vel);
            delay_ms(5);
        } else if(linetrack_reg[0]){
            /* s'ha detectat una paret lateral a l'esquerra --> gira cap
a la dreta */
            delay_ms(5);
            fright(linetrack_vel);
            delay_ms(5);
        } else if(linetrack_reg[5]){
            /* s'ha detectat una paret lateral a la dreta --> gira cap a
l'esquerra */
            delay_ms(5);
            fleft(linetrack_vel);
            delay_ms(5);
        } else{
            /* en qualsevol altre cas --> avança */
            delay_ms(5);
            move_forward(0.8*linetrack_vel);
            delay_ms(5);
        }
        break;
    }

/*************************************************************************
*********/
}
/*************************************************************************
************/
```

MAIN

```
/*
 * main.c
 *
 * Microcontroladors i Sistemes Empotrats
 * Curs 2024-25
 * Universitat de Barcelona
 *
 * Autors: David Garcia, Aidar Iglesias
 */


/********************************* LLIBRERIES
**********************************/
// Llibreries estandard de C i del microcontrolador
#include <msp430fr2355.h>
#include <stdint.h>
#include <stdio.h>
// Fitxers de capçalera
#include "timers.h"
#include "delay_ms.h"
#include "i2c_master.h"
#include "LEDs.h"
#include "LCD_display.h"
#include "actuadors.h"
#include "linetracking.h"
/************************************************************************
*********/


/***************************** Variables globals
*******************************/
/* En aquest apartat es defineixen tant les variables globals com les
macros emprades
 * en els diferents moduls del programa
 */
// Variables delay
#define WAIT_1MS 1              // Espera d'un mil·lisegon
#define WAIT_100MS 100          // Espera de cent mil·lisegons
#define WAIT_HALF_SEC 500       // Espera de mig segon
#define WAIT_1S 1000            // Espera d'un segon
#define WAIT_2S 2000            // Espera de dos segons
```

```
// ID del robot com a esclau
#define GLOB_ID_ROBOT 0x10      // Identificador del robot

// LCD display robot
#define GLOB_LCD_ADDR 0x3E      // Adreça del display LCD
#define LCD_TX_DELAY 10         // Espera entre missatges enviats al LCD

// Actuadors
#define GLOB_ACTUATOR_ADDR 0x00 // Adreça global del controlador
d'actuadors
#define VEL 40                  // Velocitat inicial dels actuadors

// Line-Track
#define LINETRACK_VEL 40        // Velocitat del robot en mode
line-track
/* Creem un bloc condicional a nivell de compilador per escollir el mode
de funcionament
 * del modul line-track amb les macros #ifndef, #define i #endif
 */
#define LINE_TRACK_MODE 0       // Mode de funcionament: seguiment de
linia centrada
#ifndef LINE_TRACK_MODE
#define WALL_FOLLOWER_MODE 1    // Mode de funcionament: seguiment de
paret
#endif
/**********************************************************************
*********/


/************************ Funcio inicialitzacio de rellotges
********************/
void init_clock(void)
{
    FRCTL0 = FRCTLPW | NWAITS_1;

    P2SEL1 |= BIT6 | BIT7;              // P2.6~P2.7: crystal pins
    do
    {
        CSCTL7 &= ~(XT1OFFG | DCOFFG);  // Clear XT1 and DCO fault flag
        SFRIFG1 &= ~OFIFG;
    }while (SFRIFG1 & OFIFG);           // Test oscillator fault flag
```

```
    __bis_SR_register(SCG0);                // disable FLL
    CSCTL3 |= SELREF__XT1CLK;               // Set XT1 as FLL reference
source
    //CSCTL1 = DCOFTRIMEN_1 | DCOFTRIM0 | DCOFTRIM1 | DCORSEL_5; //
DCOFTRIM=5, DCO Range = 16MHz**
    CSCTL1 = DCORSEL_5;                     // DCOFTRIM=5, DCO Range = 16MHz
    CSCTL2 = FLLD_0 + 487;                  // DCOCLKDIV = 16MHz
    __delay_cycles(3);
    __bic_SR_register(SCG0);                // enable FLL
    //Software_Trim();                      // Software Trim to get the best
DCOFTRIM value**

    CSCTL4 = SELMS__DCOCLKDIV | SELA__XT1CLK;   // set XT1 (~32768Hz) as
ACLK source, ACLK = 32768Hz
                                            // default DCOCLKDIV as
MCLK and SMCLK source

    // P1DIR |= BIT0 | BIT1;                // set SMCLK, ACLK pin as output
    // P1SEL1 |= BIT0 | BIT1;               // set SMCLK and ACLK pin as
second function
    PM5CTL0 &= ~LOCKLPM5;                   // Disable the GPIO power-on
default high-impedance mode
                                            // to activate previously
configured port settings
}
/**********************************************************************
*********/


/*********** Funcio inicialitzacio LED P1.0 de la placa Launchpad
***************/
void init_LED_MSP(){
    P1DIR |= BIT0;      // LED P1.0 com a sortida
    P1SEL0 &= ~(BIT0);  // Funcio principal del
    P1SEL1 &= ~(BIT0);  // pin com a GPIO
}
/**********************************************************************
*********/


/*********** Funcio inicialitzacio pin P2.0 de la placa Launchpad
***************/
void init_Port2IO(void){
```

```
    P2DIR |= BIT0;       // P2.0 com a sortida
    P2SEL0 &= ~(BIT0);   // Funcio principal del
    P2SEL1 &= ~(BIT0);   // pin com a GPIO
}
/**********************************************************************
*********/


/************************ BEGIN MAIN PROGRAM
**********************************/
void main(void)
{
      WDTCTL = WDTPW | WDTHOLD;     // stop watchdog timer

    /******************** Habilitar interrupcions globals
***********************/
    __enable_interrupt();

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

      /************************ Inicialitzacio de funcions
************************/
      // Inicialitzacio de rellotges
      init_clock();
      // Inicialitzacio de l'I2C
      i2c_init();
      // Inicialitzacio del LED P1.0 de la placa (nomes al Launchpad)
      init_LED_MSP();
      // Inicialitzacio del pin P2.0 de la placa (nomes al Launchpad)
      init_Port2IO();
      // Inicialitzacio del TimerB3
      init_TimerB3();
      // Inicialitzacio del display LCD
      init_LCD();

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


      /****************** Comprovacio de la funcio
delay_ms()**********************/
      // Comprovar la funcio delay_ms()
```

```
        P1OUT |= BIT0;      // Encenem el LED vermell P1.0
        P2OUT |= BIT0;      // Posem a nivell logic alt el pin 2.0
        delay_ms(WAIT_1S);  // Esperem un segon
        P1OUT &= ~(BIT0);   // Apaguem el LED vermell P1.0
        P2OUT &= ~(BIT0);   // Posem a nivell logic baix el pin 2.0

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


        /********************* Control de LEDs RGB del robot
************************/
        // Escriptura als LEDs RGB frontals del robot
        red_LEDs();                 // Encendre els LEDs de color vermell
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        green_LEDs();               // Encendre els LEDs de color verd
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        yellow_LEDs();              // Encendre els LEDs de color groc
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        blue_LEDs();                // Encendre els LEDs de color blau
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        magenta_LEDs();             // Encendre els LEDs de color magenta
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        cyan_LEDs();                // Encendre els LEDs de color cyan
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        white_LEDs();               // Encendre els LEDs de color blanc
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        cule_LEDs();                // Encendre els LEDs de color FC
Blaugrana
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        merengue_LEDs();            // Encendre els LEDs de color Real
Madrid CF
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        perico_LEDs();              // Encendre els LEDs de color RCD
Espanyol
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        gironi_LEDs();              // Encendre els LEDs de color Girona
FC
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon
        reset_LEDs();               // Apagar els LEDs
        delay_ms(WAIT_HALF_SEC);    // Esperar mig segon

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

```
*****/


      /********************** Comunicacio amb LCD display
***********************/
      uint8_t msg[12];                        // Array on es desa el
missatge a transmetre
    /* Desem cadascun dels caracters del contingut del missatge en una
variable de
     * tipus array mitjançant la funció de C sprintf() i desem la mida
d'aquest */
      uint8_t msg_len = sprintf(msg, "@Hello World");
      delay_ms(LCD_TX_DELAY); // Esperem 10ms
      I2C_send(GLOB_LCD_ADDR, msg, msg_len);   // Enviem el missatge per
I2C al LCD
      delay_ms(WAIT_2S);      // Esperem 2s
      LCD_clear();            // Esborrem el contingut en pantalla
      delay_ms(LCD_TX_DELAY); // Esperem 10ms
      LCD_send();             // Enviem el missatge "INFORME SOFTWARE
MISE" al LCD

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


      /************************ Control dels actuadors
***************************/
      move_forward(VEL);  // Habilitar els actuadors per avançar
      delay_ms(WAIT_1S);  // Esperar un segon
      fleft(VEL);         // Habilitar els actuadors per girar cap a
l'esquerra endavant
      delay_ms(WAIT_1S);  // Esperar un segon
      fright(VEL);        // Habilitar els actuadors per girar cap a la
dreta endavant
      delay_ms(WAIT_1S);  // Esperar un segon
      stop();             // Deshabilitar els actuadors per aturar el
robot
      delay_ms(WAIT_1S);  // Esperar un segon
      move_backward(VEL); // Habilitar els actuadors per retrocedir
      delay_ms(WAIT_1S);  // Esperar un segon
      bleft(VEL);         // Habilitar els actuadors per girar cap a
l'esquerra enrere
      delay_ms(WAIT_1S);  // Esperar un segon
```

```
      bright(VEL);        // Habilitar els actuadors per girar cap a la
dreta enrere
      delay_ms(WAIT_1S);  // Esperar un segon
      stop();             // Deshabilitar els actuadors per aturar el
robot
      delay_ms(WAIT_1S);  // Esperar un segon
      left_rot(VEL);      // Habilitar els actuadors per rotar en sentit
antihorari
      delay_ms(WAIT_1S);  // Esperar un segon
      right_rot(VEL);     // Habilitar els actuadors per rotar en sentit
horari
      delay_ms(WAIT_1S);  // Esperar un segon
      stop();             // Deshabilitar els actuadors per aturar el
robot

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


      while(1){   // Bucle infinit per al main
          // Codi per a la comprovacio del delay a traves de
l'oscil·loscopi
          P2OUT |= BIT0;      // Posem a nivell logic alt el pin 2.0
          delay_ms(WAIT_100MS);  // Esperem un segon
          P2OUT &= ~(BIT0);   // Posem a nivell logic baix el pin 2.0
          delay_ms(WAIT_100MS);  // Esperem un segon

          // Codi del Line-Track
          delay_ms(50);
          /* En funcio de quin mode de funcionament del modul de
line-track s'ha escollit
           * al principi, s'executa la funcio amb el mode corresponent
           */
#ifdef LINE_TRACK_MODE
          linetrack(LINE_TRACK_MODE, LINETRACK_VEL);
#else
          linetrack(WALL_FOLLOWER_MODE, LINETRACK_VEL);
#endif
      }
}
/*********************************************************************
*********/
```