



UNIVERSITAT DE BARCELONA



**Projecte fi de carrera
ENGINYERIA EN ELECTRÒNICA**

Facultat de Física

**Control d'una càpsula endoscòpica mitjançant un
RTOS empotrat en un processador simple de 8 bits**

Barcelona, 17 de setembre del 2010

Autor: Joan Canals Gil

Director: Angel Diéguez Barrientos

Realitzat a: Departament d'Electrònica. UB



En aquest projecte es desenvolupen els estudis previs realitzats a l'avantprojecte fi de carrera, amb el mateix nom, realitzat al Departament d'Electrònica de la Universitat de Barcelona, el qual s'emmarca en el desenvolupament de l'electrònica per el control d'una càpsula endoscòpica amb capacitats de moviment controlat, visió i comunicacions. El control està basat en un processador 8051 de 8 bits en el qual s'executaran totes les tasques necessàries en temps real.

Es pretén desenvolupar el control d'una càpsula endoscòpica mitjançant un RTOS (de l'anglès, *Real Time Operation System*) empotrat en un processador simple de 8 bits, així com, el prototipatge de l'electrònica de control per comprovar el seu correcte funcionament.

Índex

1 INTRODUCCIÓ	7
1.1 OBJECTIU	7
1.2 METODOLOGIA I ESTRUCTURA DE LA MEMÒRIA	7
1.3 SITUACIÓ DE LA INSTAL·LACIÓ	7
1.4 ÀMBIT I ABAST DEL PROJECTE	7
2 ANTECEDENTS	9
2.1 ESTAT DE LA SITUACIÓ	9
2.2 ESTAT DE L'ART	10
2.2.1 Sistema operatiu en temps real (RTOS)	10
2.2.1.1 Disseny	10
2.2.1.2 Programació	10
2.2.1.3 Comunicació entre tasques	11
2.2.1.4 Interrupcions	12
2.2.1.5 Memòria	12
2.2.1.6 Comunicacions	12
2.2.2 Càpsules endoscòpiques	12
2.2.2.1 Primera generació	12
2.2.2.1 Nova generació	13
3 ANÀLISIS DE MERCAT	16
3.1 SECTORS ALS QUE ES DIRIGEIX	16
3.2 EVOLUCIÓ HISTÒRICA DEL MERCAT	16
3.3 PERSPECTIVES FUTURES DEL MERCAT	18
4 NORMATIVA I ASPECTES LEGALS	20
4.1 LEGISLACIÓ APLICABLE: NORMES I REGLAMENTS	20
4.2 ALTRES ASPECTES LEGALS	21
5 ENGINYERIA DE CONCEPCIÓ	22
5.1 ESTUDI DE SOLUCIONS	22
5.2 SOLUCIÓ PROPOSADA	24
6 ENGINYERIA DE DETALL	26
6.1 PROTOTIPATGE I DEPURACIÓ DE L'ELECTRÒNICA DE CONTROL	26
6.1.1 Generació i verificació de la <i>netlist</i> del DW8051	28
6.1.2 Implementació d'electrònica digital de control	38
6.1.3 Tests i depuració dels blocs del sistema	40
6.1.3.1 Test de les interfícies de connexió de les memòries	40
6.1.3.2 Test de la interfície dels registres SFRs	55
6.1.3.3 Test <i>Master I2C</i>	58
6.1.3.4 Test de la unitat de control de les comunicacions	61
6.1.3.5 Test de la unitat de control del <i>Time Stamp</i>	64
6.1.3.6 Test controlador de la locomoció (motors)	65
6.2 DISSENY SOFTWARE DE CONTROL	67
6.2.1 Requisits i característiques tècniques	67
6.2.2 Arquitectura del software de control	70
6.2.3 Tasques del software de control	71

7 VIABILITAT TÈCNICA I ECONÒMICA	80
7.1 ESPECIFICACIONS I CARACTERÍSTIQUES TÈCNIQUES	80
7.2 COSTOS D'INVERSIÓ	80
7.3 COSTOS D'EXPLOTACIÓ	80
7.4 FINANÇAMENT	80
8 CRONOGRAMA D'EXECUCIÓ	81
8.1 DEFINICIÓ DE TASQUES I TEMPS	81
8.2 FASES I OBJECTIUS	81
8.3 ASSIGNACIÓ DE RESPONSABILITATS	82
8.4 CRONOGRAMA	82
9 CONCLUSIONS	83
9.1 CONCLUSIONS	83
9.2 MILLORES PROPOSADES	83
10 BIBLIOGRAFIA I FONTS D'INFORMACIÓ	84
11 ANNEXES	86
11.1 GENERACIÓ <i>NETLIST</i> DEL DW8051 AMB <i>CORECONSULTAN</i>	86
11.2 CREACIÓ DEL SISTEMA BÀSIC BASAT EN EL DW8051 I SINTETITZACIÓ DEL MATEIX EN UNA FPGA AMB QUARTUS II	89
11.3 <i>KEIL UVISION3</i>	99
11.4 CODI FONT DELS PROGRAMES GENERATS PER ELS TESTS	102
11.5 DOCUMENTACIÓ <i>TOP TEST UNIT</i>	105
11.6 DOCUMENTACIÓ SLAVE I2C	108
11.7 DOCUMENTACIÓ <i>COMM RF</i>	109
11.8 CODI FONT DEL SOFTWARE DE CONTROL DESENVOLUPAT	110

1 Introducció

1.1 Objectiu

Aquest projecte correspon al desenvolupament de l'electrònica de control d'una càpsula endoscòpica amb capacitats de moviment controlat, visió i comunicacions. El control està basat en un processador 8051 de 8 bits en el qual s'executaran totes les tasques necessàries en temps real.

Es pretén desenvolupar el control d'una càpsula endoscòpica mitjançant un RTOS (de l'anglès, *Real Time Operation System*) embotrat en un processador simple de 8 bits, així com, el prototipatge de l'electrònica de control per comprovar el seu correcte funcionament.

1.2 Metodologia i estructura de la memòria

Les etapes a seguir seran les següents:

- Etapa de concepció: en aquesta etapa s'estudiaran les funcionalitats del sistema. Aquesta etapa es correspon a l'avantprojecte.
- Etapa de prototipatge i depuració: en aquesta etapa es tractarà la implementació del sistema de control basat en un 8051 en una FPGA. Es pot subdividir en els següents punts:
 1. Implementar l'electrònica de control en una FPGA.
 2. Realització de models funcionals sintetitzables en la FPGA per comprovar el correcte funcionament dels diferents mòduls de control.
 3. Realització del software bàsic necessari per comprovar que tots els blocs estan ben connectats i operatius.
- Etapa de disseny del software: es desenvoluparà el software necessari pel control dels diferents blocs del sistema. Aquesta etapa es pot dividir en:
 1. Disseny de les tasques individualment.
 2. Comprovació del correcte funcionament de cada tasca, dins el prototip.
 3. Unió de les tasques determinant prioritats i correspondències en els casos necessaris.
 4. Test del software complet en el prototip.

1.3 Situació de la instal·lació

El projecte es realitza al Departament d'Electrònica de la Facultat de Física de la Universitat de Barcelona, sota la direcció del Dr. Angel Diéguez Barrientos.

1.4 Àmbit i abast del projecte

L'àmbit d'aquest projecte és el de desenvolupar, mitjançant un RTOS, les tasques de control d'una càpsula necessàries, dins del projecte europeu VECTOR (*Versatile Endoscopic Capsule for gastrointestinal TumOr Recognition and therapy*).

El projecte VECTOR té per objectiu la recerca i el desenvolupament d'una píndola robòtica miniaturitzada avançada per al diagnòstic i la teràpia en el tracte digestiu humà. La missió del projecte és fer una contribució significativa al diagnòstic i tractament dels càncers digestius i dels seus precursors, i d'aquesta manera enfortir la competitivitat de la indústria biomèdica a través de tecnologies innovadores.

El projecte VECTOR està recolzat per la Unió Europea dins del 6è *EU Framework Programme for Research and Technological Development (FP6)* per una durada de quatre anys. El consorci inclou 19 socis de la indústria i organitzacions d'investigació.

2 Antecedents

2.1 Estat de la Situació

La càpsula d'endoscòpia és un terme usat per descriure una càpsula en miniatura utilitzada per gravar imatges a través del tracte digestiu pel seu ús en la medicina, figura 2.1. La càpsula és una càmera amb la mida i la forma d'una píndola.

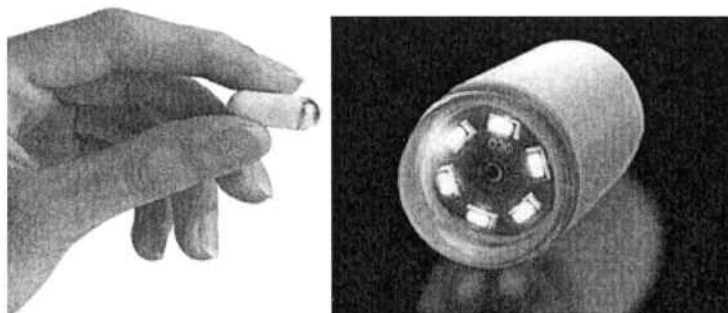


Figura 2.1: Càpsula endoscòpica

La càpsula d'endoscòpia s'utilitza com un procediment menys invasiu en l'exploració que una endoscòpia tradicional, la qual és un tub llarg i prim inserit en el recte i a través del còlon o en la cavitat oral. La tecnologia és utilitzada pels gastròlegs per detectar malalties com úlceres gàstriques, càncer de còlon, esòfag i la malaltia de Crohn. En l'actualitat la càmera de la càpsula s'utilitza principalment per visualitzar l'intestí prim.

Aquesta petita càpsula, de la mida d'una gran càpsula d'antibiòtic, és un transmissor de ràdio amb un xip de vídeo, bateria i fonts de llum. La càpsula ofereix una visió limitada de l'intestí prim, però pot proporcionar notables imatges de les petites lesions vasculars intestinals i tumors. El model actual permet la inspecció de l'intestí prim en una sola direcció, no ofereix cap possibilitat de ser dirigit i no indica la localització de les lesions vistes. La revisió de les imatges pel metge pot requerir una hora o més. El sistema inicial va ser desenvolupat, després de 20 anys d'estudi, pels doctors Iddan i Scapa, de *Given Imaging*.

Actualment existeixen diferents capsules comercials al mercat, alguns exemples són:

- **EndoCapsule** de l'empresa *Olympus* : transmet imatges en temps real, és compacte (dimensions de 26mm de llarg amb un diàmetre extern de 11mm) i de baix consum, amb una autonomia de 8 hores.
- **M2A, PillCam SB i PillCam ESO** de l'empresa *Given Imaging*: presenten unes prestacions similars a la *EndoCapsule* d'*Olympus*.
- **Sayaka** de l'empresa *RF System Lab*: encara en desenvolupament, amb la innovació de l'absència de bateria.
- **IntelliCap** de l'empresa *Philips*: encara en desenvolupament, però destaca per ser una píndola intel·ligent que conté un microcontrolador, comunicació RF, bateria i un sistema que emmagatzema medicament, que només s'aplicarà a la zona que prèviament s'hagi indicat.

Com es pot comprovar les càpsules que trobem al mercat, es limiten a la captura d'imatge en temps real, i recórrer el tracte gastrointestinal pels moviments peristàltics. Així doncs, no els és necessari un microcontrolador dotat de tasques que controlin la càpsula, ja que, amb petites màquines d'estat poden controlar el seu funcionament. A excepció del prototip de *Philips*, l'*IntelliCap*, i la *Sayaka* de *RF System Lab* que incorporen moltes més funcionalitats, però de moment estan en fase de prototip.

Per tant, com el projecte d'estudi s'emmarca dins el projecte VECTOR i com les càpsules *Sayaka* i *IntelliCap* encara no estan operatives, es pot considerar que encara no s'ha desenvolupat cap software específic per una càpsula endoscòpia intel·ligent. Així doncs aquest projecte es pot considerar un primer desenvolupament i en aquest sentit no hi ha antecedents.

2.2 Estat de l'art

2.2.1 Sistema operatiu en temps real (RTOS)

Un sistema operatiu de temps real (RTOS de l'anglès, *Real Time Operating System*), és un sistema operatiu (SO) que ha estat desenvolupat per a aplicacions de temps real. Com a tal, se li exigeix correcció en les seves respostes sota certes restriccions de temps. Si no les respecta, es dirà que el sistema ha fallat. Per garantir el comportament correcte en el temps requerit cal que el sistema sigui previsible (determinista).

Utilitzats típicament en sistemes embotrats, normalment té les següents característiques:

- No utilitza molta memòria.
- Qualsevol esdeveniment en el suport físic pot fer que s'executi una tasca.
- Multi-arquitectura (codi portat a qualsevol tipus de processador).
- Molts tenen temps de resposta predictibles per a esdeveniments electrònics.

Aquest tipus de sistema operatiu no és necessàriament eficient en el sentit de tenir una capacitat de processament alta. L'algorisme de programació especialitzat, i de vegades una taxa d'interrupció del rellotge alta poden interferir en la capacitat de processament.

Encara que per propòsit general un processador modern sol ser més ràpid, per a programació en temps real s'han d'utilitzar processadors el més predictibles possible, sense paginació. Tots els factors que afegeixen aleatorietat fan que sigui difícil demostrar que el sistema és viable, és a dir, que compleix amb els terminis temporals.

Els factors clau en un SO de temps real són espera mínima d'interrupció i espera mínima de commutació de fil. Es valora més la rapidesa o com previsiblement pot respondre, que per la quantitat de treball que pot realitzar en un període determinat de temps.

2.2.1.1 Disseny

Hi ha dos dissenys bàsics:

- Un sistema operatiu guiat per esdeveniments només canvia tasca quan un esdeveniment necessita el servei.
- Un disseny de compartició de temps canvia de tasca per interrupcions del rellotge i per esdeveniments.

El disseny de compartició de temps gasta més temps de la CPU (de l'anglès, *Central Processing Unit*) en canvis de tasca innecessaris. No obstant, dona una millor il·lusió de multitasca. Normalment s'utilitza un sistema de prioritats fixes.

2.2.1.2 Programació

En els dissenys típics, una tasca té tres estats: executant-se, preparada, bloquejada. La majoria de les tasques estan bloquejades, la major part del temps. Només una tasca s'està executant per CPU. En sistemes més simples, la llista de tasques preparades acostuma a ser curta, dos o tres tasques com a molt.

En el cas que sempre hi hagi poques tasques a la llista de preparades, aquesta llista probablement serà sempre òptima. En el cas que la llista contingui moltes tasques, aquesta llista s'ha de ordenar per prioritats d'execució, pel que trobar la tasca de major prioritats d'execució no requereix la iteració de tota la llista. Per tant, la inserció d'una nova tasca a la llista requereix examinar tota la llista fins arribar al final o trobar una tasca de menor prioritats.

El temps de resposta crític és el temps que necessari per posar en cua una nova tasca i restaurar l'estat de la tasca de més alta prioritats d'execució. En un RTOS ben dissenyat, preparar una nova tasca necessita de 3 a 20 instruccions per cada entrada a la cua, i la restauració de la tasca preparada de màxima prioritats de 5 a 30 instruccions. Com a exemple en un processador 68000 a 20MHz, els temps de canvi de tasca són de 20 microsegons amb dues tasques preparades.

Algoritmes d'ús comú en els RTOS són:

- **Programació Cooperativa:** el canvi de tasca només es pot realitzar quan la tasca crida el nucli, és a dir, la tasca es comporta de forma cooperativa i dona voluntàriament al nucli la oportunitat de canviar de tasca.
- **Programació Preventiva:** significa que canviar de tasca es pot iniciar directament des del gestor de interrupcions. Com per exemple,
 - Programació *Round-robin*, permet l'execució quasi paral·lela de varies tasques. Les tasques no són realment executades al mateix temps, però són de duració segmentada (el temps de CPU disponible se divideix en segments de temps y RTOS assigna una porció de temps a cada tasca).. Donat que l'interval de temps es curt (només uns pocs mil·lisegons) sembla que les tasques s'executin de manera simultània. Les tasques s'executen durant el seu interval de temps a menys que durin menys que aquest interval, en aquest cas es salta a la següent tasca a executar.

2.2.1.3 Comunicació entre tasques

Les diferents tasques d'un sistema no poden estar utilitzant les mateixes dades o components físics al mateix temps. Per evitar aquest problema hi ha dos dissenys destacats.

Un dels dissenys utilitza **semàfors**. En general, el semàfor pot estar tancat o obert. Quan està tancat hi ha una cua de tasques esperant a que s'obri. Els problemes que es presenten en aquest disseny són:

- **Inversió de prioritats:** una tasca de alta prioritats espera perquè una tasca de menor prioritats té un semàfor. En altres paraules, si una tasca de prioritats intermitja impedeix l'execució de la tasca de baixa prioritats, la de més alta prioritats mai s'executarà. Una solució típica seria tenir la tasca que té el semàfor, executada a la mateixa prioritats de la tasca que porta més temps esperant.
- **Punts Morts:** dues tasques tenen dos semàfors però en l'ordre invers. Això es resol normalment mitjançant un disseny acurat, realitzant cues o traient semàfors, que passen el control de un semàfor a la tasca de més alta prioritats en determinades condicions.

L'alternativa als semàfors és que les tasques s'enviïn missatges entre elles. Això té els mateixos problemes: la inversió de prioritats, que té lloc quan una tasca està funcionant en un missatge de baixa prioritats, i ignora un missatge de alta prioritats al seu correu, i els punts morts tenen lloc quan dos tasques esperen a que l'altre respongui.

Encara que el seu comportament en temps real és menys clar que en els sistemes basats en semàfors, els sistemes basats en missatges normalment es despenen i es comporten millor que els basats amb semàfors.

2.2.1.4 Interrupcions

Les interrupcions són la forma més comuna de passar informació des del món exterior al programa i són, per naturalesa, imprescindibles. En un RTOS aquestes interrupcions poden informar de diferents esdeveniments com la presència de nova informació en un port de comunicacions, de una nova mostra d'àudio en un equip de so o de un nou quadre de imatge en un gravador de vídeo digital.

Per que el programa compleixi amb seu propòsit de ser temps real, és necessari que el sistema atengui la interrupció i processi la informació obtinguda abans que es presenti la següent interrupció. Com el processador normalment només pot atendre una interrupció cada cop, és necessari que els controladors de temps real s'executin en el menor temps possible. Això s'aconsegueix no processant la senyal dins de la interrupció, sinó enviant un missatge a una tasca o solucionant un semàfor que està sent esperat per una tasca. El programador s'encarrega d'activar la tasca i aquesta s'encarrega d'adquirir la informació i completar el processament d'aquesta.

El temps que transcorre entre la generació de la interrupció i el moment en el que aquesta és atesa es diu latència d'interrupció. L'invers d'aquesta latència és una freqüència anomenada freqüència de saturació, si les senyals que s'estan processant tenen una freqüència major que la de saturació, el sistema serà físicament incapaç de processar-les. En tot cas la major freqüència que poden processar és molt menor que la freqüència de saturació i depèn de les operacions que s'hagin de realitzar sobre la informació rebuda.

2.2.1.5 Memòria

Hi ha dos problemes amb el repartiment de memòria en els RTOS:

El primer la velocitat de repartir és important. Un esquema de repartiment de memòria estàndard recorre una llista connectada de longitud indeterminada per trobar un bloc de memòria lliure; no obstant això no és acceptable ja que el repartiment de la memòria ha de succeir en un temps determinat en el RTOS.

En segon lloc, la memòria pot fragmentar-se quan les regions lliures es poden separar per regions que s'estan utilitzant. Això pot provocar que es pari un programa, sense possibilitat d'obtenir memòria, encara que en teoria existeixi suficient memòria. Una solució és tenir una llista vinculada LIFO de blocs de memòria de mida fixa. Això funciona extraordinàriament bé en un sistema simple.

La paginació s'acostuma a desactivar-se en els sistemes en temps real, ja que es un factor bastant aleatori i imprevisible, que varia el temps de resposta i no ens permet assegurar que es compliran els temps, degut al tràfec de pàgines de memòria amb un dispositiu d'emmagatzement (en anglès, *trashing*).

2.2.1.6 Comunicacions

Per les comunicacions es sol utilitzar connexions o xarxes deterministes com el bus CAN (de l'anglès, *Controller Area Network*) o port sèrie, ja que les xarxes més usals, com *Ethernet*, són indeterministes i no poden garantir el temps de resposta.

2.2.2 Càpsules endoscòpiques

2.2.2.1 Primera generació

La primera càpsula sense fils va ser dissenyada i comercialitzada per *Given imaging*, la **M2A**, que va esdevenir el punt de referència de les capsules endoscòpiques. Bàsicament és una càpsula passiva amb càmera, que recorre el tracte gastrointestinal gràcies als moviments peristàltics i va prenent imatges.

La càpsula M2A, és de 26 mm de longitud i el diàmetre és de 11 mm. Està dissenyat per a enviar les imatges adquirides fora del cos humà a través d'un transceptor de RF. Va prenent imatges del tracte gastrointestinal i les envia a una velocitat de 2 imatges per segon.

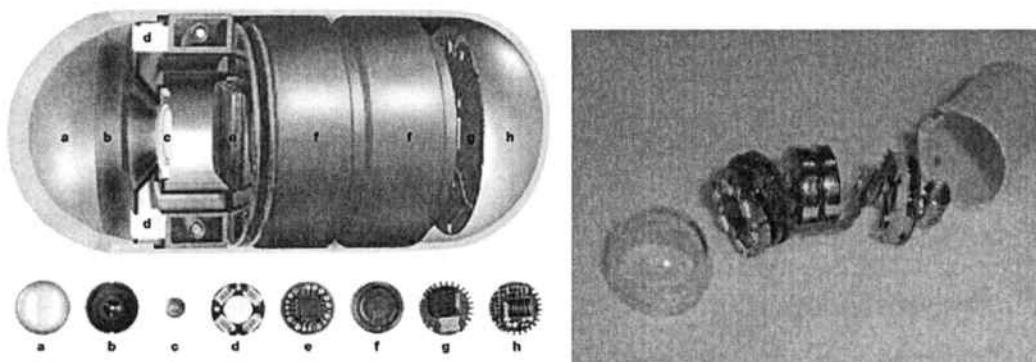


Figura 2.2: Parts de la càpsula endoscòpica M2A.

A la figura 2.2 es poden apreciar clarament les diferents parts i com està construïda la capsula endoscòpica M2A. La M2A està constituïda per: a) Cúpula, b) Protector de la càmera, c) Objectiu, d) LED, e) Sensor CMOS, f) Bateries, g) Transmissor, h) Antena.

La segona càpsula endoscòpica en comercialitzar-se va ser la *EndoCapsule* de l'empresa *Olympus*, figura 2.3, amb unes prestacions similars a les de la M2A.



Figura 2.3: EndoCapsule d'Olympus

Té una càmera amb més resolució però manté la mateixa velocitat de 2 imatges per segon. La il·luminació s'aconsegueix mitjançant l'ús de 6 LEDs blancs.

Tant la càpsula de *Given imaging* com la càpsula de *Olympus* són la primera generació de càpsules endoscòpiques amb capacitats simples, que només prenen fotos i les envien a l'estació exterior perquè els metges puguin realitzar el diagnòstic.

La segona generació de càpsules endoscòpiques s'ha dirigit a millorar el sistema de visió. En aquesta generació hi ha càpsules especialitzades segons la zona del tracte gastrointestinal que és vol examinar. Un exemple el trobem en la *PillCam ESO* i la *PillCam SB* de l'empresa *Given Imaging* que serveix per examinar l'esòfag i l'intestí prim respectivament.

2.2.2.1 Nova generació

La nova generació de càpsules endoscòpiques estarà equipada amb més capacitats. Això implica l'addició de més sensors i actuadors, i al final, la inclusió de més productes electrònics que ajudin a una millor diagnòstic i tractament de les malalties. Un exemple de la nova generació de càpsules és la càpsula endoscòpia *Sayaka* de l'empresa *RF System Lab* i la càpsula que està desenvolupant *Philips* l'*IntelliCap*.

La capsula *Sayaka*, figura 2.4, es caracteritza per:

- LEDs fluorescents que s'utilitzen també com a sensors per detectar tumors.
- No té bateria, s'alimenta via *wireless* des de l'exterior del cos.

- La càmera està en la superfície lateral en lloc de a un extrem de la càpsula.
- La doble estructura de càpsula formada per una càpsula externa i una interna. Mentre la càpsula externa viatja a través del tracte gastrointestinal, la càpsula interna gira sola dins l'externa. Aquest gir es deriva de petits imants permanents i un electroimant que intensifica la rotació.

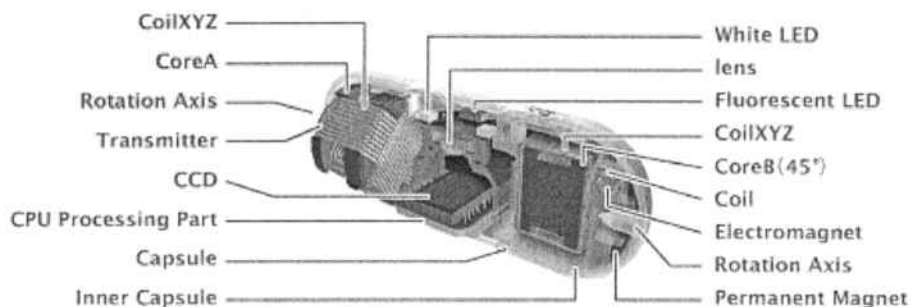


Figura 2.4: Secció de la càpsula Sayaka.

Per tal de controlar el sistema en conjunt la càpsula necessita un processador. La CPU està a càrrec del processament de la imatge, i la conducció dels LED. A més, la càpsula també està equipada amb un sistema de radiofreqüència per permetre la comunicació amb l'exterior.

Aquest nou disseny obté imatges laterals ben definides de la paret gastrointestinal mentre que la càpsula interior gira en el tracte digestiu. Millorant d'aquesta manera les imatges obtingues amb les capsules que tenen la càmera en un extrem, que només poden capturar imatges dins el seu camp de visió on veure el tracte digestiu bé es gairebé impossible.

Les imatges capturades es transmeten al receptor i després es processen en una àmplia sèrie d'imatges solapades formant un mosaic d'imatges tal i com es mostra en la figura 2.5. Aquest "mosaic" d'imatges es crea per reproduir de 6 a 8 metres de llarg del canal digestiu. El resultat és el tracte en forma de tub que pot ser disseccionat per veure en format llarg. Els tractes desenvolupats poden ser emmagatzemats en forma llarga i contínua on qualsevol punt es poden ampliar fins a 75 vegades a la pantalla.

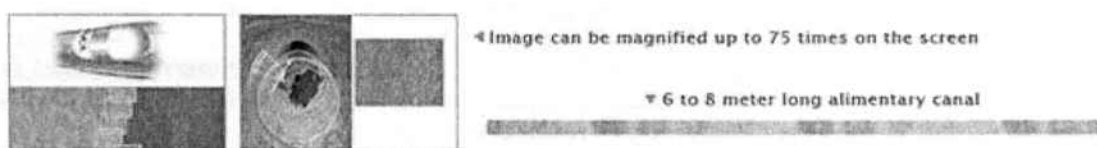


Figura 2.5: Sistema de processat d'imatge.

L'**IntelliCap** de *Philips*, figura 2.6, no és una càpsula endoscòpica completa ja que no disposa d'un sistema de visió, és només un dispositiu d'administració de fàrmacs controlat electrònicament. Tot i això, la càpsula té les dimensions de d'una càpsula 11 x 26 mm. A més l'*IntelliCap* està equipada amb:

- Microcontrolador
- Bateria
- Sensor de pH
- Sensor de Temperatura
- Transceptor RF sense fils
- Bomba de líquids
- Dipòsit de fàrmacs

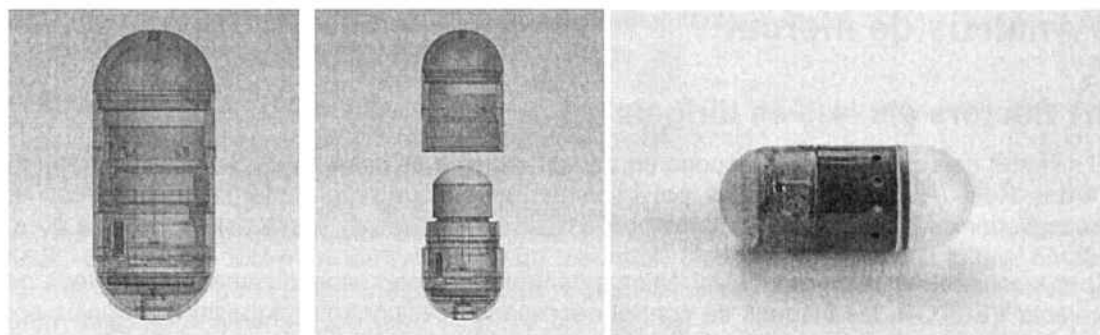


Figura 2.6: A l'esquerra l'esquema i a la dreta el prototip de l'*IntelliCap* de Philips

La càpsula es comunica a través del seu transceptor a una unitat de control fora del cos.

La distribució localitzada de fàrmacs es porta a terme mitjançant una bomba interna de l'*IntelliCap* sota el control del processador, el que permet un control precís del perfil d'administració de fàrmacs. Alguns dels possibles perfils de lliurament inclouen l'alliberament progressiu, dosificació multilocalització o de cop.

La preplanificació pot ser utilitzada per determinar la ubicació on s'ha de realitzar el lliurament dels fàrmacs, i per tant definir un programa de control pel processador. Aquest programa es carrega a l'*IntelliCap* abans de ser ingerida, i controla l'execució del perfil d'administració de medicaments en resposta a un pH que mesura a través de l'intestí. Altres dades de la *IntelliCap*, com les mesures de temperatura, s'envien a la unitat de control externa, que registra les dades i també pot transmetre senyals addicionals de control de nou a l'*IntelliCap*.

A la taula 2.1 és mostren les principals característiques de les capsules esmentades juntament amb dues capsules més de nova generació la *Sayaka* de l'empresa *RF System Lab* i l'*IntelliCap* de Philips.

Taula 2.1: Taula comparativa de les capsules endoscòpiques.

	M2A	Endocapsule	Sayaka	IntelliCap
Dimensions (DxL)	11 mm x 26 mm	11 mm x 26 mm	9 mm x 22 mm	11 mm x 26 mm
Regió d'Operació	Intestí prim	Intestí prim	Intestí prim i gros	Par baixa del tracte GI
Càmera	CMOS	CCD	CCD	No
Resolució	256 x 256 pixels	320 x 320 pixels		
Frame rate	2 imatges/segon	2 imatges/segon	30 imatges/segon	
Il·luminació	4 to 6 LEDs blancs	6 LEDs blancs	4 LEDs	No
Locomoció	Peristaltisme	Peristaltisme	Peristaltisme	Peristaltisme
Electrònica	RF transceptor Controlador de LEDs amb elements discrets	RF transceptor Controlador de LEDs amb elements discrets	CPU processador: - Processat d'imatge - Control de LEDs RF transceptor	uC RF transceptor
Sensors	No	No	LEDs Fluorescent	pH (utilitzat per la localització) Temperatura
Actuadors	No	No	Sistema de rotació	Bomba de fluids Dipòsit de fàrmacs
Altres	1era càpsula comercial.	Diagnosi en temps real	No necessita bateria	Eina electrònica dispensadora de fàrmacs

3 Anàlisi de mercat

3.1 Sectors als que es dirigeix

El sistema de control que es proposa en aquest projecte es basa en un Sistema Operatiu en Temps Real (RTOS), el qual es pot implementar en qualsevol sistema que presenti un processador de 8 bits compatible amb el 8051.

Com a conseqüència de que l'electrònica a controlar ha estat específicament dissenyada pel projecte VECTOR, les tasques de control necessàries per controlar aquesta electrònica són específiques del projecte i no es poden exportar a altres càpsules o dispositius directament.

Per tant no es realitzarà un estudi de mercat sobre el software de control específic, però sí que es realitzarà un estudi detallat sobre les possibles orientacions mercantils de les càpsules endoscòpiques, que és on s'emmarca aquest projecte.

3.2 Evolució històrica del mercat

El mercat de la càmera endoscòpia sense fils està creixent a un ritme increïble, tal i com es pot comprovar a la figura 3.1 on es pot observar el creixement del mercat americà.

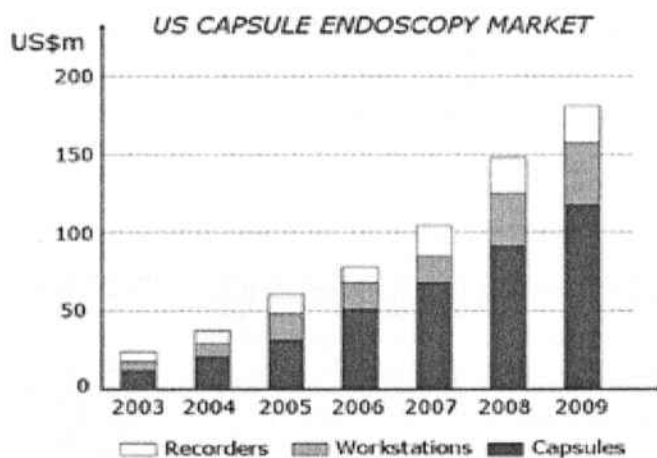


Figura 3.1: Mercat de les càpsules d'endoscòpia.

L'empresa japonesa *Olympus*, líder mundial en el mercat de les endoscòpies, és el gran competidor en l'actualitat de *Given Imaging*. Des de la seva fundació el 1919 *Olympus*, un líder en tecnologia de precisió, ha creat innovadores solucions *opto-digital*s per la salut, ciències de la vida i productes d'electrònica de consum. Ha portat a la indústria el desenvolupament d'innovacions medicinals a través de les seves línies de negoci. Al llarg de la seva història *Olympus* ha estat coneguda per molts com a pionera de la gastrocàmera, la primera computadora d'ADN per a l'anàlisi de gens i el primer endoscopi amb sistema d'alta definició i imatges de banda estreta. Avui en dia *Olympus* té una quota de mercat superior al 70% en el negoci mundial de l'endoscòpia.

Fent un cop d'ull a la cronologia de la innovació de *Olympus*, s'observa que al 1927 va ser pioner del microscopi d'alta qualitat biològica al Japó (microscopi que requereixen una gota d'oli entre mostra i l'objectiu de la lent per a les observacions). Per primera vegada els científics podien visualitzar el món microscòpic. Es troben en els principals laboratoris de recerca, clíniques i centres de institucions educatives a tot el món, als EUA més que cap altra marca.

No va ser fins al 1950 que va desenvolupar la primera gastrocàmera, muntada a l'extrem d'un tub flexible, on una càmera en miniatura ho registrava tot. Va ser una revolució per a la

endoscòpia, per primera vegada els metges podien diagnosticar sense cirurgia, la qual cosa és més ràpida i té menys risc d'infecció.

Al 1964, gràcies a la fibra òptica, van fer avenços en la gastrocàmera, que permetia l'observació de l'estómac en temps real. Amb la miniaturització va fer el tractament menys invasiu, més curt, menys costós i més eficaç.

No va ser fins al 1978 que *Olympus* va posar a prova *Automated Electrophoresis System (AES)*, un sistema totalment automatitzat, un laboratori d'anàlisi de sang per a una àmplia gamma de fases de la malaltia, com proves d'anticossos per separar les proteïnes en el sèrum sanguini anomenat globulina.

Al 2002 va desenvolupar el primer ordinador d'ADN per a l'anàlisi de gens, però no va ser fins al 2005 quan van presentar un sistema revolucionari per simplificar i racionalitzar el procediment de l'endoscòpia, i amb alta definició, amb uns escaneig electrònic de fins a 360 graus, amb capacitat avançada d'ultrasò.

Finalment, al 2007 va comercialitzar la primera càpsula amb visor en temps real, anomenada *EnteroPro*, que ajuda als metges a diagnosticar i tractar anomalies de l'intestí prim. La reproducció en color i control de brillantor automàtic són les principals millores. És una de les càpsules més venudes per *Olympus* i és de 26mm de llarg amb un diàmetre extern de 11mm. És compacta, de baix consum d'energia, tecnologia d'imatge i tecnologia de transmissió sense fils.

El mercat avui en dia està dividit entre poques empreses i models, però la principal competidora a batre és *Olympus*, tal i com es mostra en la següent figura 3.2, ja que té un major nombre de patents a WCE:

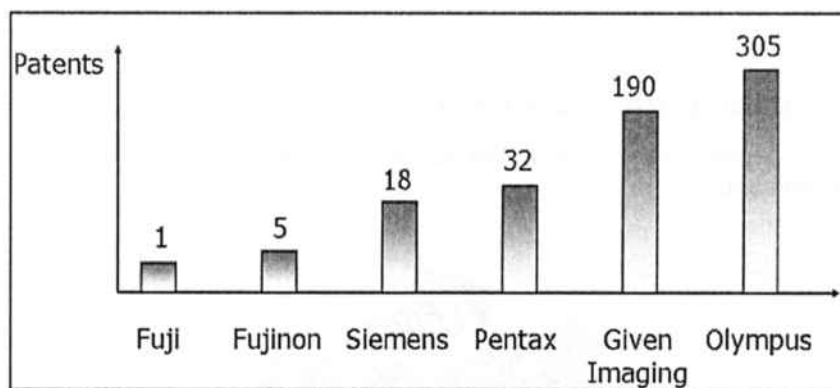


Figura 3.2: Companyies amb majors patents a la WCE.

Hi ha diversos productes semblants en el mercat, com són:

- **M2A**, de l'empresa *Given Imaging* de Israel.
- **EndoCapsule** de *Olympus* de Japó.
- **Sayaka** de *RF System Lab*, també de Japó.
- **MiroCam** de *IntroMedic* de Corea.
- **OMOM** de *Jinshan Science & Technology* de Xina.



Figura 3.3: Càpsules endoscòpiques d'esquerra a dreta *EndoCapsule*, *MiroCam*, *OMOM*.

En el panorama general del mercat de l'endoscòpia GI ha estat dominat per *Olympus* al 2007 degut al gran percentatge de dispositius que són reutilitzables. *Olympus* té una gran quota del mercat de les càmeres, però la companyia també domina el mercat de l'endoscòpia.

Després de perdre la quota de mercat en 2006, Boston Científic ha consolidat la seva posició de lideratge en segments que tradicionalment ha mantingut el control. Tant *Boston Scientific* com *Olympus* segueixen desenvolupant nous i innovadors productes als quals vetllen pel seu lideratge durant el període de previsió. Juntament amb *Wilson-Cook* representen més del 70% de tot el mercat, només aquestes 3 empreses.

El mercat de dispositius d'endoscòpia inclou endoscòpies i dispositius de la urologia, ginecologia, broncoscòpia, ORL endoscòpia GI, laparoscòpia, etc. En 2008, la combinació del mercat canadenc per a dispositius d'endoscòpia es va valorar en més de 310 milions de dòlars. Aquest mercat s'espera que creixi de manera constant durant el període de previsió i assoleixin un valor de més de 484 milions de \$ en 2015. Aquest augment serà impulsat pel ràpid creixement de l'endoscòpia GI i els vídeos d'alta tecnologia.

3.3 Perspectives futures del mercat

Més de 260.000 pacients han provat aquestes *PillCams*, figura 3.4, des de 2001 i actualment es continuen utilitzant.



Figura 3.4: Càpsula endoscòpica *PillCam*.

Tal i com indica la figura 3.5, el mercat dels dispositius d'endoscòpia als EUA s'espera que generi fins a 480 milions de dòlars al 2014, amb un increment progressiu cada any amb *Olympus* & *Given Imaging* al capdavant. El panorama futur per al mercat europeu de l'endoscòpia GI és saludable.

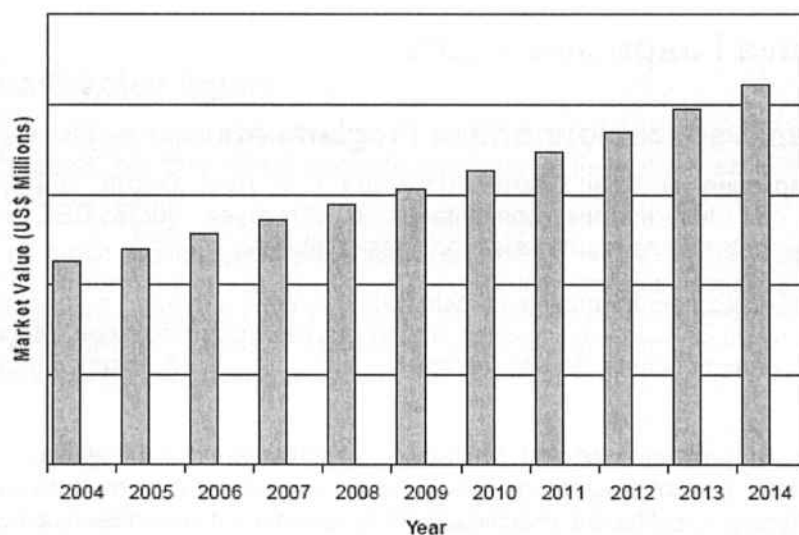


Figura 3.5: Previsió de l'evolució del mercat dels dispositius endoscòpics

4 Normativa i aspectes legals

4.1 Legislació aplicable: normes i reglaments

A Espanya apliquen el Real Decret 1616/2009 i el Real Decret 1591/2009 com a transposicions de les normes comunitàries o Directives 90/385/CEE i 93/43/CEE respectivament, pels que es regulen els productes sanitaris.

S'entén per producte sanitari qualsevol instrument, dispositiu, equip, material o altre article utilitzat només o en combinació, inclosos els programes informàtics que intervinguin en el seu bon funcionament, destinats pel fabricant a ser utilitzats en éssers humans amb les finalitats de:

- diagnòstic, prevenció, control, tractament o alleujament d'una malaltia.
- diagnòstic, control, tractament, alleujament o compensació d'una lesió o deficiència.
- investigació, substitució o modificació de l'anatomia o d'un procés fisiològic.
- regulació de la concepció.

S'ha de tenir en compte la majoria d'organitzacions del sector medicinal, com per exemple la AAMI (*Association for the Advancement of medical Instrumentation*), on han elaborat la norma ESI – 1993: *Safe current limits for electromedic apparatus*, la qual estableix els límits i els mètodes de test per perills a causa del corrent.

El lloc d'ús de l'aparell generalment serà en un hospital, on cal destacar la norma NFPA (*National Fire Protection Association*) 99-1993, on marca la seguretat elèctrica en hospitals. També inclou una secció pels sistemes de potència elèctrica, i aconsella els requisits pels fabricants d'equips mèdics, períodes d'inspecció, límit de corrents de fuga i mètodes de test.

La IEC (*International Electrotechnical Commission*) 601-1, explica els requisits generals de seguretat elèctrica per equips electromèdics. AENOR (Associació Espanyola de Normalització i Certificació) ha publicat la norma espanyola UNE 60601 pels equips electromèdics. S'ha de complir aquesta norma perquè l'equipament mèdic de fabricació nacional ha d'inscriure's en la Direcció de Tecnologia Mèdica del ANMAT, per tal de poder registrar-se i comercialitzar-se.

El mercat de la Comunitat Europea (CE) indica la conformitat del producte amb la directiva que li és aplicada, sinó no es pot vendre a Europa, és a dir, és un requisit obligatori per la comercialització de productes sanitaris a Europa, i s'indica en la seva etiqueta o instruccions d'ús amb el següent símbol mostrat a la figura 4.1:



Figura 4.1: Símbol del mercat CE.

El mercat CE d'un aparell implica complir una sèrie de requisits essencials i elaborar una documentació tècnica i establir un control del procés de fabricació.

La Normativa Europea de Dispositius Mèdics (de l'anglès, *European Medical Device Regulation*) ha escrit una sèrie de documents com un servei a la comunitat de dispositius mèdics. Tots els documents van ser publicats per el Parlament Europeu i la Comissió Europea. Els documents i els seus articles que ens impliquen en el projecte són:

- Active Implantable Medical Devices Directive (90/385/EEC).
- Cosmetic Products Directive (76/768/EEC).
- Medical Device Directive (93/42/EEC).

4.2 Altres aspectes legals

El projecte està vinculat al projecte europeu VECTOR, en el qual ja s'hi ha definit les normes i reglaments aplicats, per tant, s'està subjecte a aquesta legislació, en especial èmfasi en el contracte de confidencialitat.

Com que és en àmbit d'investigació universitària es podria realitzar algun tipus de publicació dels resultats obtinguts amb la finalitat de que els drets intel·lectuals derivats a la investigació realitzada siguin de caràcter públic i cap empresa pugui patentar com a propis. Tenint en compte que el resultat final estarà patentat pel projecte europeu VECTOR, cada part per separat pot estar patentada pel dissenyador.

5 Enginyeria de concepció

5.1 Estudi de solucions

Conegut l'objectiu del projecte, per desenvolupar el control d'una càpsula endoscòpica mitjançant un RTOS embotat en un processador 8051 de 8 bits, cal tenir present que l'electrònica de control ve determinada pel projecte VECTOR.

Aquesta electrònica de control és un prototip i per tant no hi ha cap precedent pel seu control, ja que encara no s'ha fabricat. Per aquest motiu, la primera tasca a realitzar és el seu prototipatge en una FPGA per comprovar el seu correcte funcionament, i obtenir d'aquesta manera un sistema de depuració del codi de control en temps real, ja que realitzar simulacions en l'entorn de disseny requeriria un temps excessiu.

L'arquitectura de l'electrònica de control està subdividida en blocs on es troben:

- **DW8051 de Synopsis**, el cervell del sistema que és un processador de 8 bits on s'executaran les tasques de control de la resta de blocs. Dotat de 256 Bytes de memòria RAM interna, 2 KB de memòria RAM externa i una memòria de programa de 8 KB i un bus SFR (de l'anglès, *Special Function Register*) per perifèrics externs.
- **Unitat de control de la locomoció/motors**, s'encarrega del control dels motors que fan moure les potes de la càpsula.
- **Unitat de control dels LEDs**, 16 LEDs encarregats d'il·luminar la regió on enfoca la càmera, i que s'han de controlar en grups de 4.
- **Master I2C**, tan per controlar i comunicar-se amb els dispositius que funcionen per I2C, com és la càmera CMOS o pels possibles dispositius que és puguin connectar en un futur a la càpsula per dotar-la de noves funcionalitats.
- **Unitat de control de les comunicacions**, per rebre dades de l'exterior.
- **Unitat de control del Time Stamp**, encarregat de enviar una senyal de sincronisme per les dades que s'envia cap a l'exterior.
- **Boot Loader**, és per on es pot carregar el codi a la memòria de programa.
- **Test SFRs** permet l'accés directe a la bus SFR des del exterior.
- **Generador de rellotge** encarregat de generar el rellotge necessari per el bloc de comunicacions i les bombes de carregar del controlador dels motors.

Tots els blocs controladors descrits anteriorment tenen els seus propis registres de configuració connectats al bus SFR del processador DW8051, que es correspon amb el bus intern del sistema a controlar.

Tot el sistema està regit per una senyal de rellotge de 10 MHz, que dona els límits de tasques a realitzar.

Per realitzar el control de la càpsula hi ha tres opcions:

1. **Programació amb una única tasca (Single-Tasking)**, és l'estàndard dels programes en C que comencen l'execució amb la funció principal de C *main*. En una aplicació embotada, la funció *main* normalment s'implementa com un bucle infinit que es pot considerar com una única tasca que s'executa de manera continua. Per exemple:

```
void main (void)
{
```

```

while (1)          /* repeat forever */
{
    do_something (); /* execute the do_something (task) */
}

```

En aquest exemple, la funció *do_something* pot considerar-se com una sola tasca. Donat que només hi ha una tasca que s'executa no hi ha necessitat de capacitat multitasca o d'un sistema operatiu multitasca.

2. **Programació multitasca**, programes més sofisticats en C poden posar en pràctica un règim de pseudo-multitasca, on varies funcions (o tasques) es van cridant consecutivament dins d'un bucle. Per exemple:

```

void main (void)
{
    int counter = 0;

    while (1)          /* repeat forever */
    {
        check_serial_io ();      /* check for serial input */
        process_serial_cmds (); /* process serial input */

        check_kbd_io ();        /* check for keyboard input */
        process_kbd_cmds ();    /* process keyboard input */

        adjust_ctrlr_parms (); /* adjust the controller */

        counter++;             /* increment counter */
    }
}

```

En aquest exemple, cada funció (o tasca) realitza una operació separada. Les funcions (o tasques) s'executen en ordre una darrere l'altra.

La programació i l'execució de les tasques comença a ser un problema. Per exemple, si la funció *process_kbd_cmds* s'executa durant massa temps, el bucle principal pot tardar molta estona a tornar a la funció *check_serial_io* i les dades que arriben pel port sèrie es poden perdre. Per suposat, la funció *check_serial_io* es pot cridar amb més freqüència dins el bucle principal per corregir el problema, però a la llarga aquesta tècnica no funciona. Una altra possible solució seria augmentar la freqüència del processador, però això implicaria més consum.

3. **Programació utilitzant un RTOS**, el que permet l'utilització d'un RTOS, és crear una tasca (funció) independent per cadascuna de les tasques que s'han d'executar constantment o ocasionalment. Per exemple:

```

void check_serial_io_task (void) _task_ 1
{
    /* This task checks for serial I/O */
}

void process_serial_cmds_task (void) _task_ 2
{
    /* This task processes serial commands */
}

void check_kbd_io_task (void) _task_ 3
{
    /* This task checks for keyboard I/O */
}

void process_kbd_cmds_task (void) _task_ 4
{
    /* This task processes keyboard commands */
}

```

```

void startup_task (void) _task_ 0
{
    os_create_task (1);    /* Create serial_io Task */
    os_create_task (2);    /* Create serial_cmds Task */
    os_create_task (3);    /* Create kbd_io Task */
    os_create_task (4);    /* Create kbd_cmds Task */

    os_delete_task (0);    /* Delete the Startup Task */
}

```

En aquest exemple, cada funció defineix una petita tasca del RTOS. Aquestes tasques no tenen funcions principals, al contrari, comença executant la tasca 0 que crea la resta de tasques, que comencen a executar-se. De manera que el mateix RTOS reparteix el temps per igual a les tasques que s'executen i evita així el cas de que una tasca s'executi durant massa temps.

Per tant, un sistema operatiu en temps real permet la programació flexible dels recursos del sistema, com la CPU la memòria i ofereix comunicació entre tasques, posada en espera de tasques, activació de tasques en espera, etc.

5.2 Solució proposada

Degut a que la càpsula serà controlada totalment des de l'exterior gràcies al bloc de comunicacions i que aquesta ha de respondre a les ordres de moure's, parar-se, encendre LEDs, adquirir imatge, etc en temps real. La solució proposada per a realitzar el software de control de tots els sistemes és mitjançant un RTOS.

En el mercat es troben multitud de RTOS. A l'hora d'escollir s'ha de tenir present els recursos disponibles descrits anteriorment, que són principalment la poca potència del processador (DW8051) i sobretot la poca memòria de programa (codi) disponible 8 KB.

Els RTOS més destacats són els següents:

- **RTX51 Tiny** de la casa *Keil*, és gratuït i funciona amb tots els derivats del processador 8051. Les seves característiques es mostren en la taula 5.1.

Taula 5.1: Característiques principals del *RTX51 Tiny*.

Paràmetre	Límit
Nombre màxim de tasques	16
Nombre màxim de tasques actives	16
Espai requerit per CODI	900 Bytes màxim
Espai requerit per DADES	7 Bytes
Temporitzadors usats	1
Latència d'interrupció	20 Cicles o menys
Cicles requerits per canvi entre tasques	100-700 Cicles

Requereix les següents aplicacions per ser utilitzat:

- C51 Compiler
- A51 Macro Assembler
- BL51 Linker o LX51 Linker
- Juntament amb les seves llibreries

- **RTX51** de la casa *Keil*, és la versió no gratuïta del *RTX51 Tiny*, presenta millors prestacions que l'anterior entre elles, més nombre de tasques 256, més tasques actives 19, semàfors, és preventiu, etc.

Per contra, requereix més espai per codi de 6 a 8 KBytes, el que és gairebé tota la memòria de que disposem (8 KB).

- **uC/OS-II** de l'empresa *Micrium*, com el *RTX51*, és més potent i té més prestacions que el *RTX51 Tiny*, però també requereix molta memòria de codi entre 5K - 24KBytes depenent de la configuració.

Com es pot comprovar, el que s'adequa més al projecte és el **RTX51 Tiny**, degut al poc espai que ocupa i les altes prestacions que presenta, que són suficients per assolir els objectius marcats. Per altra banda la mateixa casa *Keil* té el software necessari per compilar i crear el codi necessari pel processadors de la família del 8051 entre d'altres, aquest software és el *uVision3*.

6 Enginyeria de detall

L'objectiu principal del projecte és desenvolupar les tasques bàsiques de control del xip VECTOR V1, mitjançant la utilització del RTOS *RTX Tiny*, sobre un prototip implementat en una FPGA. D'aquesta manera s'aconsegueix un disseny simultani de software i hardware.

Per tant, hi ha una primera etapa que consisteix en la integració de l'electrònica digital de control en la FPGA, i una segona etapa de desenvolupament de software.

6.1 Prototipatge i depuració de l'electrònica de control

Per poder desenvolupar el software de control, és necessari tenir una plataforma on poder depurar el codi en temps real. Per tant, el pas previ a la realització del software de control és realitzar el prototipatge i depuració de l'electrònica de control, per comprovar el seu correcte funcionament.

Per realitzar aquestes tasques s'utilitza una FPGA que permet veure com funciona tota l'electrònica digital en temps real facilitant d'aquesta manera la seva depuració, i per altra banda proporciona la plataforma de test i depuració del software de control a desenvolupar.

La FPGA utilitzada és una *Cyclon II* d'Altera muntada en una placa DE2 (de l'anglès, *Development and Education*) de la mateixa casa Altera. A més la mateixa casa Altera proporciona el programa *Quartus II*, que permet dissenyar i sintetitzar el sistema de control a la placa DE2, podent aprofitar d'aquesta manera tots els recursos que ofereix aquesta placa de desenvolupament.

Si s'examina l'electrònica de control del xip VECTOR V1, figura 6.1, es pot comprovar que només es necessita implementar la part digital, ja que aquesta controla la part analògica mitjançant registres de configuració i màquines d'estats (FSM de l'anglès, *Finite State Machine*) de les diferents unitats de control (CU de l'anglès, *Control Unit*), i és aquesta part la que el software a desenvolupar controlarà.

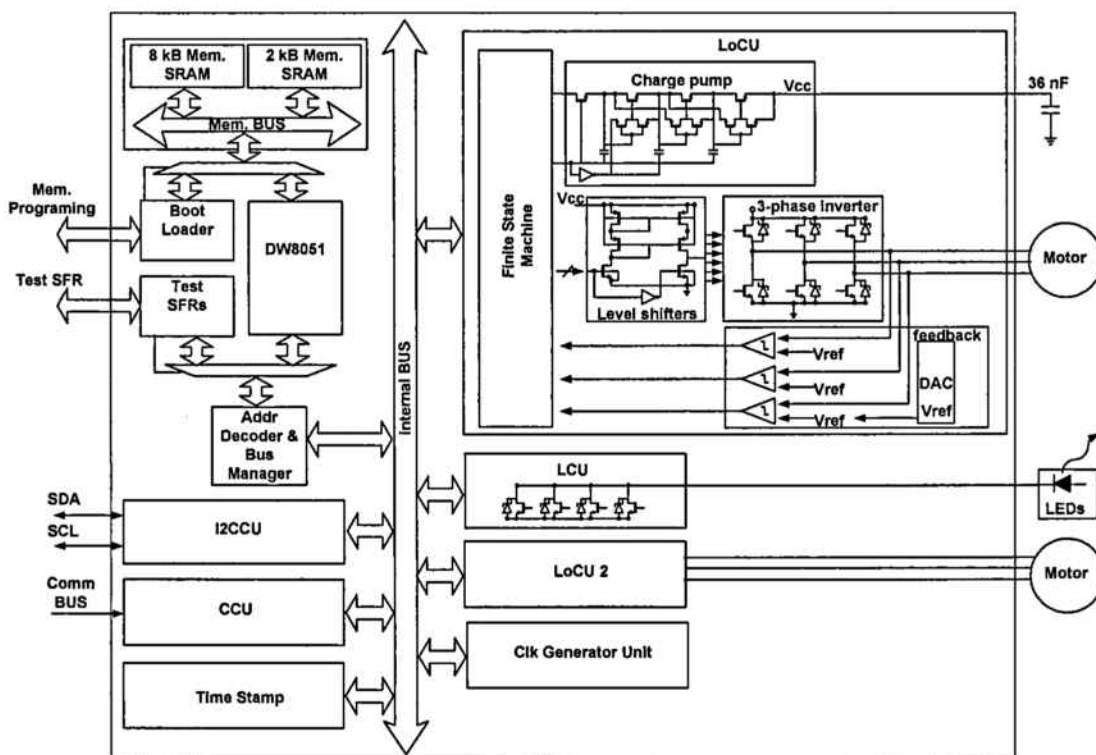


Figura 6.1: Esquema de l'electrònica de control.

La implementació de l'electrònica digital de control a la FPGA és possible, ja que gairebé tots els blocs han estat dissenyats utilitzant un llenguatge de descripció de hardware d'alt nivell, en aquest cas verilog, que permet que sigui transportable a qualsevol tecnologia i per tant pot ser sintetitzat dins la FPGA. Els únics blocs que no han estat dissenyats expressament pel projecte VECTOR són:

- El processador de 8 bits DW8051 que es tracta d'una propietat intel·lectual (IP de l'anglès, *Intellectual Property*) de la casa Synopsys.
- Les memòries SRAMs (interna de 256 bytes, externa de 2 KB i memòria de programa de 8 KB), que són pròpies de la tecnologia d'Austria Micro Systems (AMS) amb la que es fabricarà el ASIC (de l'anglès, *Application-Specific Integrated Circuit*) de les quals només es disposa de models funcionals per realitzar tests en simuladors.

Degut a que les memòries utilitzades en el projecte VECTOR no estan disponibles per sintetitzar a la FPGA, és necessari realitzar una sèrie de canvis en el disseny original. Per aquest motiu s'han de substituir les memòries originals, així com les seves interfícies de connexió amb el processador, pels models propis que ens proporciona el software de configuració de la FPGA.

Per altra banda, és necessari comprovar que les interfícies de connexió entre les memòries i el DW8051 funcionen correctament, i per aquest motiu és necessari descriure, mitjançant verilog, una memòria de 256 bytes sintetitzable a la FPGA que es comporti com les memòries SRAMs que s'utilitzaran en l'ASIC final.

Per tant els passos a seguir durant la implementació i la posterior depuració són:

1. Generar la *netlist* del DW8051 sintetitzable per la FPGA i comprovar el seu correcte funcionament.
2. Implementar l'electrònica digital de control amb les memòries pròpies de la FPGA i comprovar que és capaç d'executar un programa senzill on es vegi com s'accedeix a les memòries i s'utilitza el bus intern del sistema.
3. Realitzar test específic per cada bloc del sistema:
 - a. **Memòries** del sistema i les seves interfícies.
 - b. **Boot Loader**: interfície que permet fer l'escriptura i lectura de la memòria de programa des de l'exterior.
 - c. **Test SFRs** interfície que permet fer l'escriptura i lectura dels registres SFRs des de l'exterior.
 - d. **Bus intern del sistema (bus SFR)** i la seva interfície de test: encarregat d'escriure als registres de control dels diferents blocs controlador dels motors, controlador dels LEDs, etc.
 - e. **Master I2C**: encarregat de comunicar-se amb tots els mòduls externs del xip que es comuniquin per I2C com és la càmera.
 - f. **Generador de rellotges**: encarregat de generar les senyals de rellotges necessàries pel bloc de comunicacions, les bombes de carregar utilitzades en la locomoció (per controlar els motors).
 - g. **Unitat de control de les comunicacions**: encarregat de rebre ordres de l'exterior.
 - h. **Unitat de control del Time Stamp**: encarregat de generar una senyal de rellotge que s'utilitza com a marca temporal.
 - i. **Unitat de control de la locomoció (motors)**: encarregat de controlar i generar les senyals per fer que els motors girin cap al costat desitjat.

Aquests tests, en alguns casos, impliquen desenvolupar blocs de hardware específics en verilog, ja que els blocs com el *Boot loader*, el *master I2C* i el controlador de comunicacions interactuen amb elements externs.

6.1.1 Generació i verificació de la *netlist* del DW8051

Abans de generar la *netlist* del DW8051 és convenient tenir una idea general de la seva arquitectura i com s'organitza la memòria.

- **Visió general de l'arquitectura**

La figura 6.2 mostra l'arquitectura del nucli del DW8051. El nom del mòdul superior és *DW8051_core*. El següents submòduls i interfícies són opcionals i es poden seleccionar mitjançant paràmetres durant el procés de generació de la *netlist*:

- Els **128 bytes superiors de la RAM interna** són opcionals. Són assequibles quan *ram_256* és 1.
- El **mida de la ROM interna** és determina pel paràmetre *irom_addr_size*. Pot ser zero.
- El **temporitzador 2 (Timer 2)** és present quan *timer2* és 1.
- Els **ports series P0 i P1**. El Port 0 és present quan *serial* és 1 o 2 i el Port 1 és present només quan *serial* és 2.
- La **unitat d'interruptió** pot ser de 6 fonts d'interruptió quan *extd_intr* és 0 o de 13 fonts quan és 1.

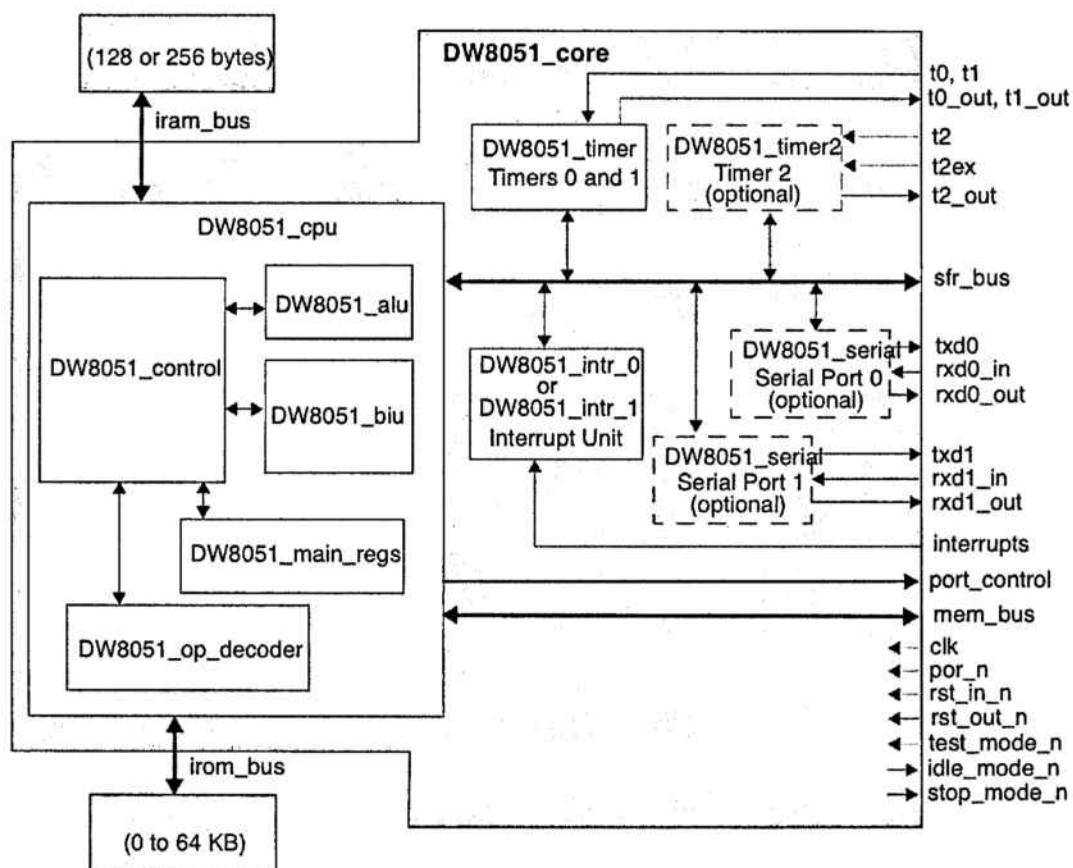


Figura 6.2: Diagrama de blocs del DW8051.

- **Organització de la memòria**

L'organització de la memòria en el DW8051 és similar a la de l'estàndard de la indústria 8051. Hi ha tres diferents àrees de memòria:

- El programa de memòria (ROM)
- Les dades de memòria (RAM externa)

- Registres (RAM interna).

La figura 6.3 mostra el mapa de memòria del DW8051. Memòria de programa i memòria de dades pot ser de fins a 64KB cada un i comparteixen el mateix rang de direcció (0000h-FFFFh), però s'accedeixen de forma diferent. Les adreces de la RAM interna es superposen a les adreces externa inferior de RAM, però s'accedeix a través de diferents tipus d'instruccions.

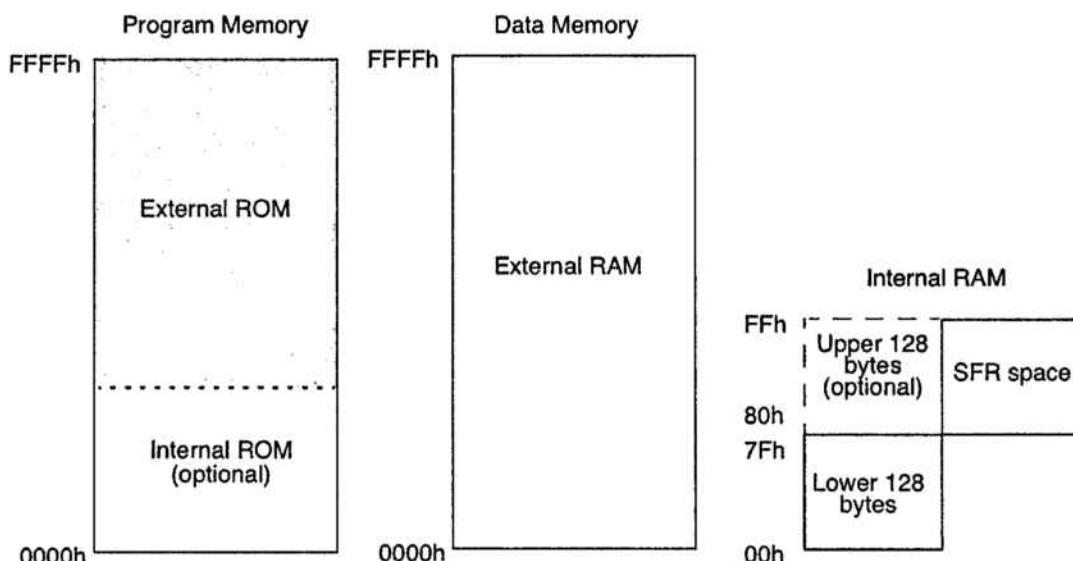


Figura 6.3: Mapa de memòria del 8051.

La memòria de programa es pot implementar com: ROM Interna, ROM externa o combinació de ROM interna i externa. En el xip VECTOR V1 només hi ha ROM externa.

La memòria RAM interna consta de:

- 128 bytes de memòria immediata i registres accessibles amb adreçament directe o indirecte (bus d' adreces *iram_addr* 00h-7Fh)
- 128 bytes superiors de la memòria immediata accessible a través d'adreçament indirecte (bus d'adreces *iram_addr* 80h-FFh). Aquestes són opcionals, en el nostre cas són accessibles.
- 128 Registres de funcions especials (SFRs) accessibles a través d'adreçament directe (bus d'adreces *sfr_addr* 80h-FFh).

Un cop vist per sobre el DW8051, el primer pas per realitzar la implementació de l'electrònica digital de control és generar la *netlist* del DW8051 de *Synopsis* amb la tecnologia de la FPGA a utilitzar, en aquest cas una *Cyclon II* d'Altera. Per generar aquesta *netlist*, s'utilitza l'eina que proporciona *Synopsis* el programa *coreConsultan*. A l'hora de generar la *netlist* s'han de definir una sèrie de paràmetres per tal configurar el DW8051 segons els requeriments del sistema. En el xip VECTOR V1 s'ha optat per:

Taula 6.1: Paràmetres escollits per genera la *netlist* del DW8051.

Nom	Valor	Descripció
Ram_256	1	128 bytes superiors de la memòria RAM interna accessibles.
iram_addr_size	0	No hi ha memòria ROM interna.
Timer2	0	No hi ha temporitzador 2.
Serial	1	1 port serie Port 0.
Extd_intr	0	6 fonts d'interruptió

A l'annex 11.1 s'explica el procés seguit per obtenir la *netlist*.

Un cop generada la *netlist* del DW8051 amb la tecnologia adient, és necessari comprovar que aquesta es pot sintetitzar a la FPGA i que funciona correctament, tot i que al ser una IP no és necessari un test intensiu per comprovar el seu funcionament.

Per realitzar aquesta comprovació s'ha creat un sistema bàsic que consta de:

- **DW8051** de Synopsys: bloc que es verifica i comprova que es sintetitza correctament a la FPGA.
- **Power On Reset**: encarregat de generar el reset del sistema a partir d'un reset extern asíncron.
- **Memòries**: Són IPs pròpies d'Altera.
- **Registre SFR**: registre que s'escriu per mitja del bus intern del sistema.

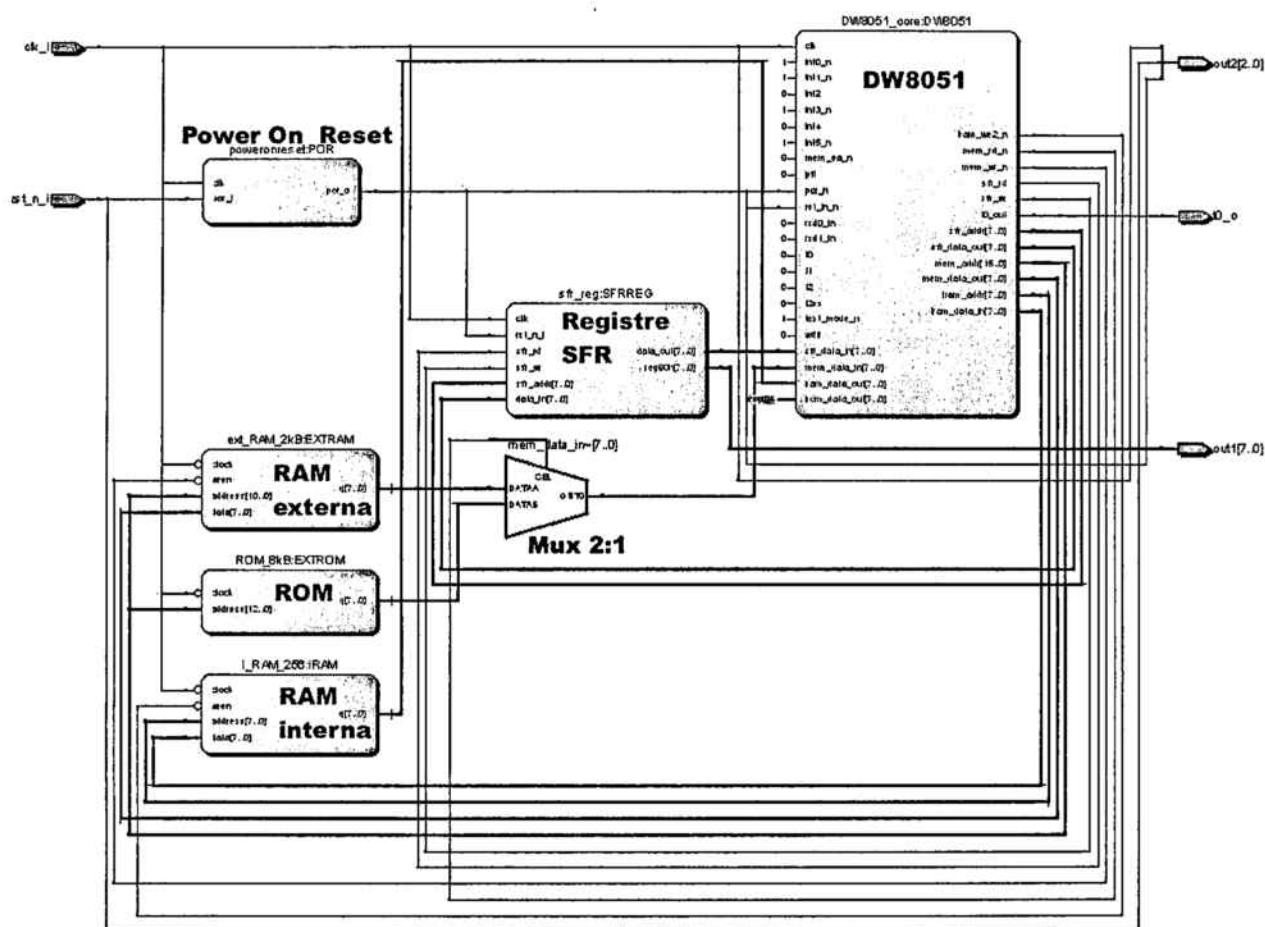


Figura 6.4: Sistema bàsic implementat a la FPGA.

A la figura 6.4 es mostra el diagrama de blocs i les connexions entre els diferents blocs que formen el sistema. Els passos realitzats per crear i sintetitzar aquest sistema a la FPGA es troben descrits a l'annex 11.2.

Els punts crítics d'aquest sistema es troben en les memòries. A continuació s'explica en detall el funcionament de les memòries, ja que, a l'hora d'implementar l'electrònica de control a la FPGA s'hauran d'utilitzar aquestes mateixes memòries i una bona comprensió del funcionament d'aquest sistema bàsic permetrà realitzar la substitució de manera més senzilla i evitar problemes i localitzar-los més fàcilment.

En primer lloc cal examinar com funcionen les memòries generades amb el *Quartus II* que s'utilitzaran. Per això cal fixar-se en les formes d'ona per la lectura i l'escriptura que

proporciona el mateix *Quartus II* en un fitxer HTML (de l'anglès, *HyperText Markup Language*) quan es creen les memòries.

- La forma d'ona de la figura 6.5a, mostra el comportament del disseny en condicions normals de lectura. La lectura succeeix en el flanc ascendent del cicle de rellotge activat i sempre està llegint la posició de memòria indicada per l'adreça.
- La forma d'ona de la figura 6.5b, mostra el comportament del disseny sota condicions normals d'escriptura. El cicle d'escriptura dura des del flanc de pujada del rellotge en què *Wren* està en estat alt fins al següent flanc ascendent de rellotge. Realment l'escriptura a la memòria RAM succeeix en el flanc de baixada del rellotge. Durant un cicle d'escriptura les noves dades flueix a través de la sortida.

S'observa clarament que les memòries són síncrones, per tant requereixen una senyal de rellotge, no tenen senyal de selecció ni senyal d'activació de lectura.

Per realitzar les connexions entre les memòries i el DW8051, es necessari consultar el *datasheet* del DW8051 on s'expliquen les interfícies de connexió així com són les ones de lectura i escriptura per cada memòria.

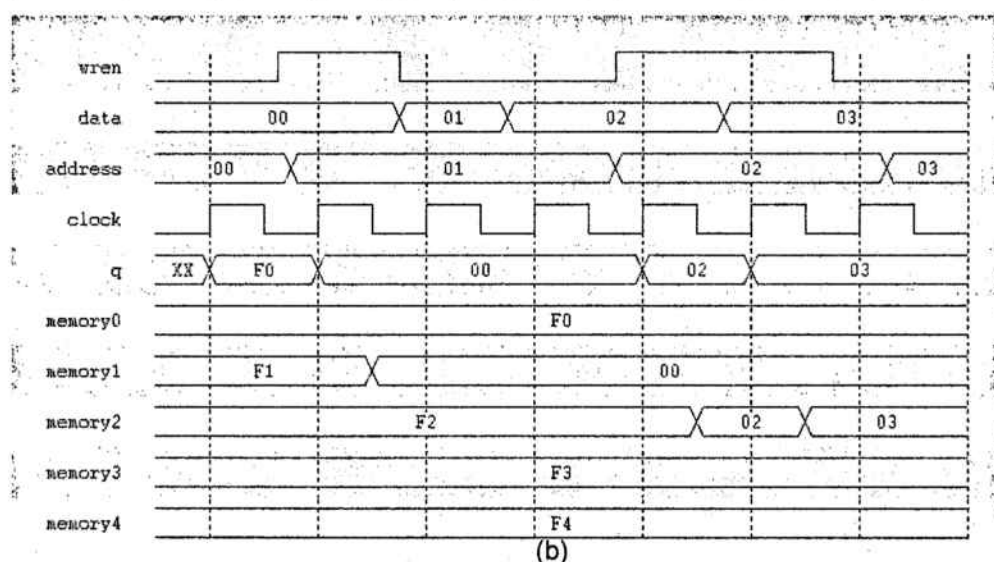
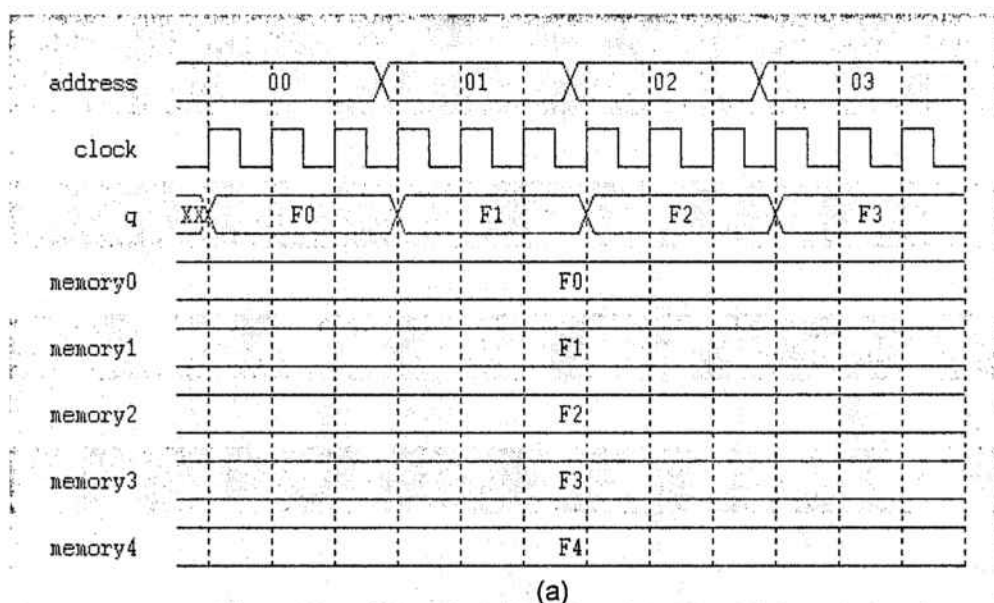


Figura 6.5: Ones per l'operació de (a) lectura (b) escriptura.

- **RAM externa** de 2 KB té les formes d'ona de les figura 6.6 i 6.7.

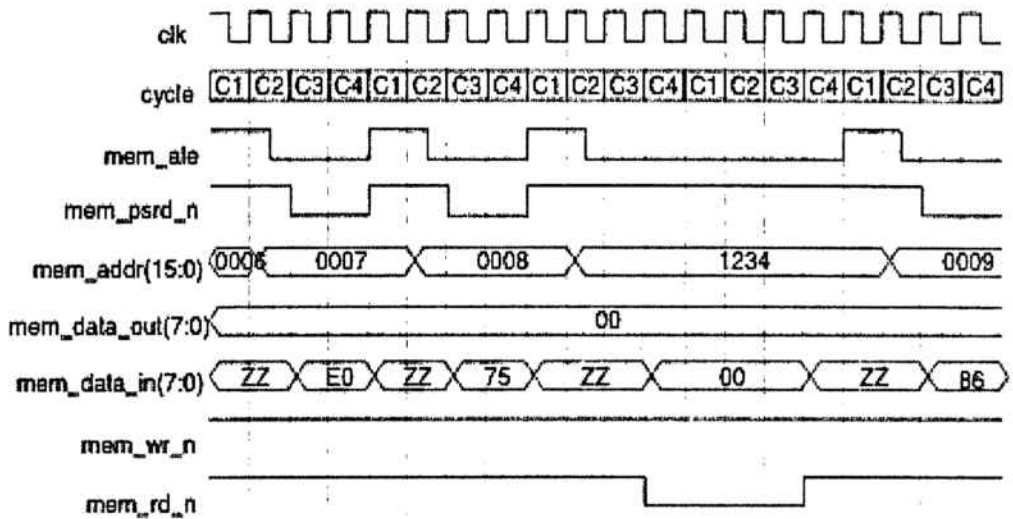


Figura 6.6: Ones per l'operació de lectura.

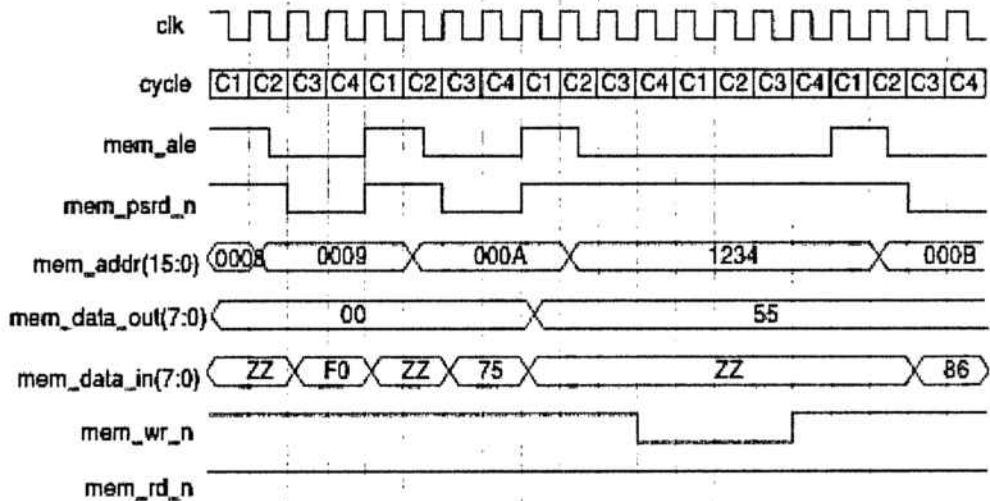


Figura 6.7: Ones per l'operació de escriptura.

Les senyals de la interfície de lectura i escriptura mostrades en les figures 6.6 i 6.7 respectivament, es troben descrites a la taula 6.2.

Taula 6.2: Senyals interfície de la memòria RAM externa.

Nom	Mida	Descripció
mem_addr	13	Bus d'adreces.
mem_data_in	8	Bus d'entrada de dades al DW8051.
mem_data_out	8	Bus de sortida de dades del DW8051.
mem_wr_n	1	Habilita l'escriptura a la memòria. Les dades han de ser mostrejades en el flanc de pujada de <i>mem_wr_n</i> .
mem_rd_n	1	Habilita la lectura de les dades. El mostreig de les dades entrants es realitza en el flanc de pujada de <i>mem_rd_n</i> .

Com es pot comprovar les senyals que habiliten la lectura i la escriptura estan negades, i per tant això s'ha de tenir present quan es connecten les memòries. Per

altra banda, es pot observar que tan en la lectura com en l'escriptura les senyals canvien d'estat en els flanc ascendent del rellotge. Per tant, s'ha d'invertir el senyal de rellotge entrant a la memòria per tal de respectar els temps de *setup* i de *hold* i permetre així una lectura i escriptura de les memòries correcta.

- **RAM interna de 256 bytes.**

Les senyals de la interfície de la memòria RAM interna es troben descrites a la taula 6.3.

Taula 6.3: Senyals interfície de la memòria RAM interna.

Nom	Mida	Descripció
<i>iram_addr</i>	8	Bus d'adreces. Vàlid durant un cicle de rellotge per la lectura i 2 cicles de rellotge per l'escriptura.
<i>iram_data_in</i>	8	Dades que s'escriuen a la memòria RAM interna, vàlides durant el segon cicle de rellotge del cicle d'escriptura.
<i>iram_data_out</i>	8	Dades que es llegeixen de la RAM interna.
<i>iram_rd_n</i>	1	Senyal informativa que indica que s'està produint una lectura a memòria RAM interna – actiu nivell baix. No s'ha d'utilitzar aquesta senyal per dissenyar <i>hardware</i> . La operació de lectura de la memòria RAM interna ha de ser funció només del bus d'adreces d'aquesta (<i>iram_addr</i>).
<i>iram_we1_n</i>	1	Actiu per nivell baix durant dos cicles de rellotge (tot el cicle d'escriptura). Indica que l'adreça (<i>iram_addr</i>) és vàlida.
<i>iram_we2_n</i>	1	Actiu per nivell baix durant la segona meitat del cicle d'escriptura (1 cicle de rellotge). Indica que la dada a escriure (<i>iram_data_in</i>) és vàlida.

Les senyals que intervenen en el procés d'escriptura són les senyals que es mostren a la figura 6.8. La senyal *iram_we1_n* s'activa en C4 per indicar que hi ha una adreça vàlida a la RAM interna *iram_addr*. La senyal *iram_we2_n* s'activa en C1 per indicar que hi ha dades vàlides a *iram_data_in*. Tenint present com funciona les memòries utilitzades, es pot veure que la senyal per activar l'escriptura és la *iram_we2_n*, i que l'escriptura s'ha de realitzar just en el flanc de baixada de rellotge que entra en el DW8051 mentre la senyal *iram_we2_n* està activada. Per tant s'ha de invertir la senyal de rellotge que entra a la memòria tal i com s'ha fet en la RAM externa.

Pel que fa a la lectura, les senyals que intervenen són *iram_addr* i *iram_data_out*. El senyal *iram_rd_n* és merament informatiu, i no ha de ser emprat com a interfície. Totes les operacions de lectura a la RAM interna són funcions del bus *iram_addr* això duu a que la lectura sempre ha d'estar habilitada.

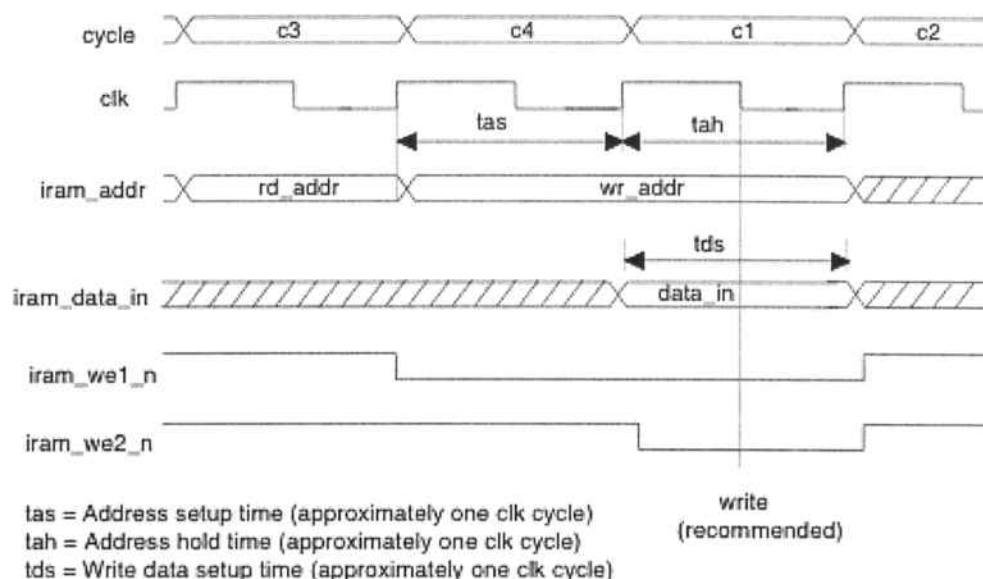


Figura 6.8: Ones per l'escriptura de la RAM interna.

- **Memòria de programa de 8 KB (ROM externa).**

Les senyals de la interfície de lectura, veure figura 6.9, són:

Taula 6.4: Senyals interfície de lectura de la memòria de programa.

Nom	Mida	Descripció
mem_addr	13	Bus d'adreces.
mem_data_in	8	Bus d'entrada de dades al DW8051.
mem_psrđ_n	1	Habilita la lectura de les dades. El mostreig de les dades entrants es realitza en el flanc de pujada d'aquesta senyal.

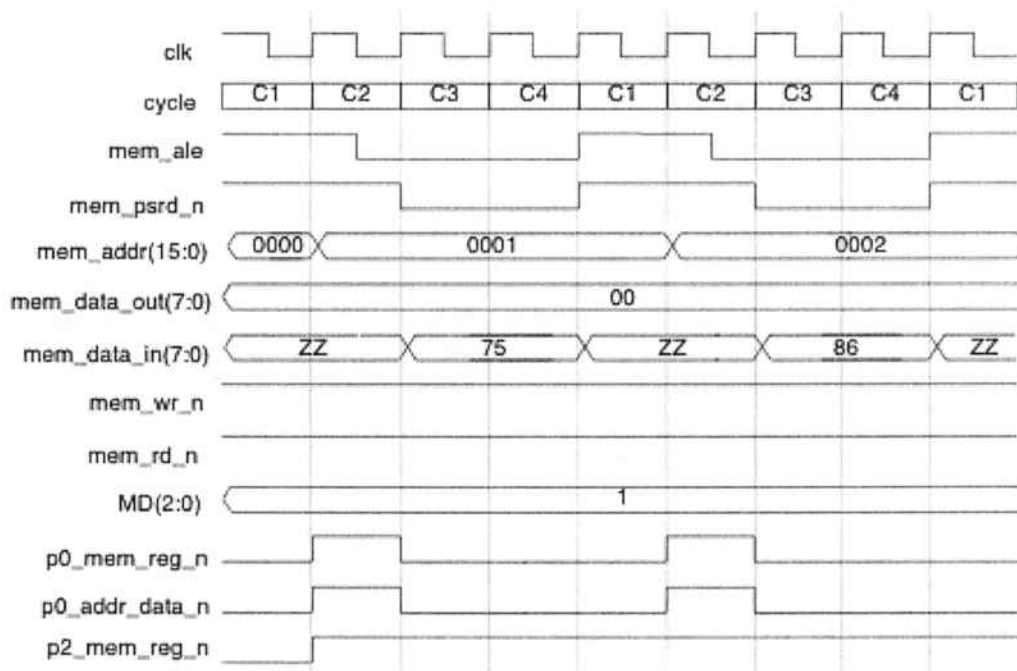


Figura 6.9: Ones per la lectura de la memòria de programa (ROM externa).

Tenint present que la memòria que s'utilitza com a ROM externa és síncrona, no té senyal d'activació de lectura, la lectura del bus d'adreces és produïx en el flanc de pujada del rellotge entrant i que el DW8051 realitza la lectura també en flanc de pujada del rellotge. És necessari invertir el rellotge entrant a la memòria per garantir que el procés de lectura es realitzarà correctament, tal i com passava amb les memòries RAMs.

El DW8051 utilitza el mateix bus d'adreces *mem_addr* i el mateix port d'entrada de dades *mem_data_in* per les memòries ROM externa i la RAM externa. Per aquest motiu és necessari un descodificador d'adreces per habilitar la dada correcte segons la memòria a la que accedeix el DW8051, en aquest cas és un multiplexor 2 a 1 on la senyal de selecció és *mem_rd_n* la senyal que utilitza el DW8051 per habilitar la lectura de la memòria RAM externa de 2 KB.

Un altre punt crític a tenir present és el **bus intern del sistema**, que correspon al bus SFR per perifèrics externs del DW8051, ja que és la interfície de connexió entre el cervell del sistema, DW8051, i la resta de blocs controladors del sistema. Al *datasheet* del DW8051 trobem que les senyals del bus són:

Taula 6.5: Senyals del bus SFR.

Nom	Mida	Descripció
sfr_addr	8	Bus d'adreces pels perifèrics externs.
sfr_data_out	8	Dades de sortida per els registres SFR. La dada és vàlida quan <i>sfr_wr</i> és activat – nivell alt.
sfr_data_in	8	Dades d'entrada procedents dels perifèrics externs (registres SFR). La dada és mostrejada en el següent flanc de pujada del rellotge quan el <i>sfr_rd</i> és activat – nivell alt.
sfr_wr	1	Senyal d'escriptura per els registres SFRs, activa durant un cicle de rellotge
sfr_rd	1	Senyal de lectura per els registres SFRs, activa durant un cicle de rellotge

En les figures 6.10 i 6.11 es pot observar els temps de lectura i escriptura del bus SFR respectivament.

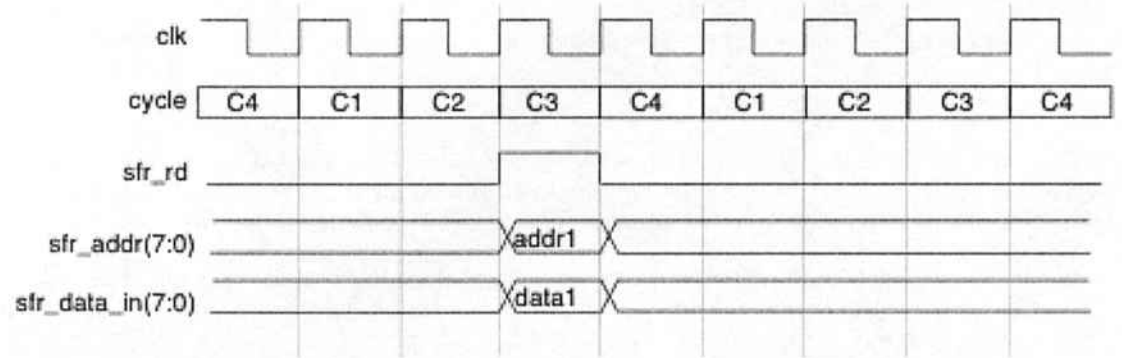


Figura 6.10: Temps de lectura del bus SFR.

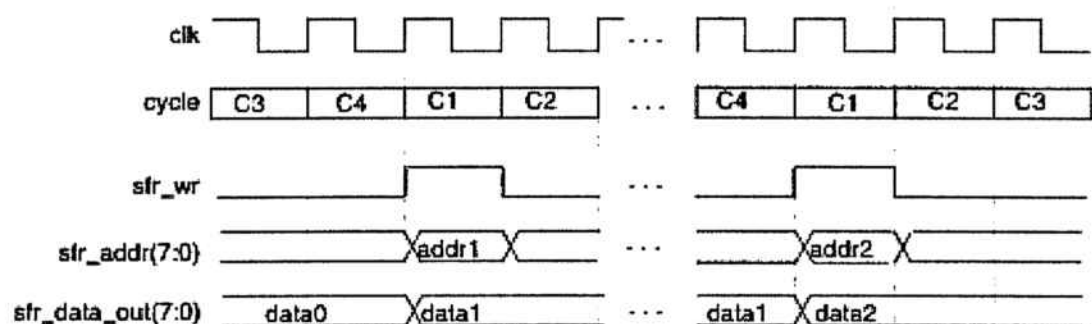


Figura 6.11: Temps d'escriptura del bus SFR.

En el sistema creat només s'ha disposat un registre SFR, concretament el corresponent a l'adreça 80h, i per tant la connexió és directa i senzilla.

L'últim pas necessari és crear un programa que utilitzi els recursos del sistema. Per realitzar el codi i compilar-lo s'utilitza l'entorn de programació *uVision3* de la casa *Keil*, a l'annex 11.3 hi ha descrits els passos a seguir per obtenir el fitxer que carregarem a la memòria de programa i el codi font.

El programa creat i carregat a la memòria utilitza un dels temporitzadors del DW8051, per escriure cada 307us un valor diferent al registre SFR 80h a la figura 6.12 es mostra el diagrama de flux del programa. A la figura 6.13 es pot comprovar com cada cop que salta el temporitzador, senyal *t0_overflow*, el processador escriu un valor diferent al registre SFR 80h, senyal de color blanc. Això demostra que les memòries estan ben connectades amb el processador i que la *netlist* generada funciona correctament.

Per altra banda, cal tenir present que segons com s'hagin declarat les variables dins el codi, aquestes s'emmagatzemen en una posició del mapa de memòria o en un altre. Hi ha 3 posicions possibles on es poden emmagatzemar:

- **DATA:** la dada s'emmagatzema a la memòria RAM interna amb adreçament directe.
- **IDATA:** la dada s'emmagatzema a la memòria RAM interna amb adreçament indirecte.
- **XDATA:** la dada s'emmagatzema a la memòria RAM externa.

Per tant és necessari repetir el test forçant que la variable que varia dins la interrupció estigui situada en cadascuna d'aquestes posicions. D'aquesta manera es pot verificar que totes les memòries funcionen correctament.

El *uVision3* permet especificar si la variable és del tipus *data*, *idata* o *xdata* quan és fa la declaració de la variable. Per exemple per declarar una variable de nom *count* com a *char* (8 bits) i que estigui a la memòria RAM interna d'adreçament indirecte seria: ***char idata count;***

Repetint el test pels 3 possibles casos s'obté sempre el mateix resultat que mostra la figura 6.13. Així doncs, es pot concloure que el sistema i per tant la *netlist* del DW8051 generada funcionen correctament.

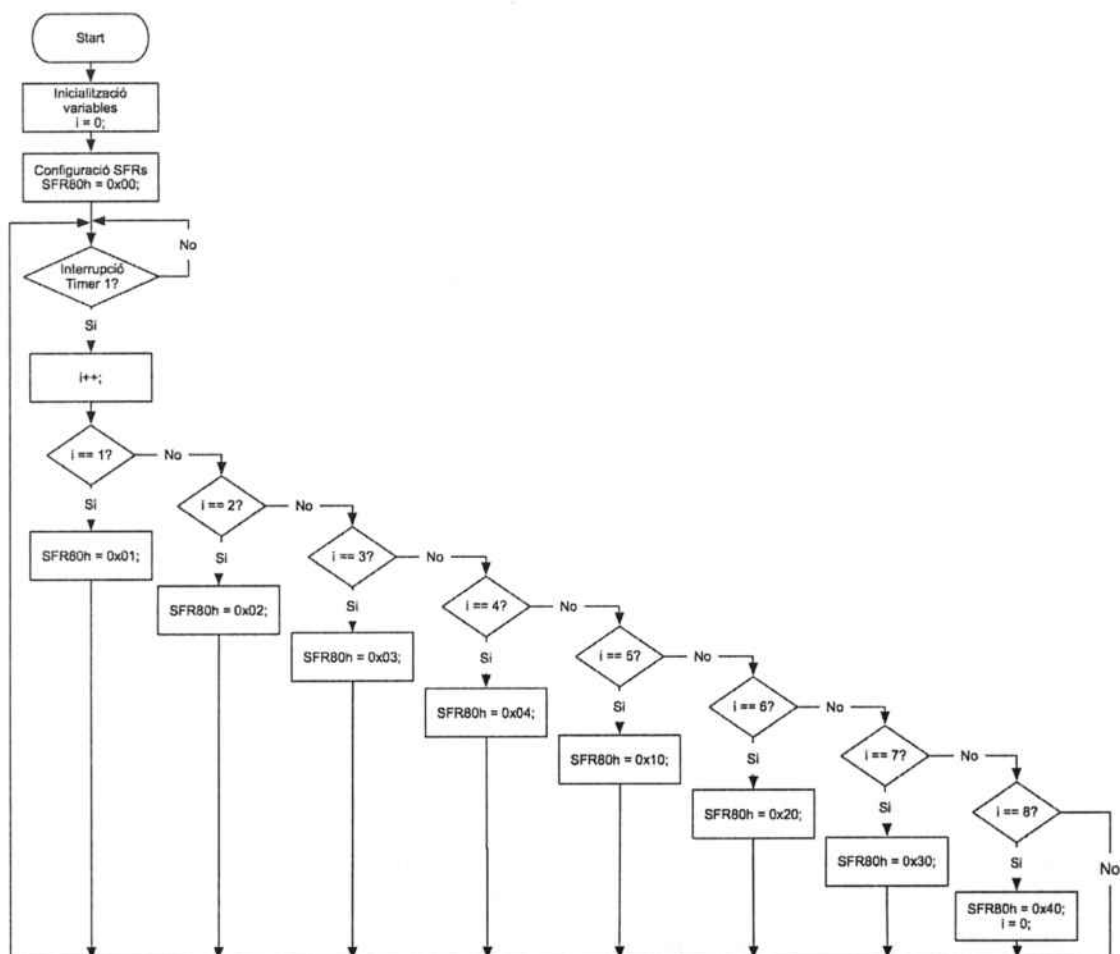


Figura 6.12: Diagrama de flux.

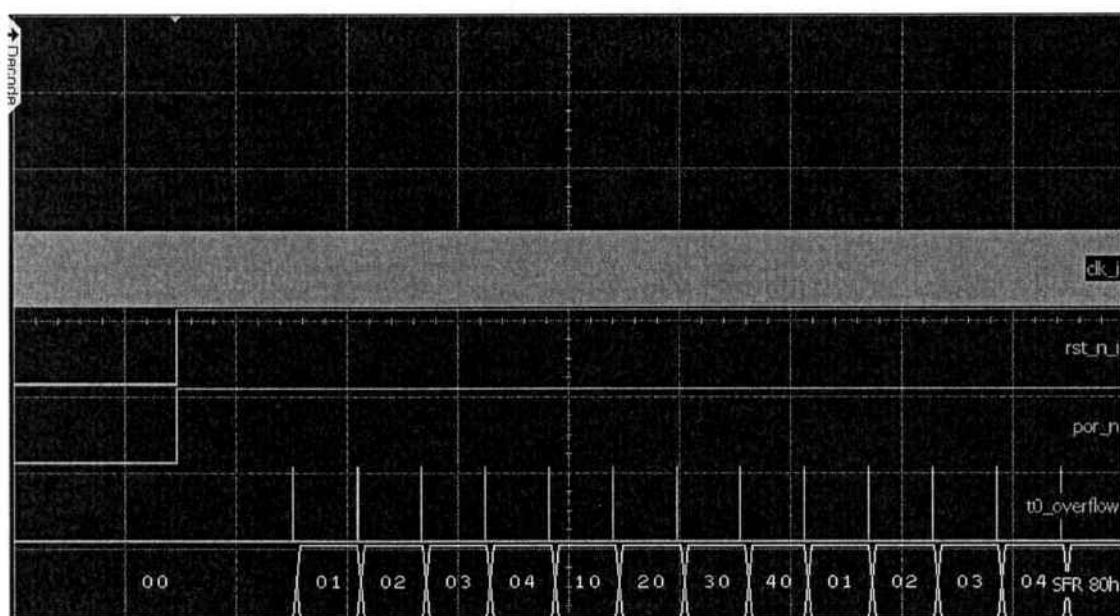


Figura 6.13: Captura de l'oscil·loscopi.

6.1.2 Implementació d'electrònica digital de control

Verificat el correcte funcionament de la *netlist* del DW8051 i analitzat el funcionament de les memòries així com del bus SFR, el següent pas és implementar l'electrònica digital de control a la FPGA amb les memòries del *Quartus II* i comprovar que el sistema arrenca i funciona.

Per realitzar la implementació és genera un nou projecte amb el *Quartus II* i es realitzen els canvis necessaris per poder instanciar les memòries síncrones generades amb el *Quartus II*. En trets generals, els passos bàsics a seguir són els mateixos que els descrits per el sistema creat a l'apartat 6.1.1. Per altra banda, és necessari mirar el *datasheet* del xip VECTOR V1, per poder veure quines entrades són necessàries per poder posar en marxa el sistema. A la figura 6.14 es mostra el *top* del sistema generat que conté l'electrònica digital de control. Es poden veure les entrades necessàries per controlar el comportament del sistema, descrites a la taula 6.10, i les sortides bàsiques per poder dur a terme els testos.

Taula 6.10: Senyals d'entrada.

Nom	Descripció
Clk	Relloige del sistema 10MHz
Rst_n	Reset del sistema – Actiu nivell baix
Eeprom_en	Quan és '1' habilita l'escriptura externa de la memòria de programa. Quan és '0' el processador controla la memòria de programa.
Test_en	Quan és '1' habilita el mode test. El sistema es controla des del exterior. Quan és '0' el processador controla el sistema.

La resta d'entrades no són necessàries en aquest test i per tant no s'utilitzen, per aquest motiu se'ls ha donat el valor per defecte que no genera cap esdeveniment.

Pel que fa a les sortides es pot observar com només es té el rellotge del sistema, i dos sortides de 8 bits Out1 i Out2, que s'utilitzen per treure les senyals internes del sistema que no tenen sortida directe a l'exterior i que són necessàries per comprovar el correcte funcionament dels diferents blocs. A més hi ha les línies del bus I2C (SCL i SDA), que són ports bidireccionals.

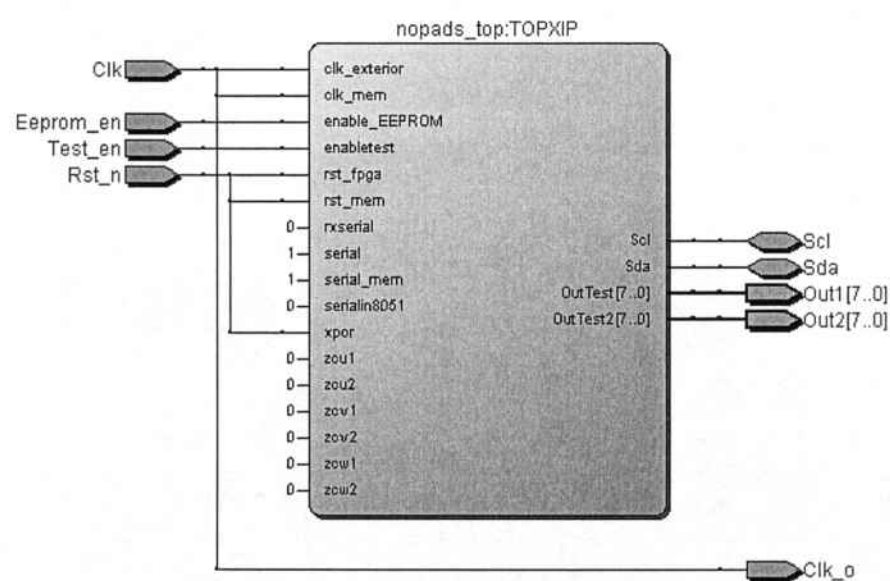


Figura 6.14: Top del sistema de l'electrònica digital.

Per comprovar el que el sistema arrenca i que el canvi de les memòries s'ha realitzat correctament, s'ha creat un petit programa que utilitza el temporitzador 1 (*Timer1*) del

DW8051, per escriure al SFR de propòsit general (SFRGP) de que disposa el sistema. A la figura 6.15 es mostra el diagrama de flux del codi realitzat, a l'annex 11.4 s'hi troba el codi font.

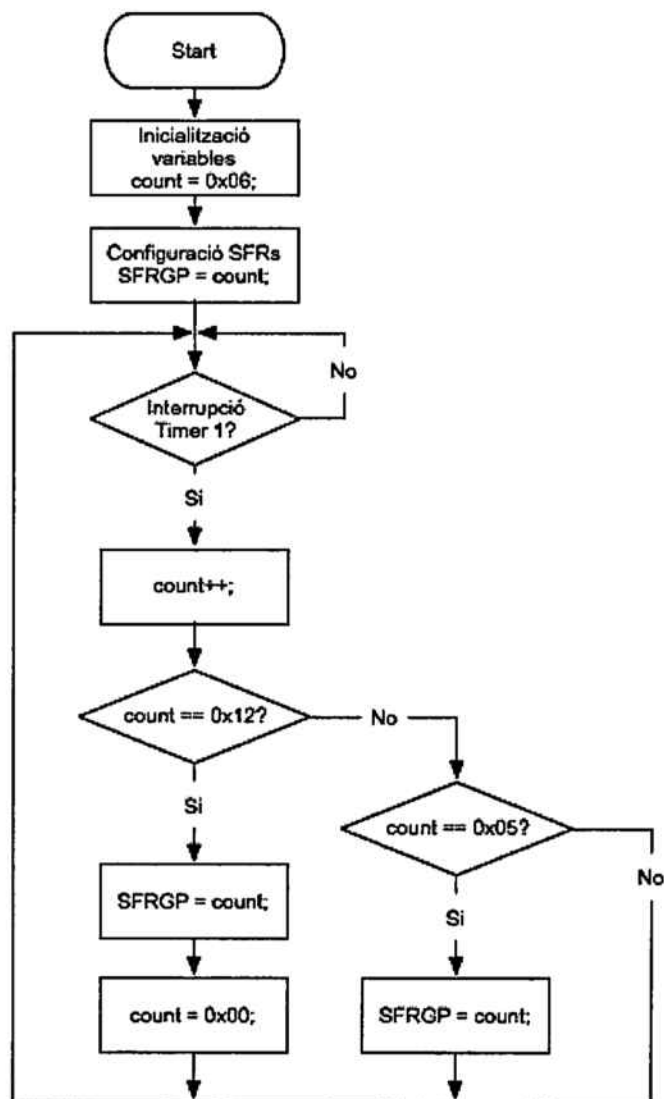


Figura 6.15: Diagrama de flux.

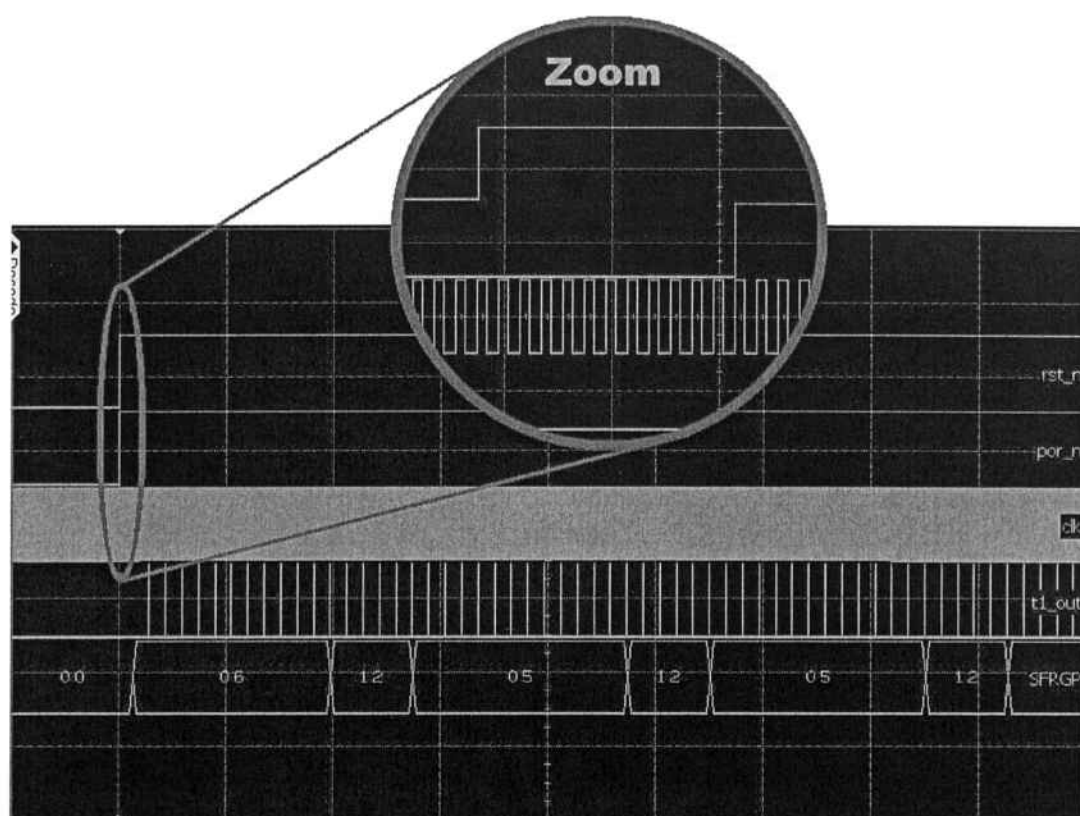


Figura 6.16: Captura de l'oscil·loscopi (escala de temps 2ms/divisio).

A la figura 6.16, es pot observar com un cop s'aixeca el *reset* del sistema, aquest arrenca sense cap problema i executa el programa. En el zoom de la mateixa figura és pot apreciar com el senyal del *power on reset* es genera amb un cert retard, per tal de permetre que el rellotge hagi arribat a tot el sistema a abans d'arrancar el processador.

S'ha repetit el test fent que la variable *count* estigui en les diferents posicions dins el mapa de memòria: ***data***, ***idata***, ***xdata***. El resultats obtinguts en cadascun dels 3 tests són els iguals que es mostren a la figura 6.16. Per tant es pot concloure que els canvis realitzats per poder implementar les memòries del *Quartus II* no han afectat al sistema i que aquestes estan connectades correctament.

6.1.3 Tests i depuració dels blocs dels sistema

6.1.3.1 Test de les interfícies de connexió de les memòries

Per poder testear les interfícies de connexió de les memòries del sistema, entenent per interfícies de connexió les senyals que es generen per controlar les memòries així com els corresponents blocs CS (de l'anglès, *Chip Select*) i el *Boot Loader*, és necessari crear una memòria que funcioni com les memòries d'AMS que aniran al xip VECTOR V1.

Degut a que la instrucció, `reg [DATA_SIZE-1:0] mem[0:MEM_SIZE-1]`, utilitzada en verilog per modelitzar memòries no és sintetitzable pel *Quartus II*, s'ha de crear la memòria com si fos un gran banc de registres de forma manual. Però degut a les limitacions de la FPGA només es poden crear bancs de 256 registres de 8 bits cadascun, és a dir, que només podrem treballar amb memòries de 256 bytes en lloc de 8 KB per la memòria de programa i de 2 KB per la memòria RAM externa. Per la memòria RAM interna, que és la més crítica ja que conté registres que utilitza el processador per realitzar les seves operacions, no hi haurà cap problema ja que és de 256 bytes.

Tot i aquestes limitacions, 256 bytes de memòria són suficients per realitzar els tests, ja que el codi dels programes utilitzats no requereix més memòria i la memòria RAM externa no s'hi emmagatzemaran gaires variables.

❖ Generació de la memòria 256 bytes

Per generar la memòria de 256 bytes ens basem amb el *datasheet* de la memòria d'AMS de 256 bytes. A la figura 6.17 es mostra el diagrama de blocs de la memòria on es veuen les senyals d'entrada i sortida que es troben descrites a la taula 6.11.

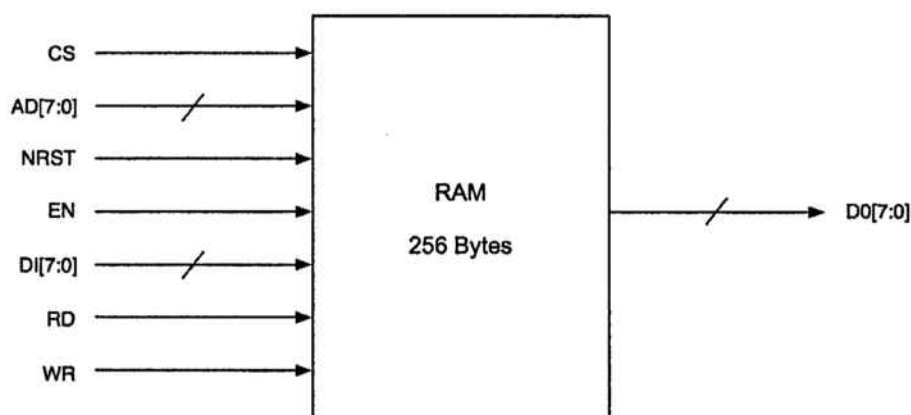


Figura 6.17: Diagrama de blocs de la memòria RAM.

Taula 6.11: Senyals d'entrada i sortida de la memòria RAM d'AMS.

Nom	Mida	I/O	Descripció
CS	1	In	El flanc de pujada del CS estableix un cicle d'operació depenen de la sol·licitud de WR o RD.
RD	1	In	Lectura activada – Actiu per nivell alt.
WR	1	In	Espectura activada – Actiu per nivell alt.
AD[7:0]	8	In	Bus d'adreces.
NRST	1	In	Para o desactiva les operacions de la memòria. Actiu per nivell baix.
EN	1	In	Posa en alta impedància la senyal de sortida D0[7:0]. Actiu per nivell alt.
DI[7:0]	8	In	Bus d'entrada de dades.
D0[7:0]	8	Out	Bus de sortida de dades.

De la descripció del senyals i de l'observació dels diagrames temporals del *datasheet*, es pot concloure que el senyal CS és el que activa realment les operacions d'espectura i lectura i que per tant la resta de senyals han d'estar definides abans del flanc de pujada del CS, respectant sempre els temps de *setup* i *hold*.

❖ T1: Test de la interfície de la memòria RAM interna

Per la realització d'aquest test es seguirà la mateixa metodologia que en el test de l'apartat 6.1.2. El codi que s'executarà és el mateix que en l'apartat 6.1.2 i s'anirà variant la posició de la variable *count* dins el mapa de memòria, *xdata*, *data*, *idata*. D'aquesta manera es pot utilitzar el comportament del sistema amb les memòries del *Quartus II* com a punt de control per comprovar si el sistema funciona correctament.

El canvi realitzat en el disseny és la substitució de la memòria RAM interna síncrona del *Quartus II* per la creada anteriorment. La resta de memòries són les del *Quartus II*, de manera que es puguin detectar els possibles errors de funcionament de la interfície de la memòria RAM interna.

Un cop fet el canvi, es carrega el programa a la memòria i s'executa amb la variable *count* en les diferents posicions de memòria.

- **Variable *count* a XDATA.**

A la figura 6.18 es pot observar com va saltant la interrupció del temporitzador 1 senyal *t1_out*. Pel que fa al registre SFRGP s'aprecia que no s'ha inicialitzat a 06h, sinó que s'ha quedat amb el valor per defecte, 00h, i que ha començat la seqüència de manera diferent a l'esperada, si ho comparem amb la figura 6.16.

Seqüència esperada: 06 → 12 → 05 → 12 → 05 → 12...

Seqüència obtinguda: 00 → 05 → 12 → 05 → 12 → 05...

Per tant, es pot deduir que la variable *count* que s'hauria d'haver inicialitzat a 06h s'ha inicialitzat a 00h, però a part d'això la resta del codi s'executa correctament, i la variable augmenta amb normalitat dins la interrupció.

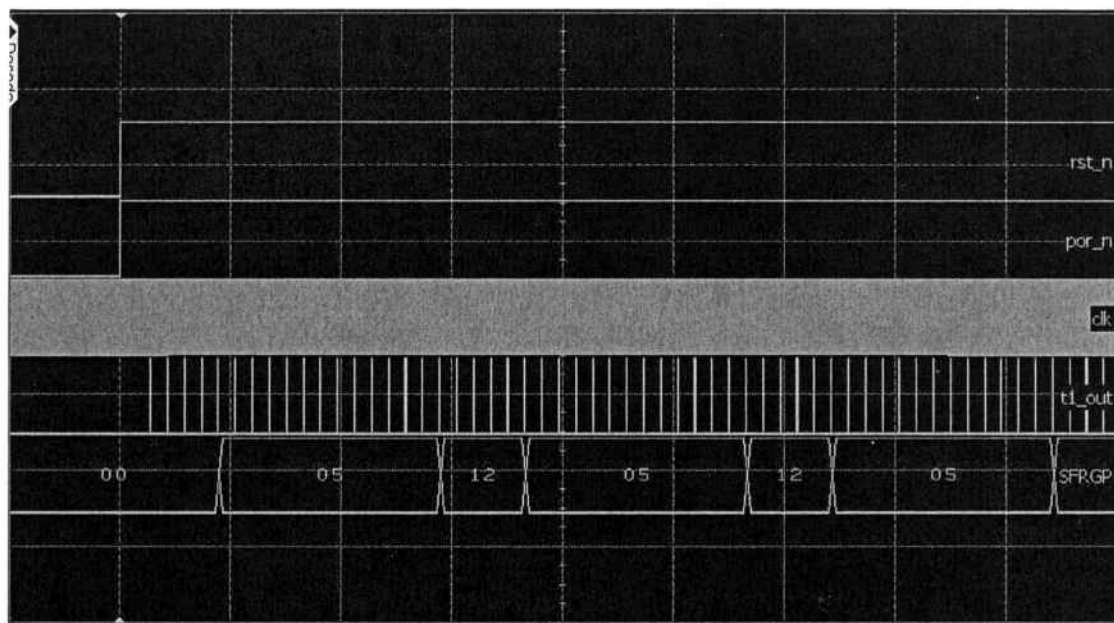


Figura 6.18: Captura oscil·loscopi (escala de temps 2ms/divisio).

- **Variable *count* a DATA**

A la figura 6.19 s'observa com la variable *count* s'inicialitza correctament a 06h però que ho fa uns 2 ms més tard si ho comparem amb la figura 6.16, i el mateix passa amb la interrupció, *t1_out*, que comença però amb el mateix retard.

Tot i aquest retard de 2 ms aproximadament, el sistema acaba executant el codi de forma correcta ja que la seqüència obtinguda és la correcta.

Seqüència esperada: 06 → 12 → 05 → 12 → 05 → 12...

Seqüència obtinguda: 06 → 12 → 05 → 12 → 05 → 12...

Per tant, es pot deduir que el sistema no acaba de funcionar del tot bé encara que s'acabi executant la seqüència amb retard. El que fa pensar que hi ha algun problema amb la memòria interna.

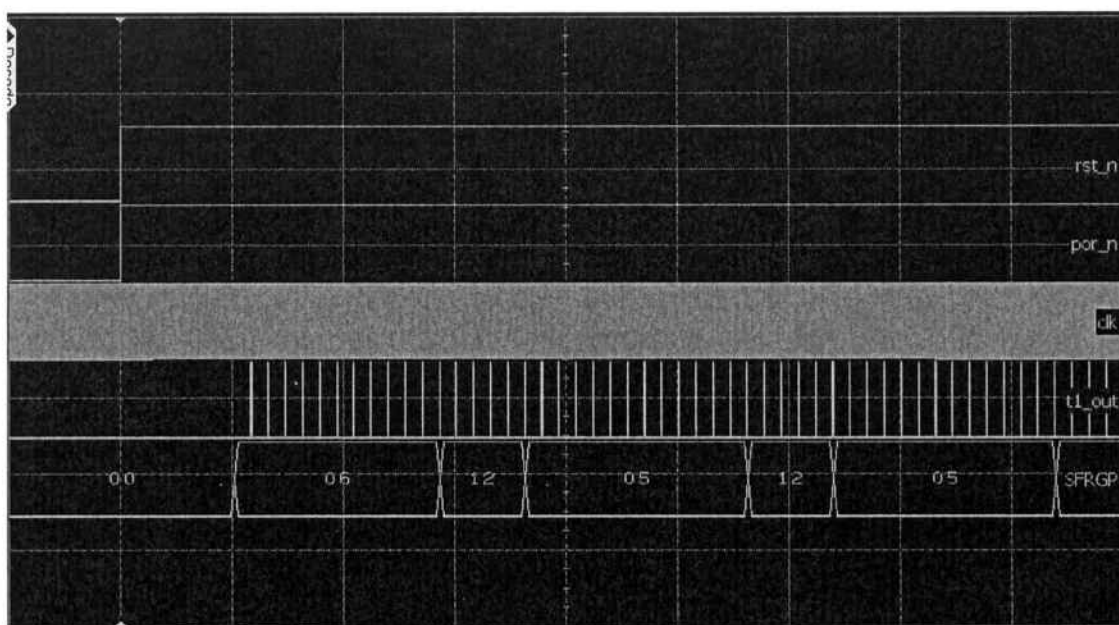


Figura 6.19: Captura d'oscil·loscopi (escala de temps 2ms/divisio).

- **Variable *count* a IDATA**

A la figura 6.20 es pot observar que la interrupció s'acaba activant periòdicament però amb el mateix retard de 2 ms que en el punt anterior. La diferència en aquest cas és que la variable *count* no s'inicialitza correctament, ja que, es veu com el valor del SFRGP passa de 00h a 02h i ja no varia més.

Per tant, es dedueix que els registres SFRs que configuren el temporitzador 1 s'han configurat bé, però que totes les operacions relacionades amb la variable *count* no s'executen correctament.

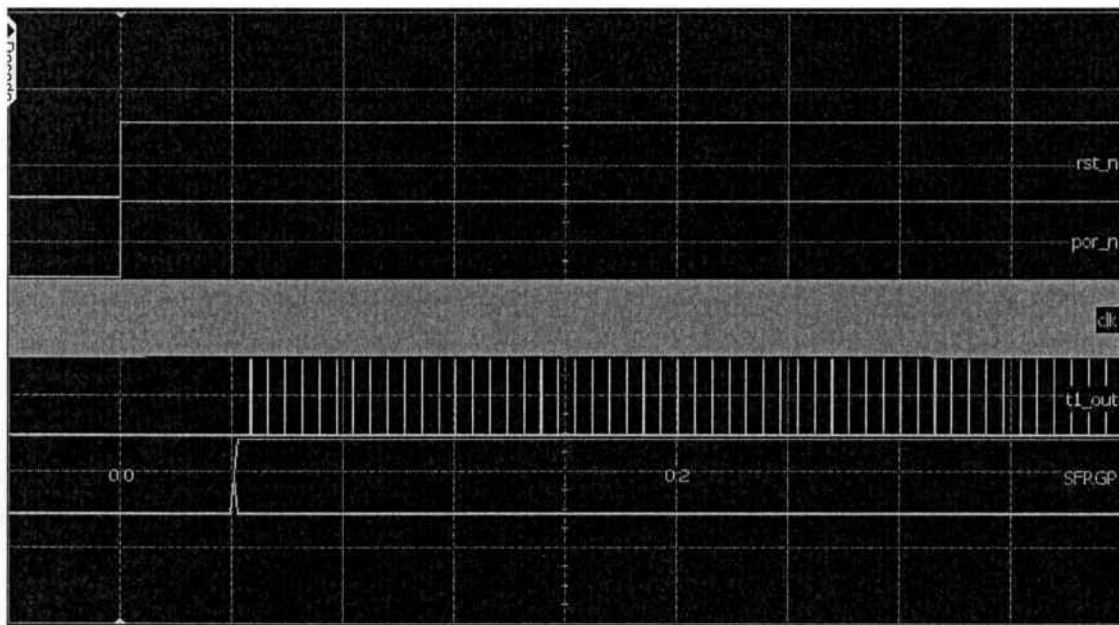


Figura 6.20: Captura d'oscil·loscopi (escala de temps 2ms/divisio).

Com s'ha pogut veure en els 3 tests realitzats en cap dels casos el sistema ha funcionat com s'esperava. Per tant és lògic pensar que hi ha un problema amb la interfície de connexió de la memòria RAM interna, ja que la RAM interna està testejada i funciona correctament.

Cal repetir els testos, però aquesta vegada examinant les senyals que controlen la RAM interna.

- **Variable count a XDATA**

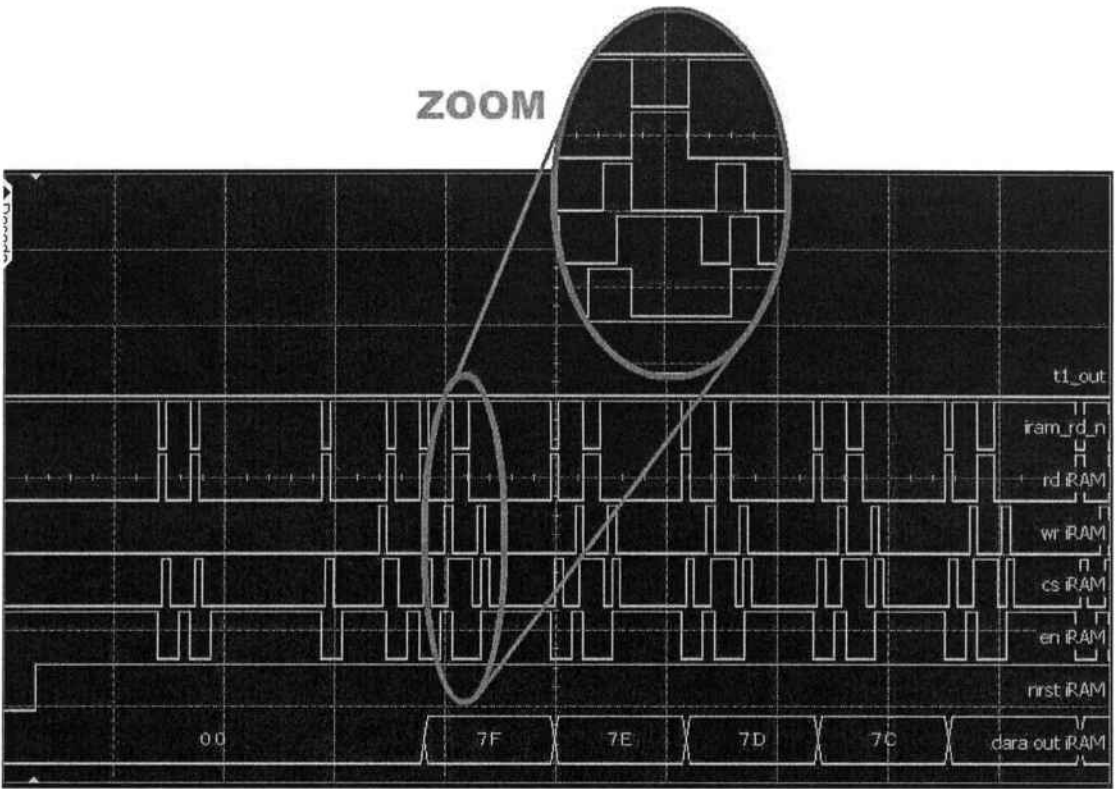


Figura 6.21: Captura de l'oscil·loscopi.

- **Variable count a DATA**

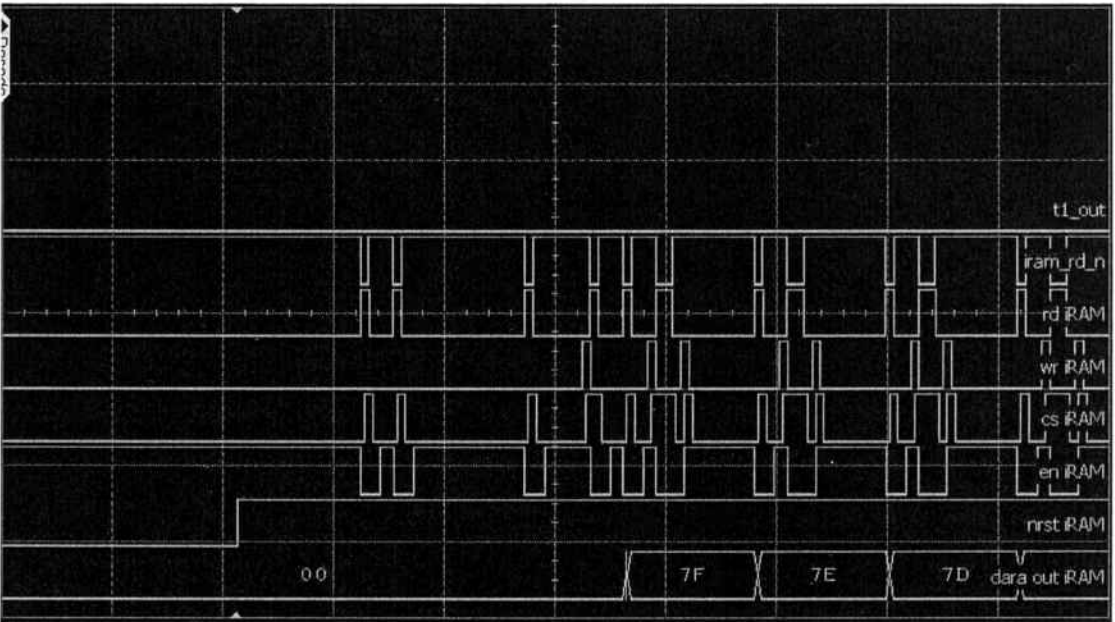


Figura 6.22: Captura de l'oscil·loscopi.

- **Variable count a IDATA**

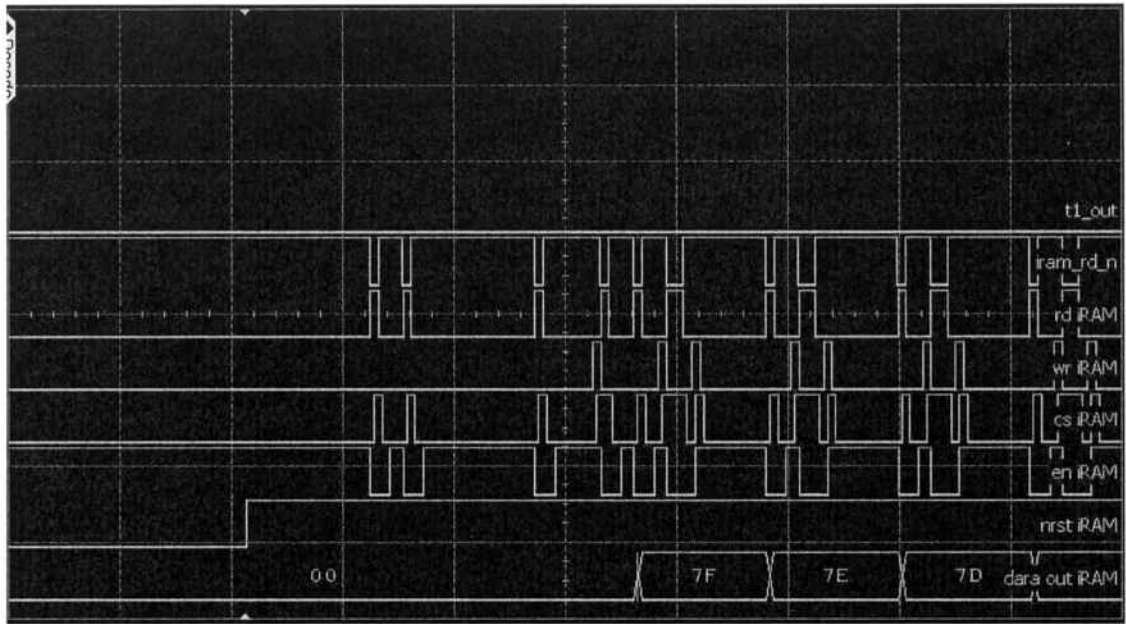


Figura 6.23: Captura de l'oscil·loscopi.

Si s'observen les figures 6.21, 6.22 i 6.23 es pot veure com en diferents llocs, és realitzen de forma consecutiva les operacions d'escriptura, lectura i escriptura, però la senyal CS només s'activa per les dues escriptures, no baixa i torna a pujar per la lectura de manera que aquesta no és realitza. Per tant, no es compleix el comportament esperat de les senyals de control explicat anteriorment.

Examinant el codi on es troba instanciada la memòria, es pot veure com està connectada la memòria RAM interna. A la figura 6.24 es mostra el diagrama de blocs simplificat de la interfície de connexió de la memòria.

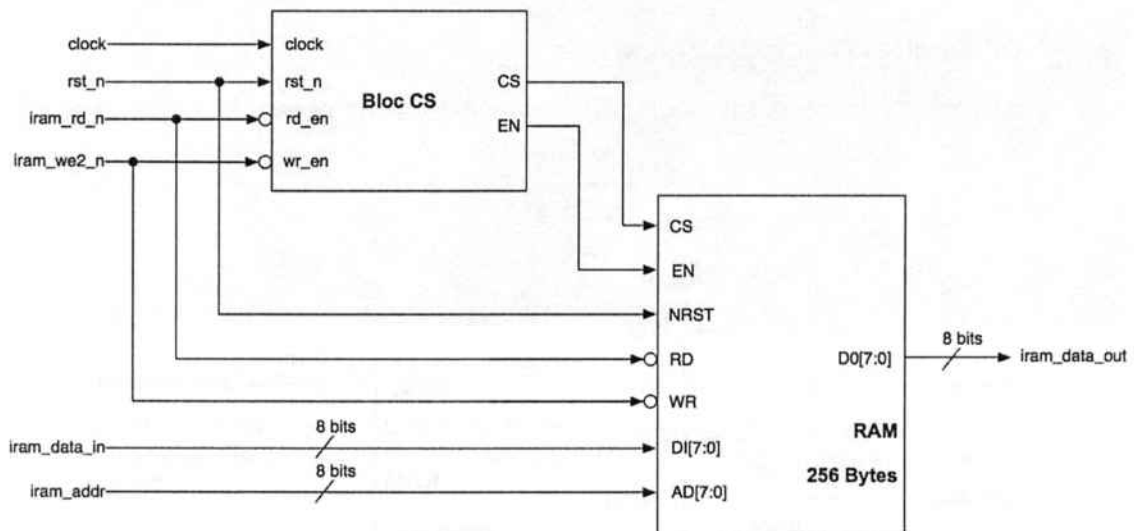


Figura 6.24: Diagrama de blocs de la interfície de la memòria RAM interna.

Analitzant detingudament la interfície, s'observa que:

- El bloc CS RAM interna, és l'encarregat de generar les senyals de CS i EN a partir de les senyals, *clock*, *rst_n*, *iram_rd_n* i *iram_we2_n*.
- La senyal *iram_rd_n* també s'utilitza com a senyal d'habilitació de la lectura.

A l'utilitzar la senyal *iram_rd_n* s'està comenten un error, ja que aquesta senyal no serveix per indicar que s'habilita la lectura de la memòria a l'hora de dissenyar hardware, sinó que és una senyal purament informativa, tal i com ja s'havia vist anteriorment en l'estudi de les interfícies de connexió del DW8051 amb les memòries.

Per altra banda, el bloc CS també té algun error, ja que com s'ha vist a les figures 6.21, 6.22 i 6.23, la senyal de CS no feia la transicions d'estat alt-baix-alt quan hi havia una escriptura seguida immediatament d'una lectura.

Aquest mal funcionament realment es va detectar directament amb el xip VECTOR V1 al veure el problema que no s'inicialitzaven les variables globals. Gràcies al prototipatge en la FPGA es va poder esbrinar d'on venia aquest mal funcionament, i es va solucionar pel següent xip el VECTOR V2 i per en la FPGA per poder desenvolupar bé el software.

Cal redissenyar la interfície de connexió de la memòria RAM interna tenint present:

1. La lectura sempre ha d'estar habilitada ja que és funció única i exclusivament del bus d'adreces *iram_addr*.
2. L'escriptura s'ha de realitzar just en el flanc de baixada de rellotge que entra en el DW8051 mentre la senyal *iram_we2_n* està activada, ja que les senyals que surten del DW8051 canvien d'estat en el flanc de pujada.

D'aquestes dues condicions se'n pot extreure que:

1. La senyal CS és el rellotge del sistema invertit. D'aquesta manera es garanteix en els processos d'escriptura i lectura es compleixin el temps de *setup* i de *hold*.
2. La senyal WR ha de ser la senyal *iram_we2_n* invertida, ja que indica que la dada a escriure és vàlida.
3. La senyal RD ha de ser la senyal *iram_we2_n*, perquè sempre que no s'està escrivint a la memòria s'està llegint.
4. La senyal EN ha de ser sempre '0' o pot ser també *iram_we2_n*. Pel mateix motiu que RD.
5. La senyal NRST és directament el reset del sistema, *rst_n*.

Les noves connexions de la interfície la memòria RAM interna queda com es mostra a la figura 6.25.

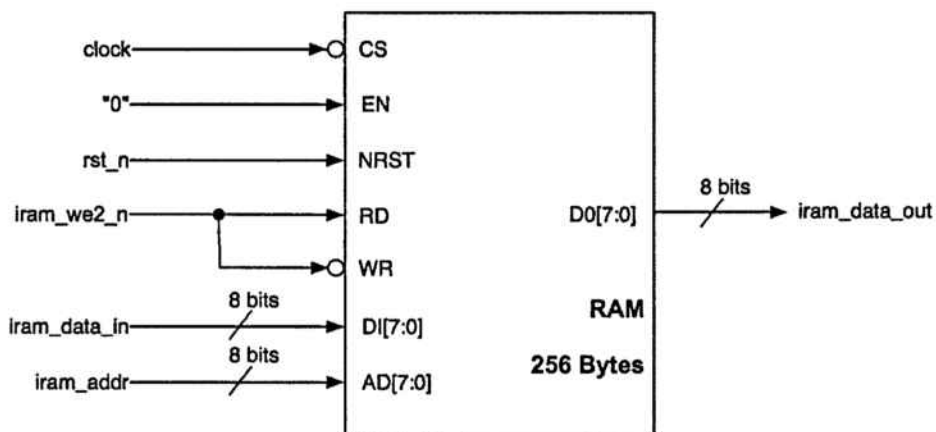


Figura 6.25: Diagrama de blocs de la nova interfície de la memòria RAM interna.

Una vegada substituïda la interfície vella per la nova es tornen a repetir els testos per les tres posicions de la variable *count*.

- Variable *count* a XDATA

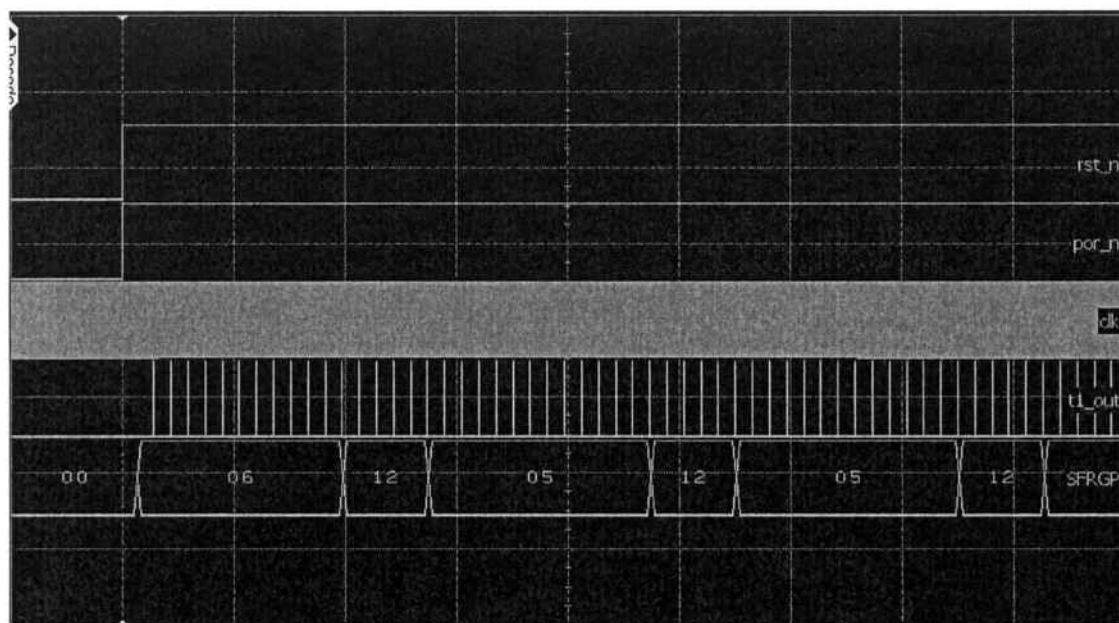


Figura 6.26: Captura d'oscil·loscopi (escala de temps 2ms/divisio).

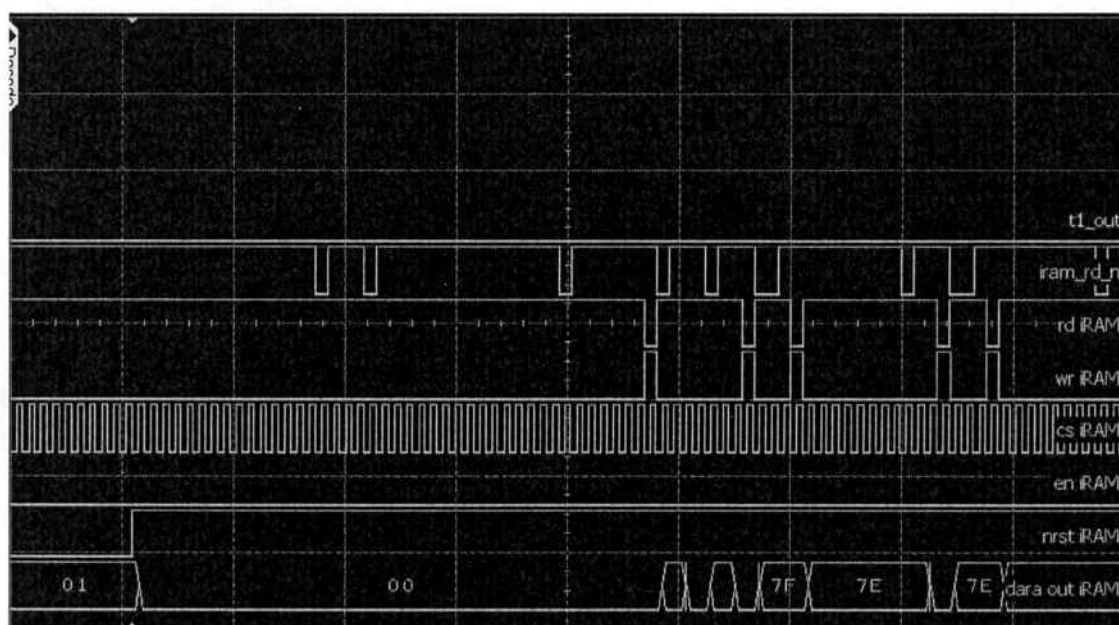


Figura 6.27: Captura d'oscil·loscopi (escala de temps 2ms/divisio).

Com es pot comprovar a la figura 6.26 el sistema executa el codi com s'esperava, ja que descriu la seqüència correctament i sense cap retard. El comportament és idèntic al descrit a la figura 6.16.

Pel que fa a les senyals de control de la memòria, figura 6.27, es pot observar com es realitzen correctament les lectures i escriptures, de manera que ara no es salta cap lectura a memòria. Si es compara la figura 6.27 amb la 6.21 que correspon a la variable *count* en XDATA però amb la interfície vella, es pot veure que quan la senyal *iram_rd_n* informa de la quarta lectura, a la interfície vella el bus *iram_data_out* (*data out RAM*, a les figures) es mate a zero, mentre que amb la interfície nova canvia de valor. Això demostra que efectivament l'anterior interfície no funcionava correctament i per això no s'obtenia un funcionament correcte del sistema.

Per la variable *count* a DATA i IDATA els resultats són igual de satisfactoris i verifiquen els resultats obtinguts per XDATA.

❖ T2: Test de la interfície de la memòria RAM externa

En la realització d'aquest test s'executarà el mateix codi que en el test T1, i es seguirà la mateixa metodologia. S'utilitzarà els resultats del punt 6.1.2 com a punt de control.

Els canvis realitzats en el disseny són:

- Substitució de la memòria RAM externa de 2 KB del *Quartus II* per la memòria RAM creada de 256 bytes. Tot i la diferència de mides de les memòries no suposa un problema ja que el codi és molt senzill i només posarem una variable en memòria.
- La memòria RAM interna és la utilitzada en el T1 amb la nova interfície de connexió.
- La memòria de programa continuarà sent la memòria síncrona del *Quartus II*.

En aquest cas la variable *count* només es posarà en XDATA, a la memòria RAM externa, ja que es vol avaluar el funcionament de la interfície de la memòria RAM externa.

Un cop fet el canvi, es carregar el programa a la memòria i s'executa amb la variable *count* a XDATA.

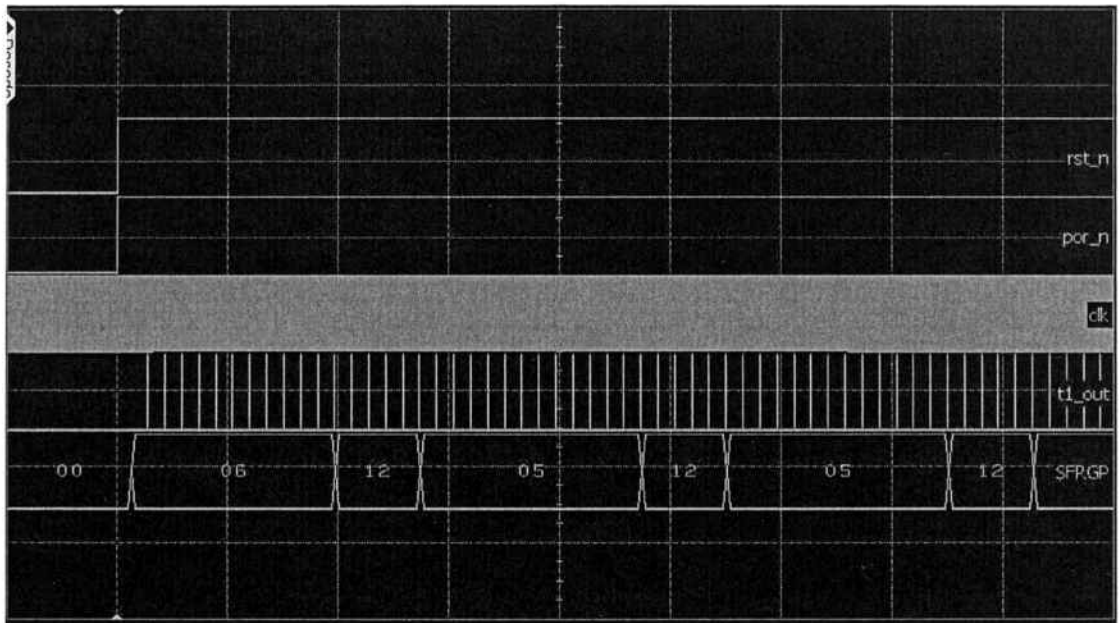


Figura 6.27: Captura de l'oscil·loscopi (escala de temps 2ms/divisio).

La figura 2.27 mostra els resultats obtinguts en el test. Com es pot observar, el codi s'executa aparentment de manera correcta ja que la seqüència obtinguda del registre SFRGP és l'esperada. A més en aquest cas no s'observa cap anormalitat com un retard en l'execució. Tot i això és necessari examinar les senyals de control de la memòria per verificar el seu correcte funcionament.

Abans d'examinar les senyals obtingudes en el test, convé examinar el codi on es troba instanciada la memòria RAM externa, per saber com està connectada. A la figura 6.28 es mostra el diagrama de blocs simplificat de la interfície de connexió de la memòria.

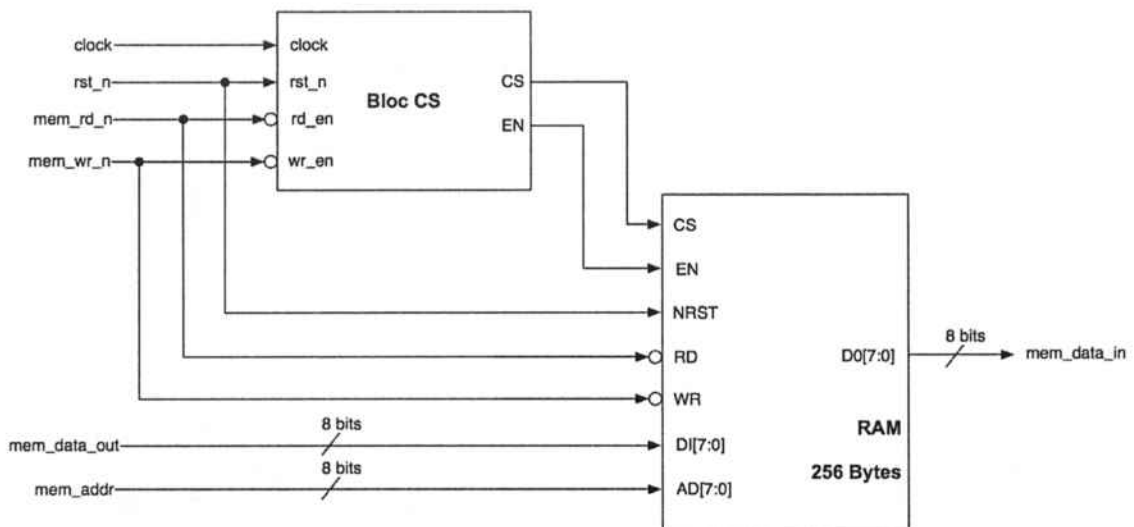


Figura 6.28: Diagrama de blocs de la interfície de la memòria RAM externa.

Un cop coneguda la interfície de la memòria RAM externa, es precedeix a examinar les senyals els resultats que es mostren a les figures 6.29 i 6.30.

A la figura 6.29 es pot veure clarament els moments en que s'accedeix a la memòria RAM externa. Examinant el diagrama de flux del programa carregat, figura 6.15, es pot veure que accedeix a la RAM externa per escriure o llegir la variable *count* en la inicialització del SFRGP i cada vegada que salta la interrupció senyal *t1_out*. Aquest comportament és pot identificar a la figura 6.29 on la primera vegada que s'accedeix a la memòria RAM externa no ha saltat la interrupció i la resta de vegades que si accedeix és immediatament després de que salti la interrupció.

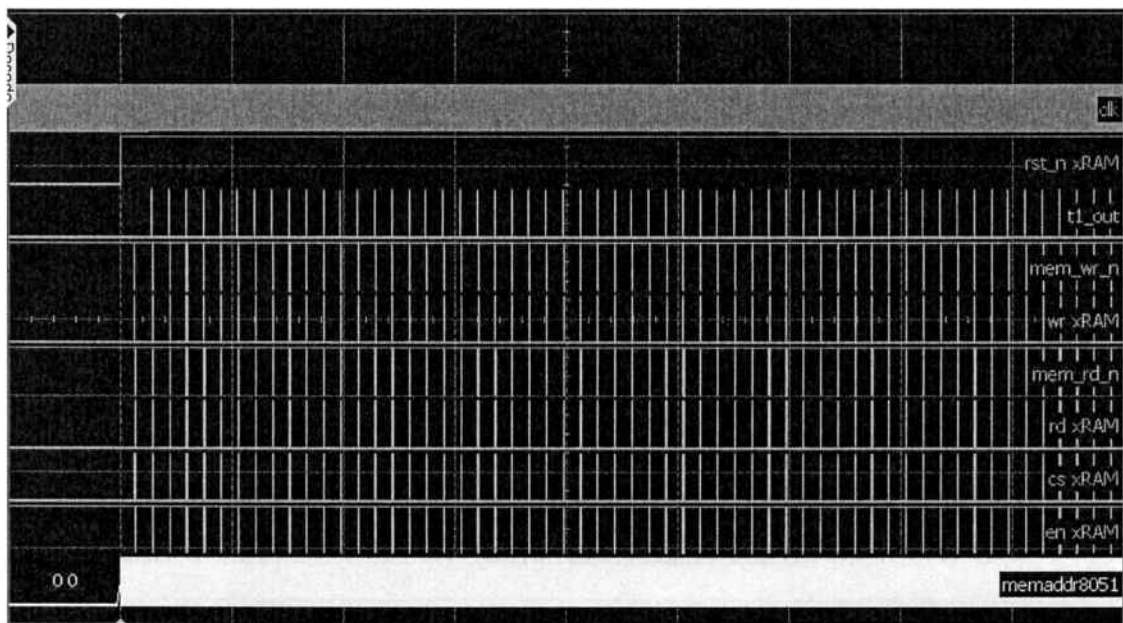


Figura 6.29: Captura de l'oscil·loscopi.

A la figura 6.30 es mostra la regió del primer accés a la memòria RAM externa ampliada, on s'observen una lectura seguida d'una escriptura i d'una lectura. Es pot veure com les senyals es generen correctament. En aquest cas no hi ha perill de que el CS no baixi per culpa de dues operacions consecutives ja que cada instrucció que accedeix a la memòria RAM externa, requereix 4 cicles d'instrucció on cada cicle d'instrucció són 4 cicles de rellotge.

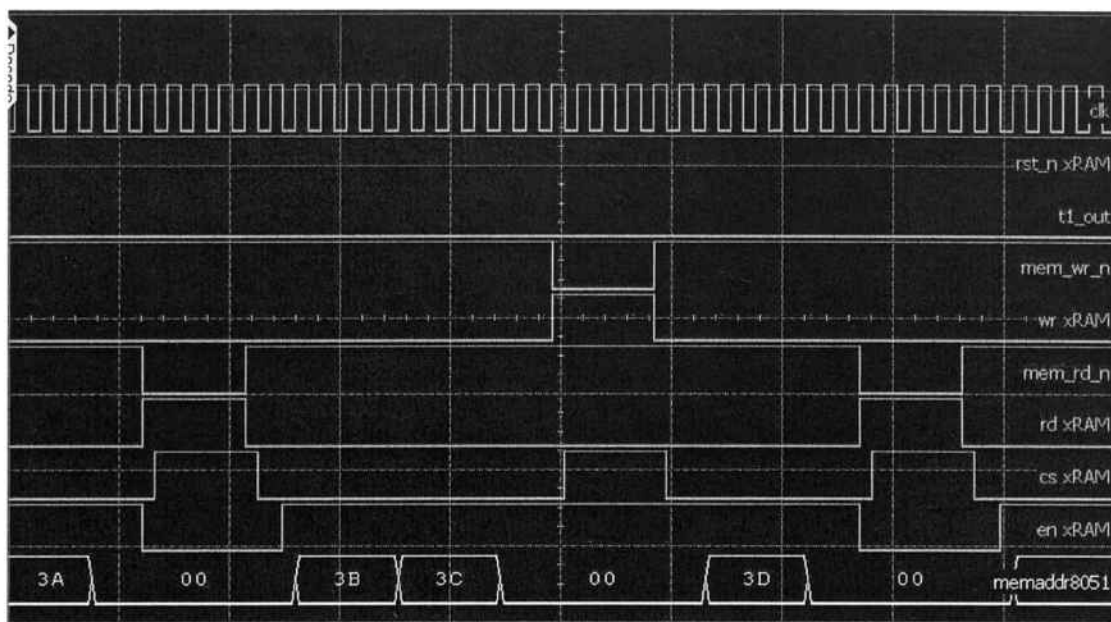


Figura 6.30: Captura de l'oscil·loscopi.

❖ T3: Test de la interfície de la memòria de programa

La interfície de la memòria de programa està dividida en dos. Per una banda les senyals de control procedents del processador i per l'altra les senyals de control procedents del *Boot Loader*. A la figura 6.31 es mostra un diagrama de blocs simplificat de la interfície de la memòria de programa.

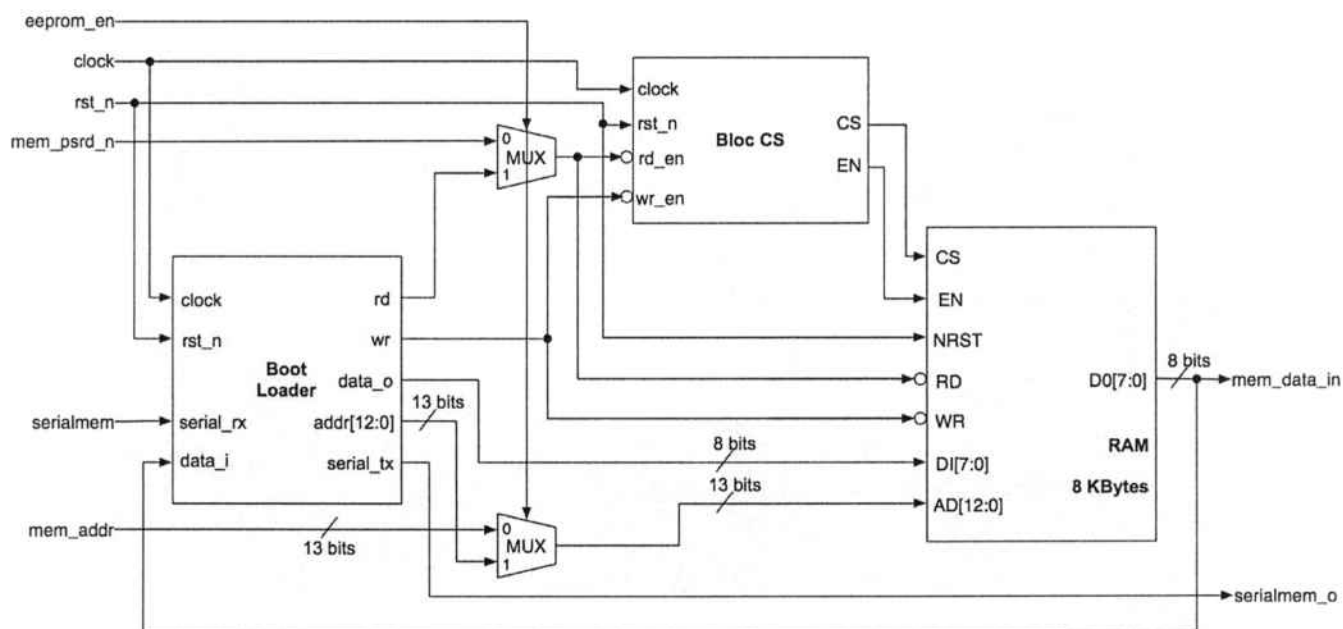


Figura 6.31: Diagrama de blocs de la interfície de la memòria de programa.

A la figura 6.31 es veu que depenent del valor de l'entrada *eeprom_en* canvia qui té el control de la memòria. Si és '0' el processador té el control, si és '1' és el *Boot Loader*. També es pot veure com el processador no pot escriure a la memòria de programa, mentre que el *Boot Loader* sí, ja que és la interfície que permet carregar el programa a la memòria des de l'exterior. El bloc CS és el mateix que el de la memòria RAM externa donem per suposat que genera correctament les senyals de control CS i EN.

Per tan, abans de canviar la memòria de programa síncrona del *Quartus II* per la memòria de 256 bytes creada per testejar, cal comprovar que el *Boot Loader* funciona correctament.

Per realitzar el test del *Boot Loader* ha estat necessari crear un bloc extern que es comuniqui amb aquest, *top_test_unit*, a l'annex 11.5 es troba la documentació del bloc creat. Un cop implementat el bloc creat, el top del sistema passa a ser el mostrat a la figura 6.32.

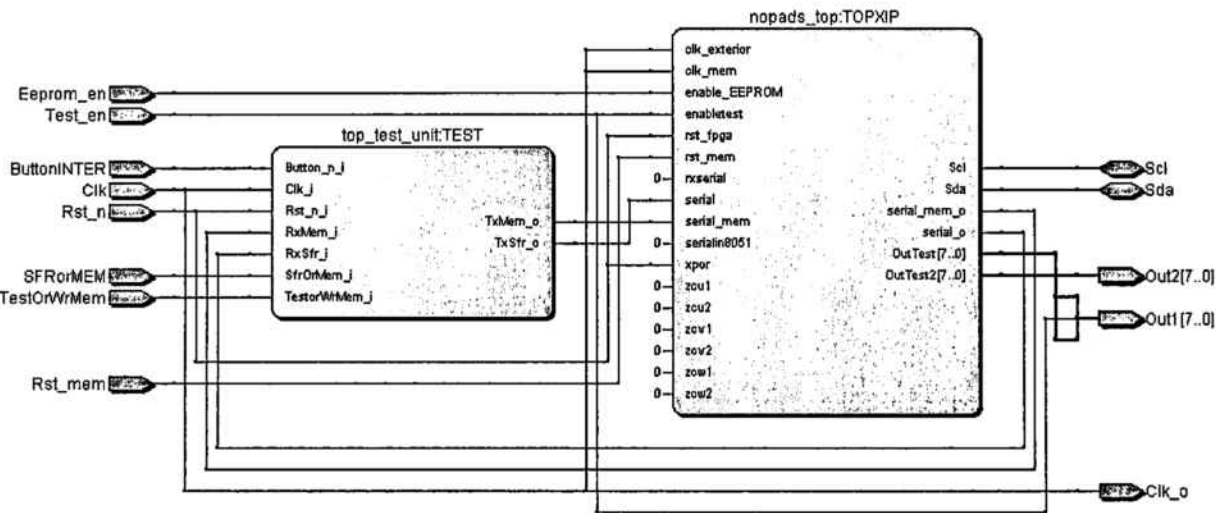


Figura 6.32: Top del sistema per el test del *Boot Loader*.

El *Boot Loader* es comunica amb l'exterior per mitja d'un port sèrie que consta d'un receptor i un transmissor. El protocol de comunicació es mostra a la figura 6.33.

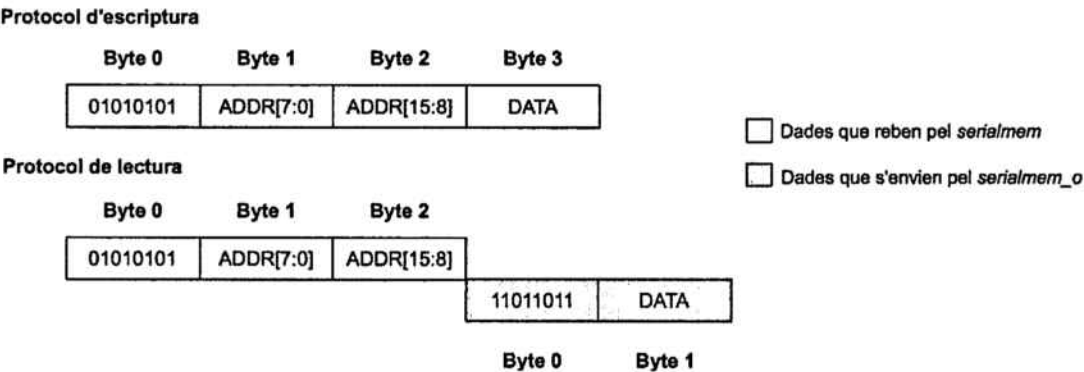


Figura 6.33: Diagrama temporal per escriure/llegir a/de una posició de la memòria de programa.

On cada byte s'ha d'enviar cap al *Boot Loader* començant pel bit menys significatiu i acabant pel bit més significatiu. De la mateixa manera, el *Boot Loader* envia les dades començant pel bit menys significatiu immediatament després d'haver rebut l'últim bit del byte 2 de l'ordre de lectura.

Conegut el protocol, ja es pot procedir a realitzar el test i analitzar-ne els resultats. El test consisteix en escriure i llegir una posició de memòria a través del *Boot Loader*.

- L'adreça a escriure i llegir és la EDh
- La dada a escriure és 11h.

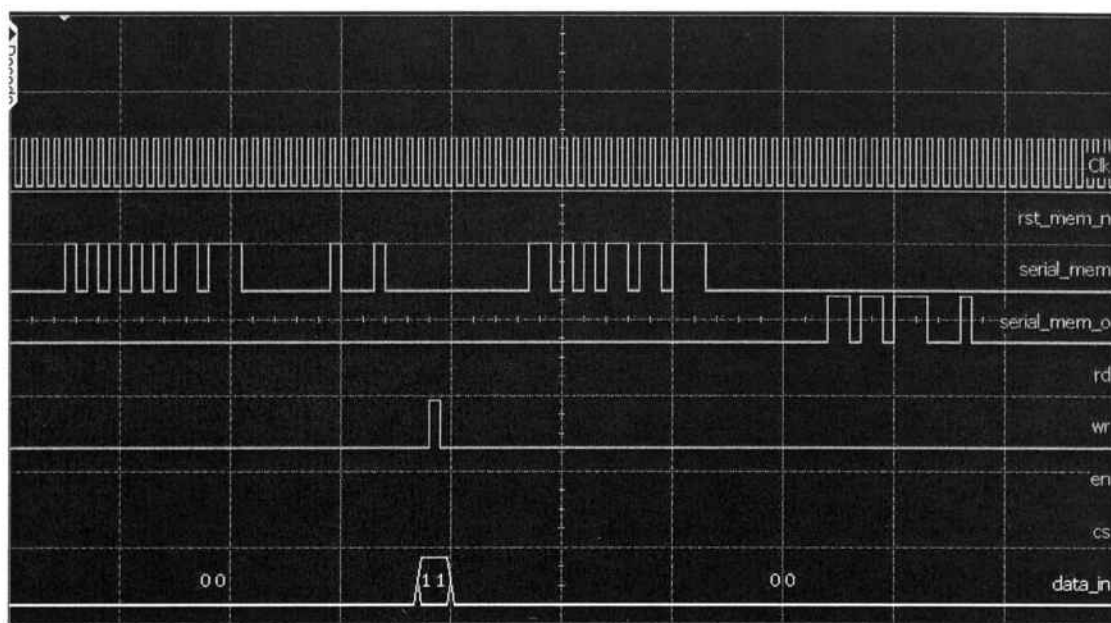


Figura 6.34: Captura de l'oscil·loscopi.

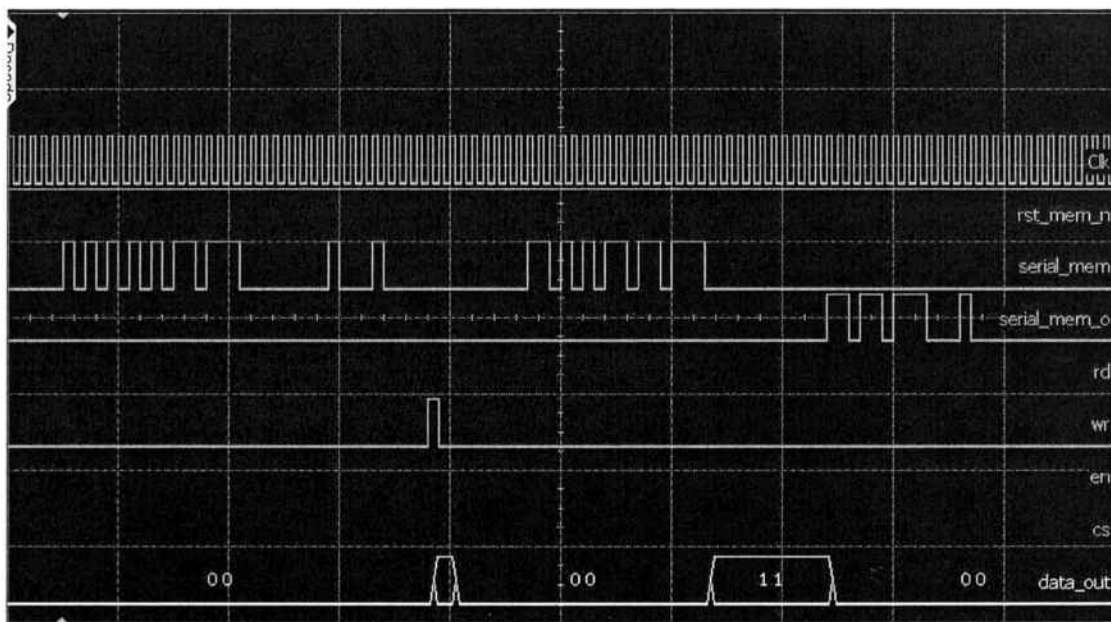


Figura 6.35: Captura de l'oscil·loscopi.

Els resultats obtinguts es mostren a les figures 6.34 i 6.35, la diferència entre les dues figures és que el la primera es veu el bus de entrada de dades a la memòria, senyal *data_in*, i en la segona el bus de sortida de dades de la memòria, *data-out*. Si s'analitzen les formes d'ona es pot observar que pel *serialmem* primer rep el següent,

10101010 10110111 00000000 10001000

Tenint present que primer s'envien els bit menys significatiu de cada byte, reordenant s'obté,

01010101 00000000 11101101 00010001 = 5500ED11h

Per tant s'ha complert el protocol d'escriptura. Es pot veure que acabada la transmissió de l'últim bit, s'activa el senyal d'escriptura a memòria, *wr*, i el valor el bus de dades entrants a la memòria *data_in* canvia de 00h a 11h, escrivint d'aquesta manera a la memòria de programa. Per comprovar que efectivament s'ha escrit a la memòria, un cop finalitzat el

procés d'escriptura, s'envia l'ordre de lectura de la mateixa posició de memòria, es veu com pel *serialmem* es rep,

11010101 10110111 00000000

Reordenant donà,

10101011 00000000 11101101 = AB00EDh

Això ens demostra que s'ha complert el protocol de lectura, i immediatament després de rebre l'últim bit, veure figura 6.35, el *Boot Loader* llegeix el valor de la posició indicada i l'envia pel port *serialmem_o*,

11011011 10001000

Reordenat els bits donà,

11011011 00010001 = DB11h

Repetint el test per diferents posicions de memòria i dades de diferents valors, s'ha de verificar que el *Boot Loader* funciona correctament. Cal notar que en la figura 6.34 el senyal *data_out* varia quan es realitza l'escriptura a memòria, això es degut al comportament de les memòries generades amb el *Quartus II*, que l'entrada de dades flueix directament cap a al sortida.

Un cop comprovat el correcte funcionament del *Boot Loader* amb la memòria del *Quartus II* com a memòria de programa, s'ha de repetir el test substituint aquesta per la memòria creada de 256 bytes.

En aquest cas la adreça a escriure/llegir és la 35h i la dada a escriure és 62h, els resultats d'aquest segon test es mostren a la figura 6.36 i 6.37.

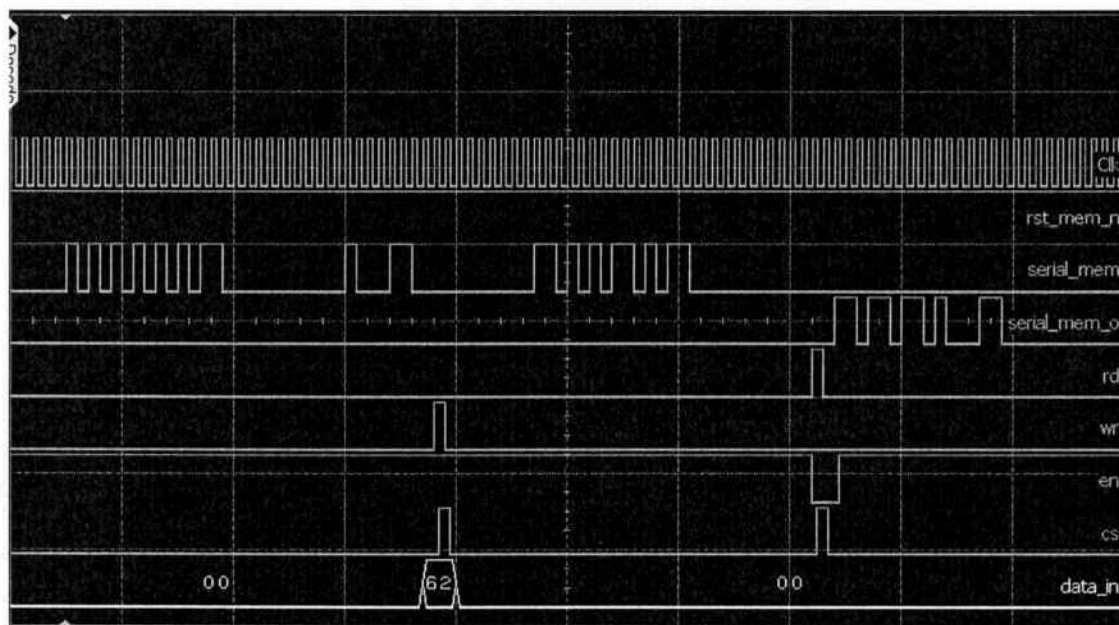


Figura 6.36: Captura de l'oscil·loscopi.

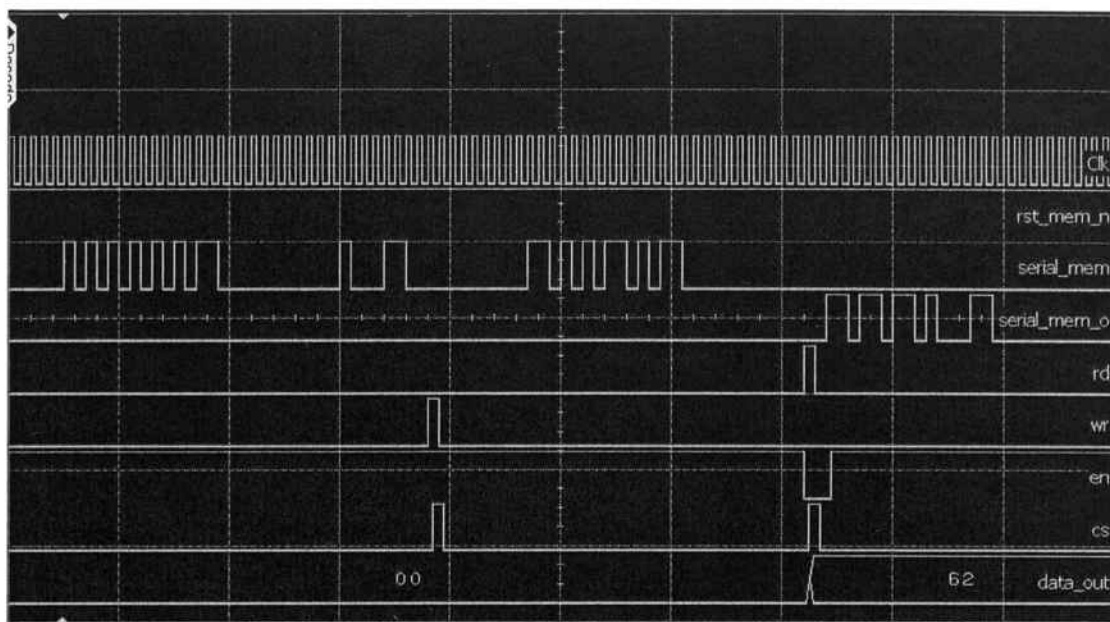


Figura 6.37: Captura de l'oscil·loscopi.

En aquest cas també s'obtenen uns resultats satisfactoris, ja que es pot comprovar com les senyals de control de la memòria es generen correctament i la memòria s'escriu i es llegeix a través del *Boot Loader*.

Per posar a prova tota la interfície de la memòria de programa, en el següent test es realitza un bolcat d'un programa complet a la memòria i es desperta el processador un cop finalitzat el procés d'escriptura, de manera que s'executa el programa que s'acaba d'escriure a la memòria.

El programa que s'escriu a la memòria és el mateix que en els testos anteriors, ja que ocupa menys de 256 bytes. Els resultats obtinguts es mostren en les figures 6.38, 6.39, 6.40, on les figures 6.39 i 6.40 són ampliacions de l'escriptura a la memòria i l'execució del programa respectivament.

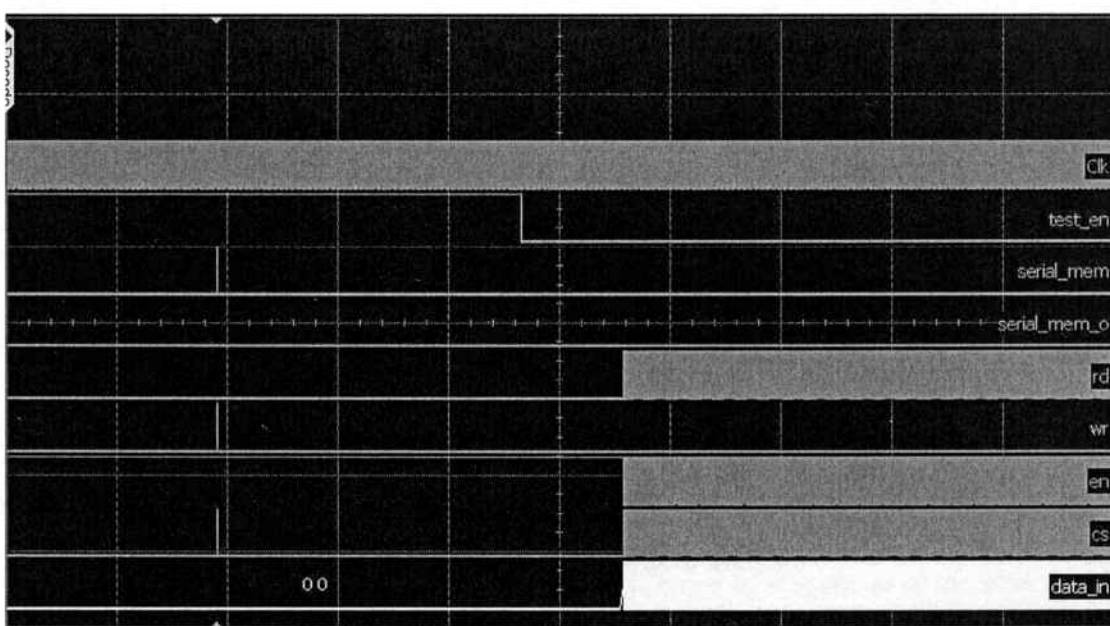


Figura 6.38: Captura de l'oscil·loscopi.

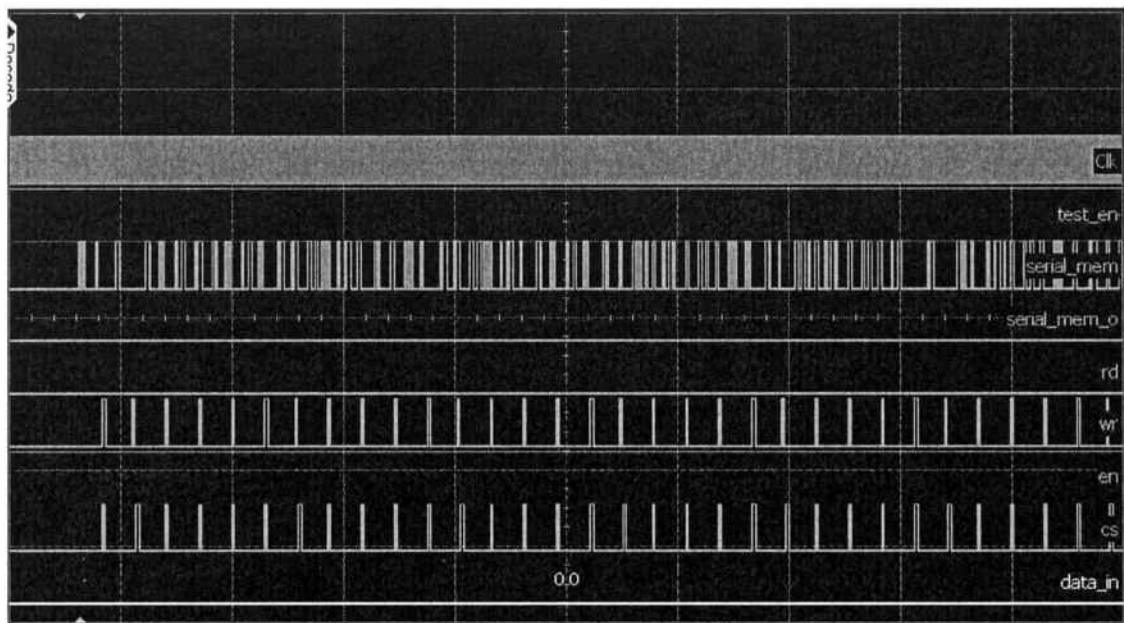


Figura 6.39: Captura de l'oscil·loscopi.

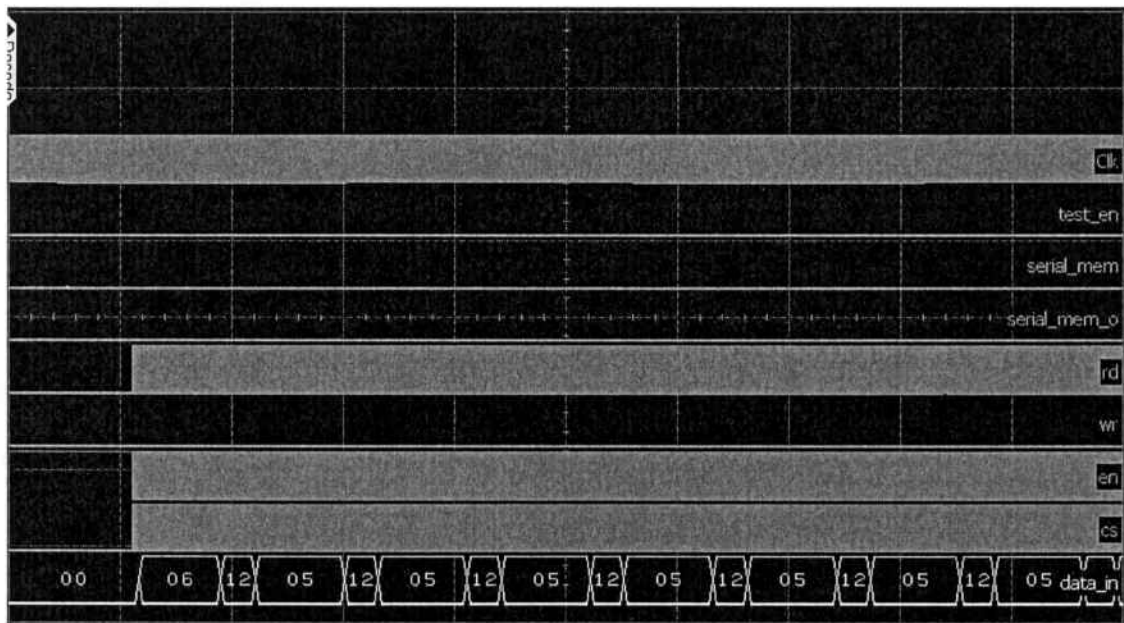


Figura 6.40: Captura de l'oscil·loscopi.

6.1.3.2 Test de la interfície dels registres SFRs

La interfície de test dels registres SFRs està format pels blocs *Test SFRs* i *Addr Decoder*, tal i com es mostra en el diagrama de blocs simplificat de la figura 6.41. D'aquest diagrama se n'extreu que depenent del valor de l'entrada *test_en*, el controlador del bus manager és el processador si és '0', i si és '1' el control el té el bloc *Test SFRs*.

Per realitzar el test del bloc *Test SFRs*, s'ha utilitzat el mateix bloc extern que en el test T3, per comunicar-se amb el bloc *Test SFRs*. Un cop implementat el bloc creat el top del sistema és el mateix que es mostrat a la figura 6.42.

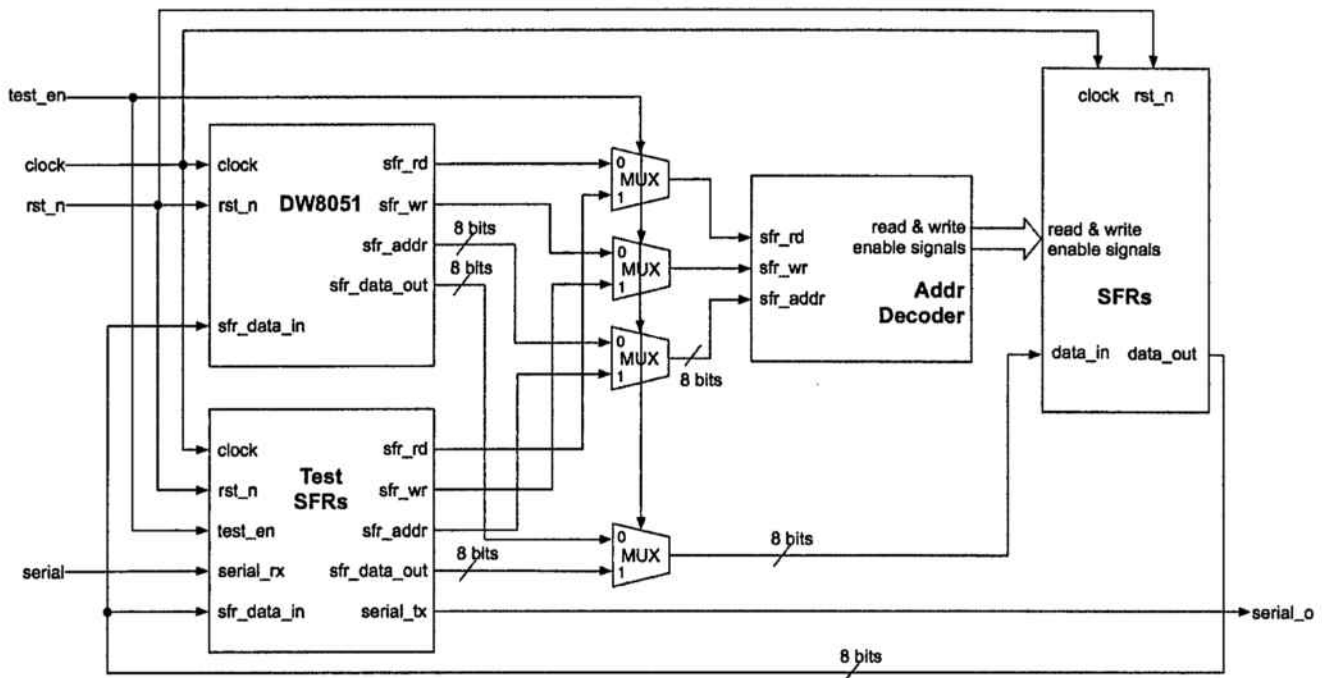


Figura 6.41: Diagrama de blocs de la interfície de test dels registres SFRs.

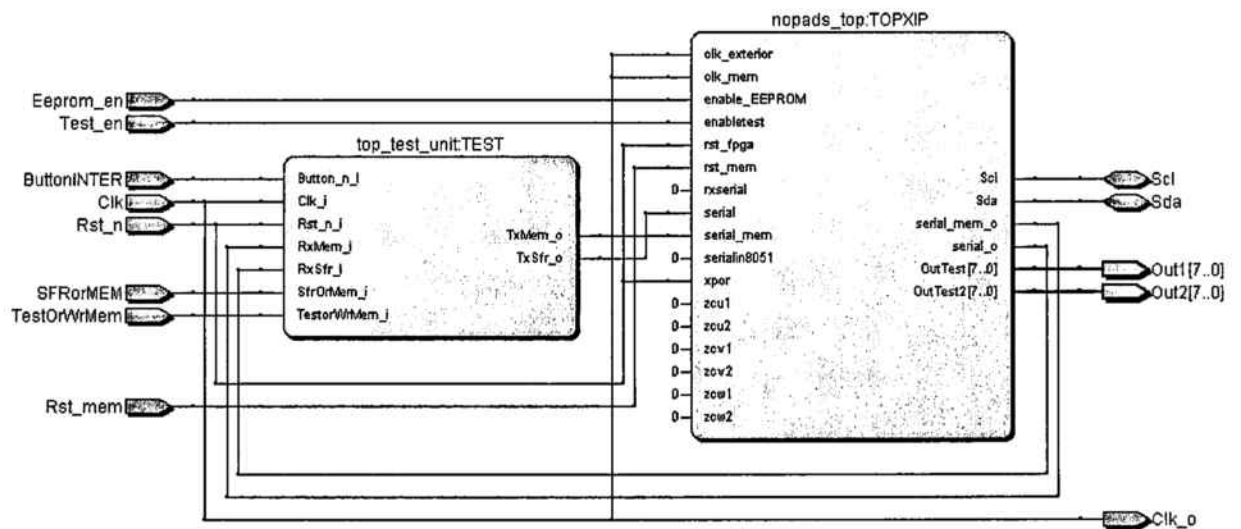
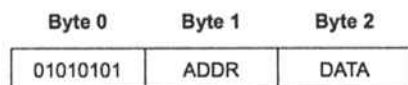


Figura 6.42: Top del sistema per el test del bloc *Test SFRs*.

El bloc *Test SFRs* es comunica amb l'exterior per mitja d'un port sèrie que consta d'un receptor i un transmissor. El protocol de comunicació és el que es mostra en la figura 6.43.

Protocol d'escriptura



☐ Dades que reben pel *serial*

☒ Dades que s'envien pel *serial_o*

Protocol de lectura

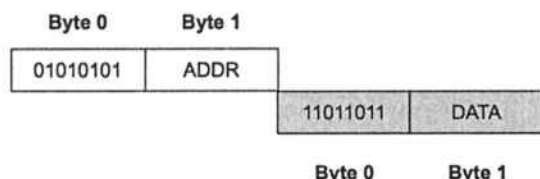


Figura 6.43: Diagrama temporal per escriure/llegir un SFR.

On cada byte que s'ha d'enviar cap al *Test SFRs* començant pel bit menys significatiu i acabant pel bit més significatiu. De la mateixa manera el bloc *Test SFRs* envia les dades començant pel bit menys significatiu immediatament després d'haver rebut l'últim bit del byte 1 de l'ordre de lectura.

Conegut el protocol, ja es pot procedir a realitzar el test i analitzar-ne els resultats. El test consisteix en escriure i llegir un SFR a través del bloc *Test SFRs*.

- L'adreça del registre SFR a escriure/llegir és la C2h
- La dada a escriure és 11h.

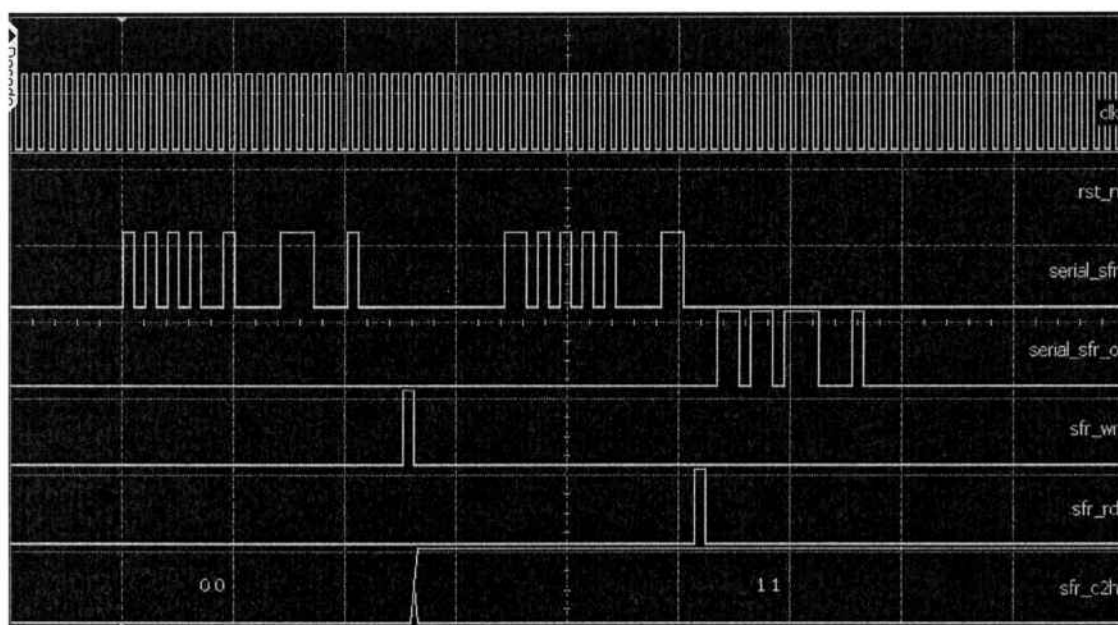


Figura 6.44: Captura de l'oscil·loscopi.

Els resultats obtinguts es mostren a la figura 6.44, analitzant les formes d'ona es pot observar que pel *serial* primer es rep el següent,

10101010 01000011 10001000

Tenint present que primer s'envien els bit menys significatiu de cada byte, reordenant s'obté,

01010101 11000010 00010001 = 55C211h

Per tant, s'ha complert el protocol d'escriptura. Es pot veure que acabada la transmissió de l'últim bit, s'activa la senyal d'escriptura al SFR, *sfr_wr*, i el valor del registre *sfr_c2h* passa de 00h a 11h. Per comprovar que efectivament s'ha escrit el *sfr_c2h*, un cop finalitzat el procés d'escriptura, s'envia l'ordre de lectura del mateix, es veu com pel *serial* es rep,

11010101 01000011

Reordenant donà,

01010101 11000010 = ABC2h

Això ens demostra que s'ha complert el protocol de lectura, i immediatament després de rebre l'últim bit, s'activa el senyal de lectura a SFR, *sfr_rd*, i es llegeix el valor del *sfr_c2h* i l'envia pel port *serial_o*,

11011011 10001000

Reordenat els bits donà,

11011011 00010001 = DB11h

Repetint el test pels diferents registres SFR del sistema s'ha verificar que la interfície de test dels registres SFRs funciona correctament.

6.1.3.3 Test Master I2C

Per testejar el Master I2C s'ha creat un bloc Slave I2C per poder realitzar una transmissió I2C amb el master i comprovar que aquest funciona correctament, a l'annex 11.6 es troba la documentació del bloc Slave I2C creat. Un cop implementat el bloc creat el top del sistema passa a ser el mostrat a la figura 6.45.

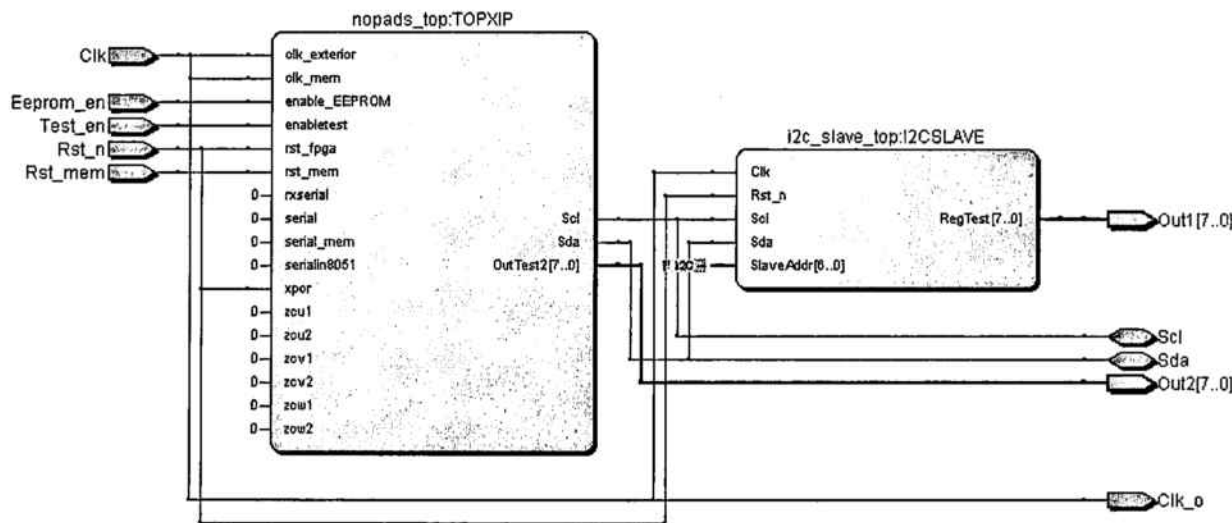
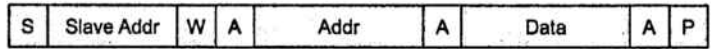


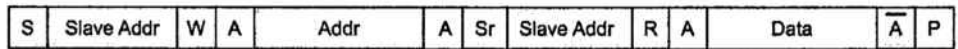
Figura 6.45: Top del sistema pel test del Master I2C.

El protocol de comunicació I2C que s'utilitzarà es mostra a la figura 6.46, es tracta del format amb 7 bits d'adreça del *slave*, i les dades enviades i rebudes són de 8 bits.

Protocol d'escriptura



Protocol de lectura



- ☐ Del slave al master
- ☐ Del master al slave
- S = Start condition
- P = Stop condition
- Sr = repeated Start condition
- A = ACK (SDA low)
- $\bar{A}$ = NACK (SDA high)
- W = Write (SDA low)
- R = Read (SDA high)

Figura 6.46: Protocol I2C per escriure i llegir al Slave I2C.

Els registres SFR que controlen i configuren el *Master I2C* es troben descrits a la taula 6.12.

Taula 6.12: Registres de control/configuració del *Master I2C*.

Nom	Addr	Accés	Descripció
PRERlo	B0h	RW	LSB del registre de prescalat del rellotge SCL.
PRERhi	B1h	RW	MSB del registre de prescalat del rellotge SCL.
CTRL	B2h	RW	Registre de control. Habilita el <i>Master I2C</i> .
TXR	B3h	RW	Registre de Transmissió. Conté el byte a transmetre.
CR	B4h	RW	Registre de comandes. Genera/habilita les comandes pròpies del protocol com són: START, STOP, ACK, NACK, Escriptura/lectura al <i>slave</i> .
RXR	B5h	R	Registre de recepció. Emmagatzema l'última dada rebuda.
SR	B6h	R	Registre d'estat. Informa de l'estat del bus, si s'ha rebut un ACK, estat de la transmissió, si s'hi ha pèrdua d'arbitratge.

Coneguts el protocol i els registres SFR que controlen el *Master I2C*, s'ha d'elaborar el codi que s'executarà en el DW8051 fent que el *Master I2C* realitzi de forma consecutiva,

1. Lectura del registre 9Eh del *Slave I2C*. Valor per defecte 00h.
2. Escripció del valor 33h al del registre 9Eh.
3. Lectura del registre 9Eh per verificar que s'ha escrit bé.

A l'annex 11.4 s'hi troba el codi font del programa.

A la captura mostrada a la figura 6.47, es poden veure les línies del bus I2C (SDA i SCL), amb la seva descodificació, i també es pot veure el valor que té en tot moment el registre 9Eh. En primera instància s'observa que l'execució del codi és correcta i el *Master I2C* es comunica satisfactòriament amb el *Slave I2C*.

En la primera lectura al registre d'adreça 9Eh, el *slave* respon al *master* enviant el valor del registre que és 00h, a la figura 6.48 es mostra una ampliació d'aquesta operació, on es poden apreciar millor el protocol.

Finalitzat el procés de lectura el *master* realitza la transmissió per escriure el valor 33h al registre 9Eh. Com es pot apreciar a la figura 6.47, quan s'acaba de rebre l'últim bit de la dada a escriure el valor del registre canvia a 33h, a la figura 6.49 es mostra l'operació d'escripció ampliada. Per últim, el *master* tornar a llegir el registre del *slave* i aquest li retorna el valor escrit anteriorment. Per tant, és pot concloure que el bloc *Master I2C* funciona correctament.

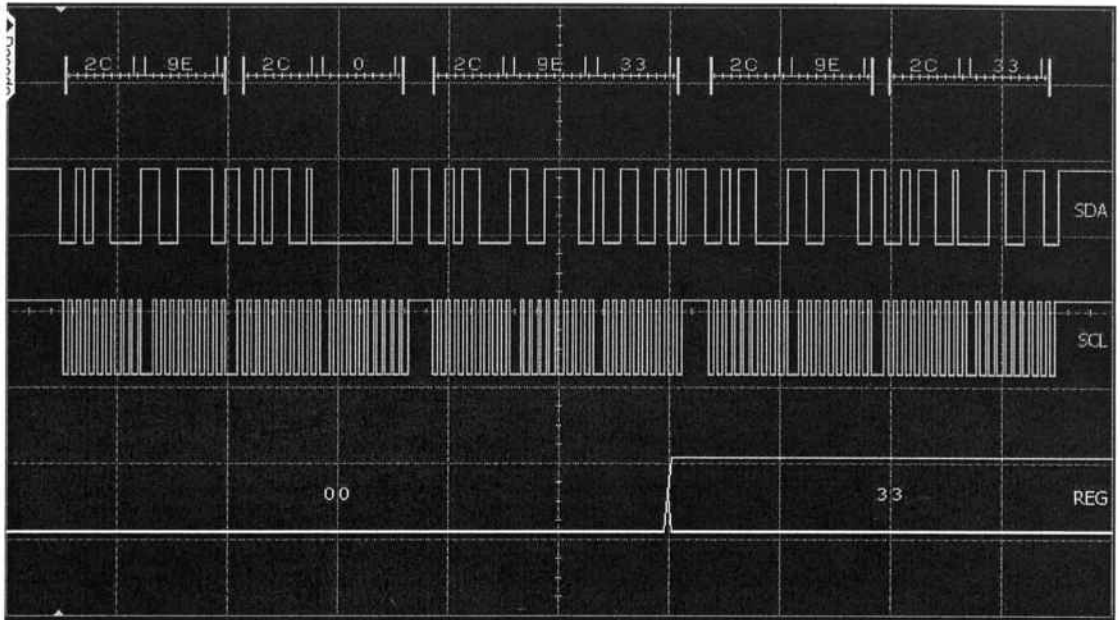


Figura 6.47: Captura de l'oscil·loscopi.

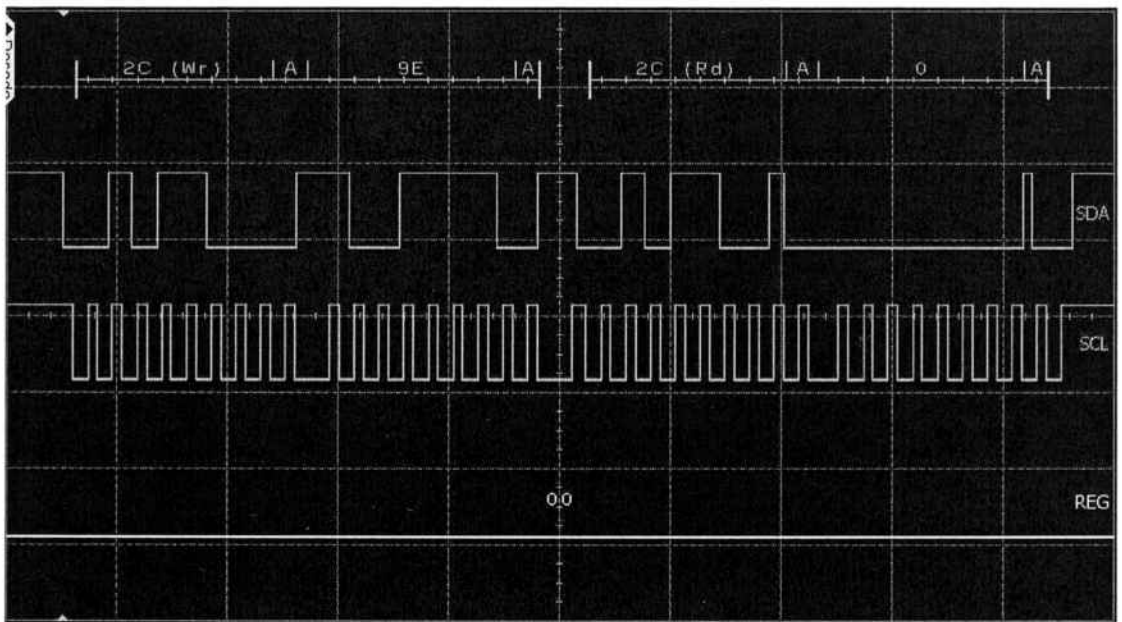


Figura 6.48: Captura de l'oscil·loscopi.

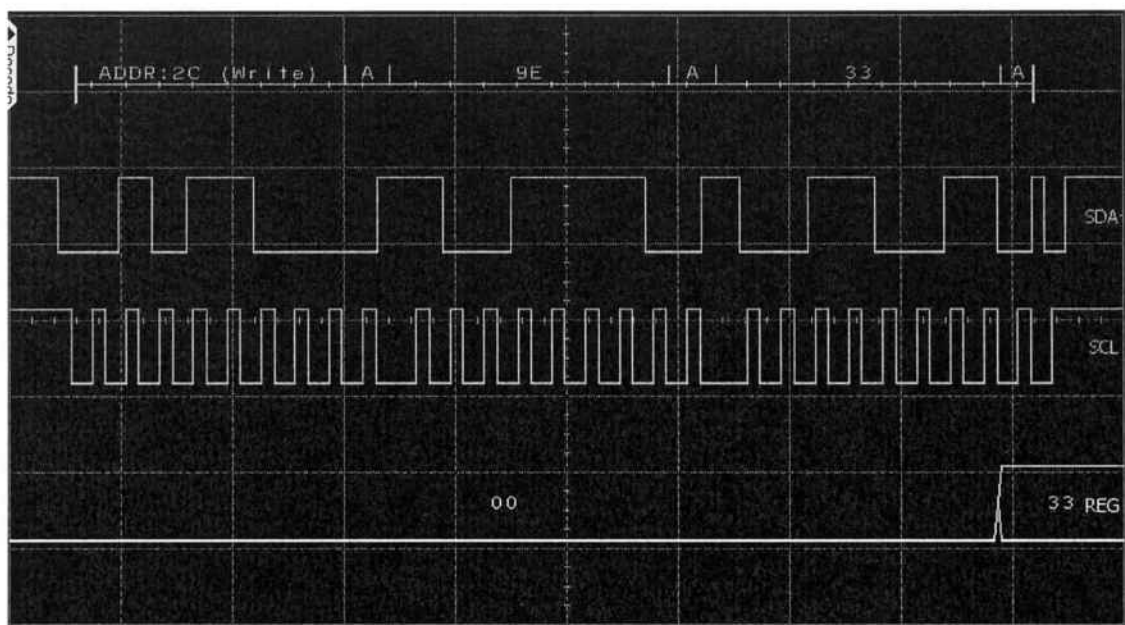


Figura 6.49: Captura de l'oscil·loscopi.

6.1.3.4 Test de la unitat de control de les comunicacions

A diferència de la resta de blocs del sistema, la unitat de control de les comunicacions (CCU, de l'anglès *Communications Control Unit*) funciona amb un rellotge de 1 MHz. Per tant, primer s'ha de comprovar que la unitat generadora de rellotges (CGU, de l'anglès *Clock Generator Unit*) genera correctament el rellotge de 1MHz.

El CGU genera el rellotge de 1 MHz mitjançant un prescalat del rellotge del sistema (10 MHz). El registre SFR que s'utilitza per controlar el bloc CGU el trobem definit a la taula 6.13.

Taula 6.13: Registres SFR del bloc CGU.

Nom	Addr	Accés	Descripció
CLK1M	AFh	RW	Bit [7] habilita la generació del rellotge de 1 MHz. Bits [6:0] s'utilitzen per especificar el valor del prescalat. $AFh[6:0] = \frac{clk_system}{2 \cdot 1MHz}$

Aplicant la formula s'obté que per generar un rellotge de 1 MHz s'ha d'escriure un 84h al registre SFR AFh. El resultats d'escriure aquest valor al registre es mostren a la figura 6.50.

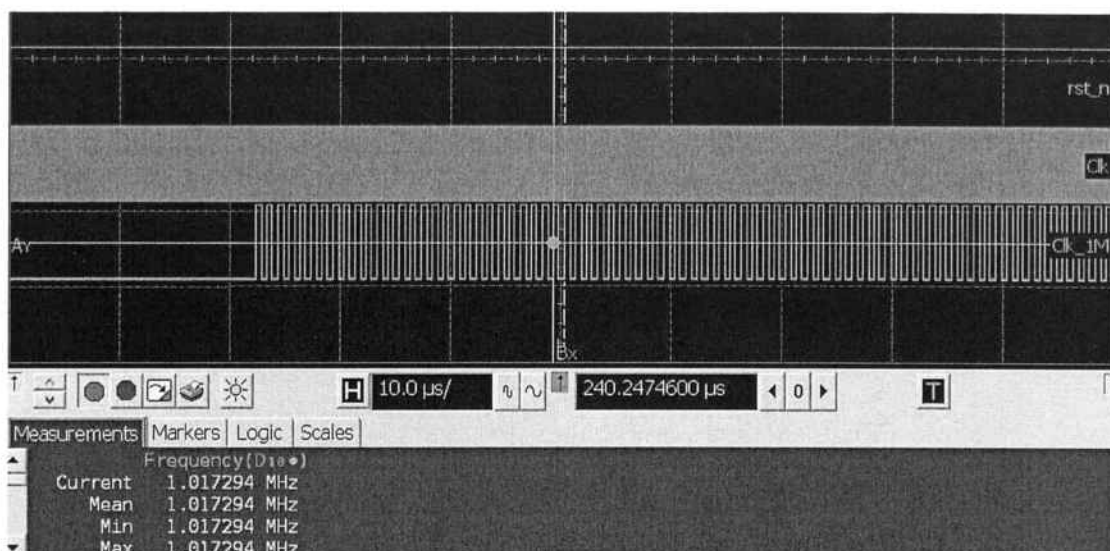


Figura 6.50: Captura de l'oscil·loscopi.

Un cop comprovat que es genera correctament el rellotge de 1 MHz, es pot procedir amb el test de la unitat de control de les comunicacions. Per realitzar el test del bloc CCU s'ha creat el bloc extern el *Comm RF*. Aquest bloc realitza la funció de transmissor de dades que rep el controlador de comunicacions a través del seu port sèrie *rxserial*, a l'annex 11.7 es troba la documentació del bloc *Comm RF* creat. Un cop implementat el bloc creat, el top del sistema passa a ser el mostrat a la figura 6.51.

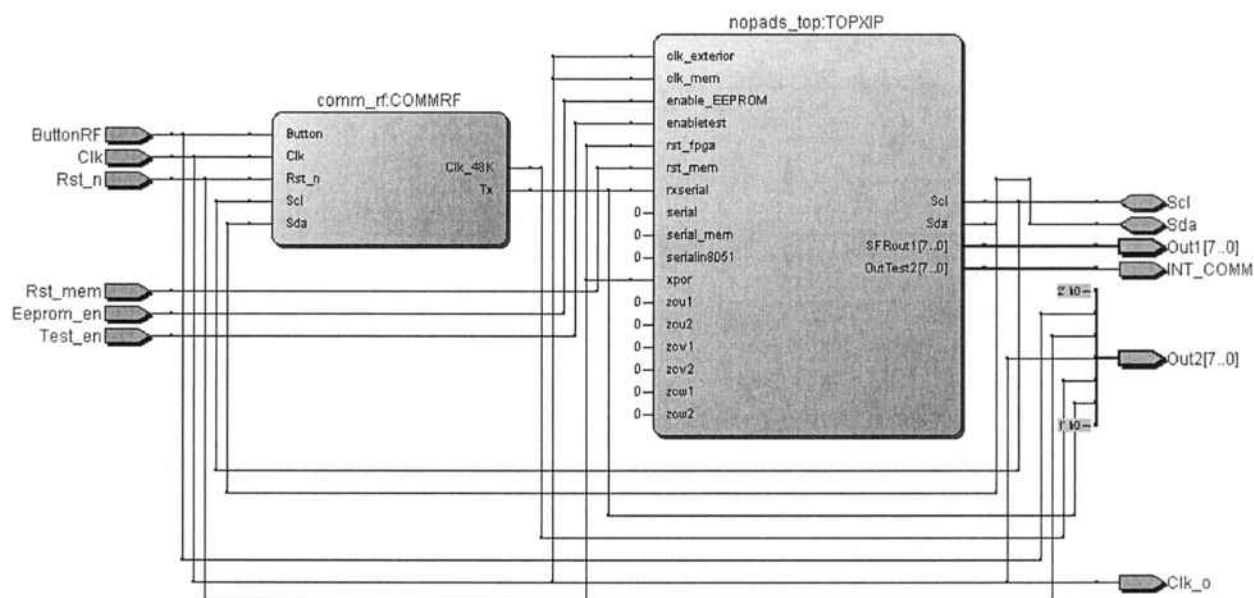


Figura 6.51: Top del sistema pel test del CCU.

El protocol de comunicació es mostra a la figura 6.52.



Figura 6.52: Protocol del bloc de comunicacions.

Com es veu a la figura 6.52, el canal Rx normalment es troba en repòs en un '0' lògic. Per començar el procés de recepció, el transmissor ha d'enviar un '1' lògic seguit dels 16 bits de dades començant pel bit menys significatiu. Un vegada rebudes les dades el CCU les guarda

i genera una interrupció al processador i torna a l'estat de repòs. El CCU rep les dades procedents del bloc *Comm RF* a 2.4 KHz i les mostreja amb una freqüència de 4.8 KHz.

Els registres SFR utilitza el CCU es troben descrits a la taula 6.14.

Taula 6.14: Registres SFR del bloc CCU.

Nom	Addr	Accés	Descripció
BITIME	BDh	RW	Temps de bit s'utilitza per mostreja la senyal d'entrada amb una freqüència de 4.8 KHz. Bit 0: ha de ser '1' per habilitat la recepció de dades. $\text{BITIME} = \frac{1\text{MHz}}{2 \cdot 4.8\text{KHz}}$
MDATA	BEh	R	MSB de la dada rebuda.
LDATA	BFh	R	LSB de la dada rebuda.

Coneguts el protocol i els registres SFR que controlen el bloc CCU, s'ha d'elaborar el codi que s'executarà en el DW8051. El test consisteix en enviar els 16 bits de dades a través del port *rxserial* a mode d'instrucció de manera que:

- El byte menys significatiu diu l'acció a realitzar. Si és 41h escriu al registre de propòsit general SFRGP el valor del byte més significatiu.

A la figura 6.53 es mostra el diagrama de flux del codi realitzat per aquest test, a l'annex 11.4 s'hi troba el codi font.

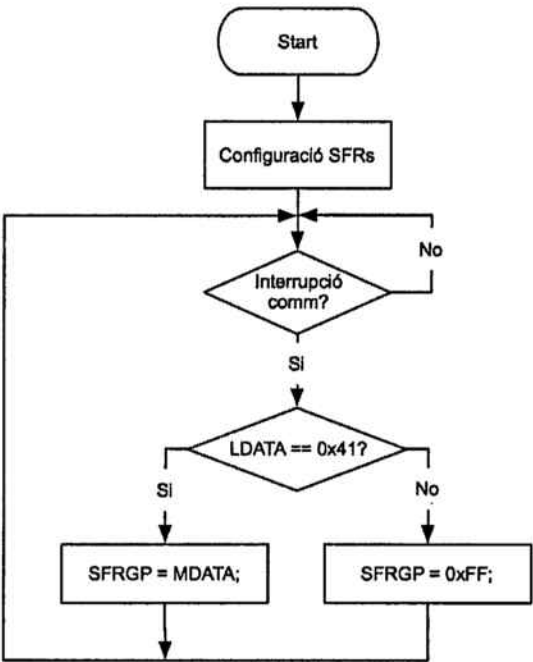


Figura 6.53: Diagrama de flux.

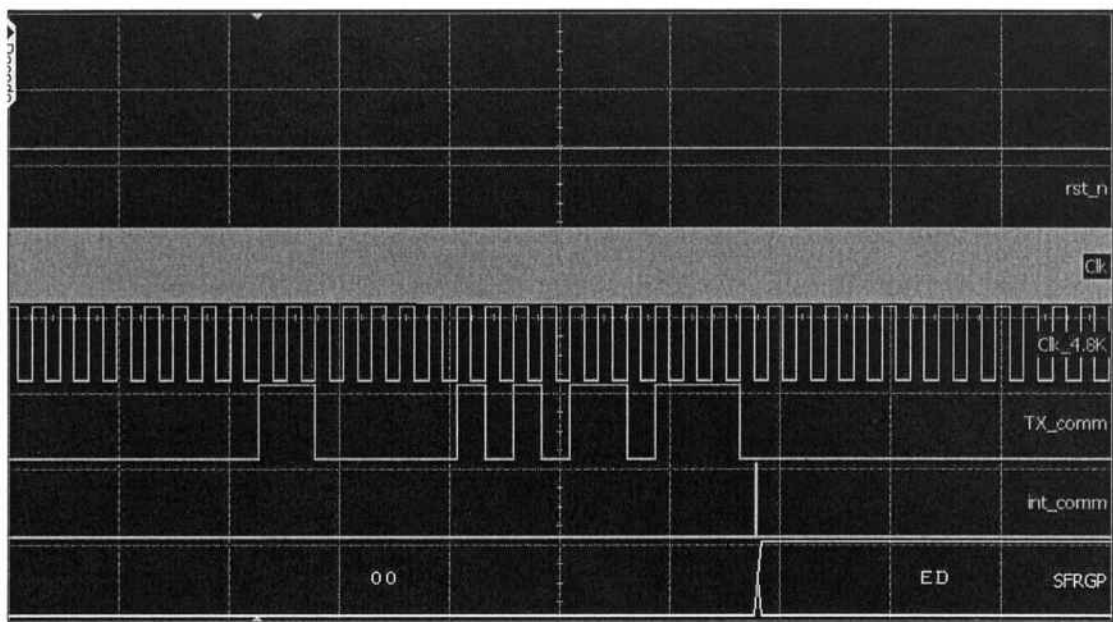


Figura 6.54: Captura de l'oscil·loscopi.

Els resultats obtinguts es mostren a la figura 6.54, analitzant les formes d'ona es pot observar que pel *Tx_comm*, que és correspon al *rxserial*, s'envia el següent,

1 10000010 10110111

Tenint present que primer s'envien els bit menys significatiu de cada byte, reordenant s'obté,

1 01000001 11101101

Ignorant el primer bit, que és per protocol, es pot decodificar el missatge i comprovar que s'ha seguit el protocol enviant primer el LSB (41h) després el MSB (EDh). Acabada la transmissió de l'últim bit, s'activa la senyal d'interrupció de les comunicacions *int_comm*, que fa saltar la rutina d'interrupció que escriu el valor del MSB rebut al registre SFRGP. El que ens demostra que el bloc CCU funciona correctament.

6.1.3.5 Test de la unitat de control del *Time Stamp*

La unitat de control del *Time Stamp* (TSCU, de l'anglès *Time Stamp Control Unit*) genera un senyal el rellotge de freqüència programable mitjançant un prescalat del rellotge del sistema de 10 MHz. Els registres SFR que s'utilitza per controlar la TSCU els trobem definit a la taula 6.15.

Taula 6.15: Registres SFR del TSCU.

Nom	Addr	Accés	Descripció
TSPER1	D9h	RW	LSB del registre de prescalat del senyal de <i>Time Stamp</i> .
TSPER2	DAh	RW	[15:8] bits del registre de prescalat del senyal <i>Time Stamp</i> .
TSPER3	DBh	RW	Bit [7] habilita la generació del senyal del <i>Time Stamp</i> . Bits [6:0] MSB del registre de prescalat del senyal de <i>Time Stamp</i> .
$\text{clk_timestamp} = \frac{\text{clk_system}}{2 \cdot (\{ \text{TSPER3}[6:0], \text{TSPER2}, \text{TSPER1} \} + 1)}$			

Per comprovar que funciona correctament, s'escriu als registres SFR el següents valors:

- TSPER1 = 08h

- TSPER2 = 00h
- TSPER3 = 80h

Aplicant la formula s'obté que es generarà un rellotge de 555.56 KHz. El senyal generat es mostra a la captura feta del oscil·loscopi de la figura 6.55, on es veu com efectivament es genera un senyal periòdic de la freqüència esperada.

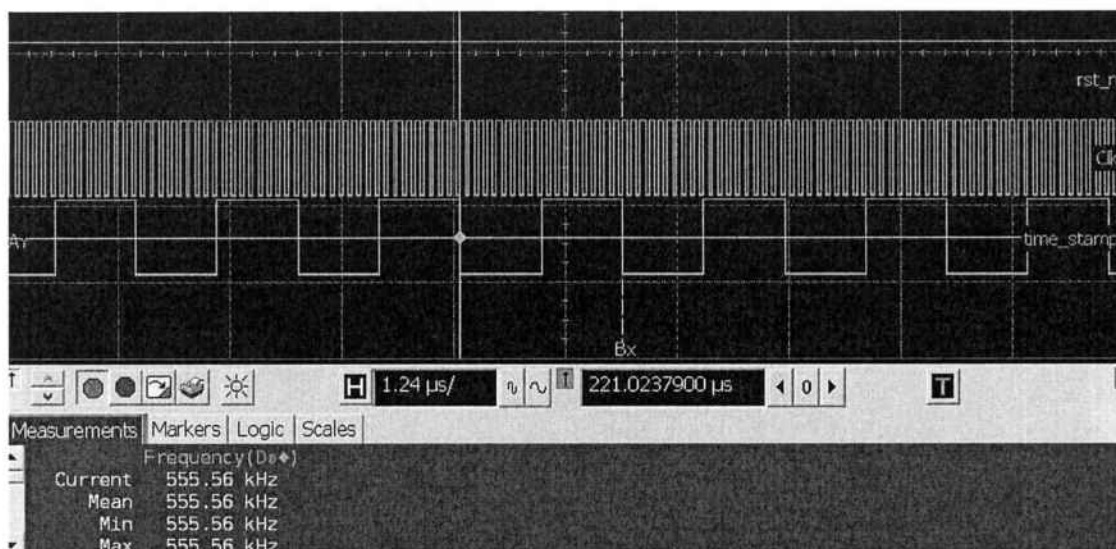


Figura 6.55: Captura de l'oscil·loscopi.

6.1.3.6 Test controlador de la locomoció (motors)

El control de la locomoció es pot realitzar de dues maneres diferents:

- **Mode 1:** El processador controla directament els *drivers* analògics, per mitjà dels registres SFR assignats per aquesta finalitat.
- **Mode 2:** El processador activa la unitat de control de la locomoció (LoCU, de l'anglès *Locomotion Control Unit*) que s'encarrega de generar les senyals de control automàticament.

Per a la realització del test de la LoCU s'ha utilitzat un ASIC previ al VECTOR V1. En aquest ASIC hi ha la part analògica dels controladors dels motors necessària per poder controlar el motor BLDC (de l'anglès *Brushless Direct Current*).

A la figura 6.56 es mostra el diagrama de blocs simplificat del muntatge realitzat per el test.

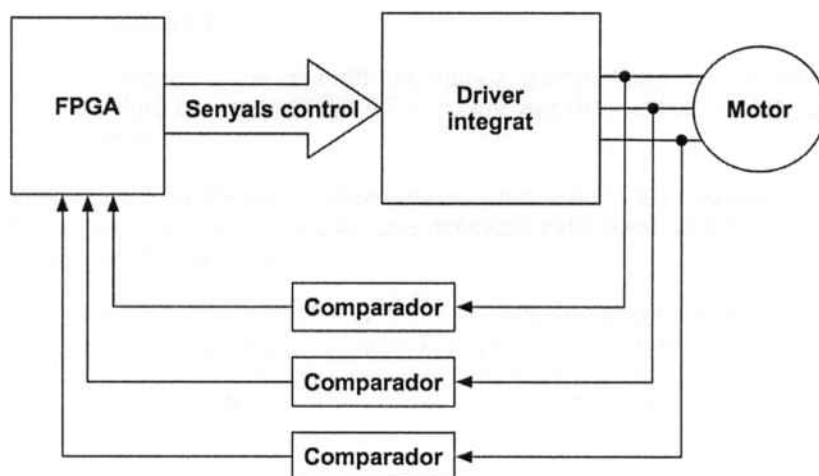


Fig 6.56: Diagrama de blocs del muntatge pel test del controlador de la locomoció.

A la figura 6.57 es mostren les senyals de posada en marxa del motor. Al principi els senyals no estan en fase però després d'uns períodes entren en fase i el motor comença a girar, verificant que el LoCU funciona correctament.

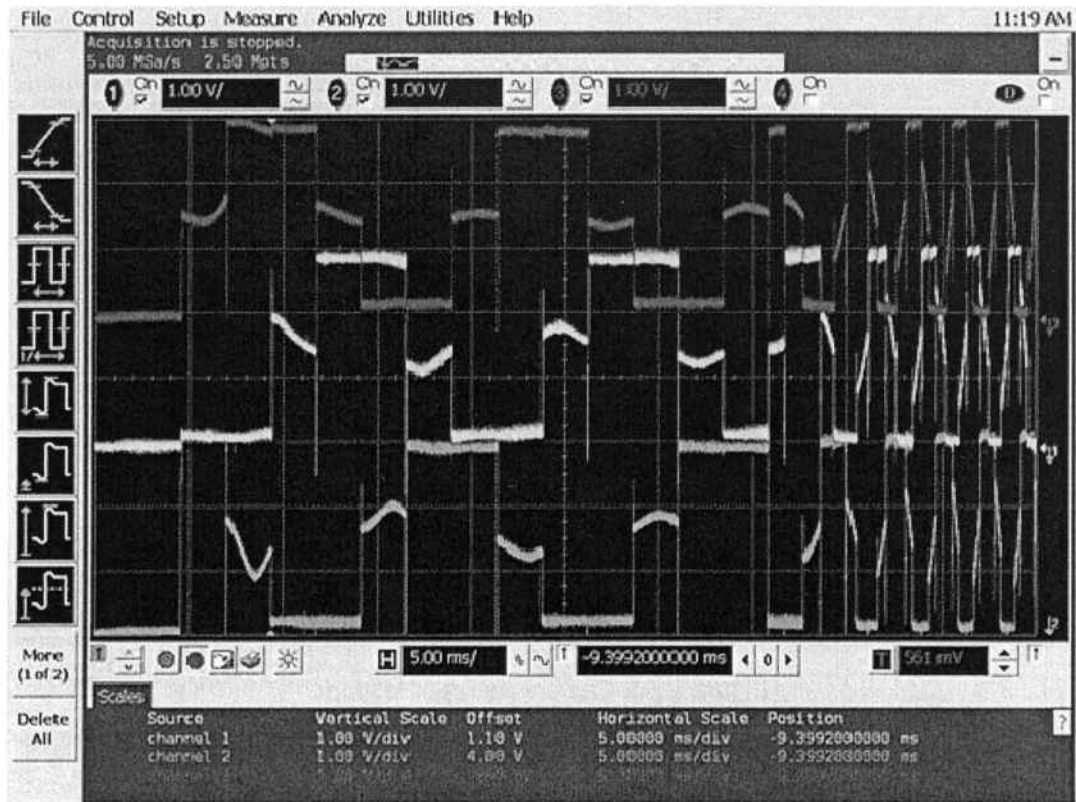


Figura 6.57: Captura de l'oscil·loscopi.

6.2 Disseny software de control

6.2.1 Requisits i característiques tècniques

A l'hora de dissenyar el software de control, s'han de tenir presents els següents requisits:

1. Control de les accions del sistema des de l'exterior.
2. Accés a tots els registres de configuració del sistema des de l'exterior.
3. Control de la locomoció, és a dir, els motors.
4. Control del sistema de visió. Basat en la càmera VECTOR2 de l'empresa *Neuricam* que és controla via I2C.
5. Control del sistema d'iluminació LED.

Totes aquestes tasques s'han d'executar en temps real. Per això el disseny del software parteix de l'ús del RTOS *RTX51 Tiny* de la casa *keil*. Per dissenyar correctament el software cal saber com funciona el *RTX51 Tiny* i les eines que ens proporciona per poder crear bé l'aplicació de control del xip VECTOR V1.

❖ Regles que s'han de seguir a l'hora de fer programes amb el *RTX51 Tiny*:

1. Assegurar-se d'incloure l'arxiu de capçalera *RTX51TNY.H*.
2. No crear una funció principal de C. El *RTX51 Tiny* té la seva pròpia funció principal.
3. El programa ha de contenir almenys una funció de tasca.
4. Els programes fets amb el *RTX51 Tiny* ha de tenir les interrupcions generals habilitades ($EA = 1$).
5. El programa ha de cridar almenys una rutina de biblioteca del *RTX51 Tiny* (com *os_wait*). En cas contrari, l'enllaç no inclourà la biblioteca *RTX51 Tiny*.
6. La tasca 0 és la primera funció que s'executa del programa que es creat. S'ha de cridar la funció *os_create_task* des de la tasca 0 per executar les altres tasques.
7. Les funcions tasca mai han d'acabar o retornar. La tasca s'ha de repetir amb un *while (1)* o similar. S'ha d'utilitzar la funció *os_delete_task* per aturar una tasca en execució.
8. Especificar *RTX51 Tiny* en el *µVision*.

❖ *RTX51 Tiny* i interrupcions

El *RTX51 Tiny* funciona en paral·lel amb les rutines d'interrupció. Les rutines d'interrupció poden comunicar-se amb les tasques del *RTX51 Tiny*, per l'enviament de senyals (mitjançant la funció *isr_send_signal*).

Les rutines d'interrupció ha de ser implementades i habilitades en l'aplicació creada amb el *RTX51 Tiny* tal i com es fa normalment en una aplicació estàndard. El *RTX51 Tiny* no realitza la gestió de les rutines d'interrupció.

RTX51 Tiny utilitza temporitzador 0 (*Timer 0*), la interrupció del temporitzador 0, i el banc de registres 1. El *RTX51 Tiny* genera una interrupció periòdica amb el *Timer 0*, que utilitza per calcular els temps d'espera i l'interval de valors especificats per a les rutines de la seva biblioteca. Si el programa utilitza el *Timer 0* el nucli del *RTX51 Tiny* deixarà de funcionar correctament.

RTX51 Tiny suposa que les interrupcions globals estan sempre habilitades (EA = 1). Les rutines de del *RTX51 Tiny* canvien l'estat del sistema d'interrupció (el bit EA) segons sigui necessari per garantir que les estructures internes *RTX51 Tiny* no estan danyats per les interrupcions. Quan habilita i deshabilita les interrupcions globals, el *RTX51 Tiny* simplement canvia l'estat de EA. No guardar i restaurar EA. Per tant, si el programa a crear desactivada les interrupcions abans de cridar a una rutina de *RTX51 Tiny*, aquest pot deixar de respondre.

❖ *RTX51 Tiny* i tasques

RTX51 Tiny és bàsicament un commutador de tasques. Per crear un programa d'*RTX51 Tiny*, s'ha de crear una aplicació amb una o més tasques/funcions de treball, amb un màxim de 16 tasques.

- Les tasques són simplement les funcions de C que tenen un tipus de retorn *nul* i sense llista d'arguments i es declaren amb l'atribut de la funció `_task_`. Per exemple:

```
void func (void) _task_ task_id
```

On *func* és el nom de la tasca i *task_id* és el numero de identificació (ID) de la tasca pot ser de 0 a 15.

- *RTX51 Tiny* manté cada tasca exactament en un estat (*Running*, *Ready*, *Waiting*, *Deleted*, or *Time-Out*).
- Només una tasca alhora pot estar en l'estat *Running*.
- En cas que totes les tasques definides estiguin bloquejades, el *RTX51 Tiny* executa un tasca buida *Idle_Task*.

El *RTX51 Tiny* pot ser configurat per utilitzar multitasca *Round-Robin*. *Round-Robin* permet executar quasi-paral·lelament diverses tasques. Les tasques no són realment executats al mateix temps, però són dividides en el temps (el temps de CPU disponible es divideix en segments de temps i *RTX51 Tiny* assigna una porció de temps a cada tasca). Atès que l'interval de temps és curt (només uns pocs mil·lisegons) sembla com si les tasques s'executin de forma simultània.

Les tasques s'executen duran el temps assignat. Llavors, el *RTX51 Tiny* canvia a la següent tasca que estigui llesta per a funcionar. O en lloc d'esperar que el temps una tasca estigui a punt d'expirar, es pot utilitzar la funció `os_wait` per permetre al *RTX51 Tiny* canviar a una altra tasca. La funció `os_wait` suspèn la tasca actual (la canvia a l'estat *Waiting*) fins que es produeix l'esdeveniment especificat (i la tasca es canvia a la l'estat *Ready*). Durant el temps en que la tasca està en l'estat *Waiting*, qualsevol nombre d'altres tasques poden anar-se executant.

❖ Funcions pròpies del *RTX51 Tiny*

El *RTX51 Tiny* disposa de varies funcions per facilitar el control de l'execució de les tasques. Aquest control és realitza mitjançant el canvi d'estat de cada tasca. Les funcions que s'utilitzaran en el disseny del software són les següents.

- **`os_create_task`**

```
#include <rtx51tiny.h>

char os_create_task (
    unsigned char task_id);    /* ID of task to start */
```

La funció `os_create_task` inicia la funció amb nombre de tasca *task_id*. La tasca es posa en estat *Ready* i comença la seva execució en la primera oportunitat.

- **`os_delete_task`**

```
#include <rtx51tny.h>

char os_delete_task (
    unsigned char task_id);    /* task to delete */
```

La funció *os_delete_task* atura la tasca especificada per *task_id* i la elimina de la llista de tasques.

- **os_wait**

```
#include <rtx51tny.h>

char os_wait (
    unsigned char event_sel,    /* events to wait for */
    unsigned char ticks,       /* timer ticks to wait */
    unsigned int dummy);       /* unused argument */
```

La funció *os_wait* atura la tasca actual i la posa en espera d'un o diversos esdeveniments, com ara un interval de temps, un temps d'espera, o un senyal d'una altra tasca o interrupció. L'argument *event_sel* especifica l'esdeveniment o esdeveniments a esperar i pot ser qualsevol combinació de les següents:

Taula 6.16: Valors de l'argument *event_sel*.

Esdeveniment	Descripció
K_IVL	Espera l'interval especificats per l'argument <i>ticks</i> .
K_SIG	Espera un a senyal.
K_TMO	Espera el temps d'espera especificat per l'argument <i>ticks</i> .

Els esdeveniments poden combinar lògicament usant el caràcter de barra vertical ('|'). Per exemple, *K_TMO | K_SIG*, especifica que la tasca esta a l'espera espera de un senyal o a la finalització de un temps d'espera. Quan un dels esdeveniments especificats es produeix, la tasca es posa en l'estat *Ready*.

- **isr_send_signal**

```
#include <rtx51tny.h>

char isr_send_signal (
    unsigned char task_id);    /* Task to signal */
```

La funció *isr_send_signal* envia un senyal a la tasca amb nombre de tasca *task_id*. Si la tasca especificada ja està esperant una senyal, aquesta funció es prepara per a l'execució de la tasca, però no l'inicia. En cas contrari, el senyal s'emmagatzema en la bandera de la senyal de la tasca. S'utilitza quan es vol enviar un senyal a una tasca des de dins una rutina d'interrupció.

- **os_send_signal**

```
#include <rtx51tny.h>

char os_send_signal (
    char task_id);    /* Task to signal */
```

Funciona exactament com la funció *isr_send_signal*, però en aquest cas s'utilitza quan es vol enviar un senyal a una tasca des d'una altra tasca.

6.2.2 Arquitectura del software de control

Tenint presents els punts anteriors es creen les tasques per controlar el xip VECTOR V1. A la figura 6.58 es mostra como estan estructurades diferents tasques que formen l'aplicació creada.

- La tasca en vermell només s'executen una vegada.
- La tasca en groc és la tasca principal, que crida/executa la resta de tasques segons les instruccions que es reben a través del port de comunicacions. Roman a l'espera de la interrupció de les comunicacions.
- Les tasques en blau, són les encarregades de modificar la configuració de cada bloc segons l'ordre rebuda del exterior. Romanen inactives fins rebre la ordre de la tasca *system*.
- La ISR (de l'anglès *Interrupt Service Routine*) en carbassa, conté el codi que s'executa quan salta les interrupcions del temporitzador 1.

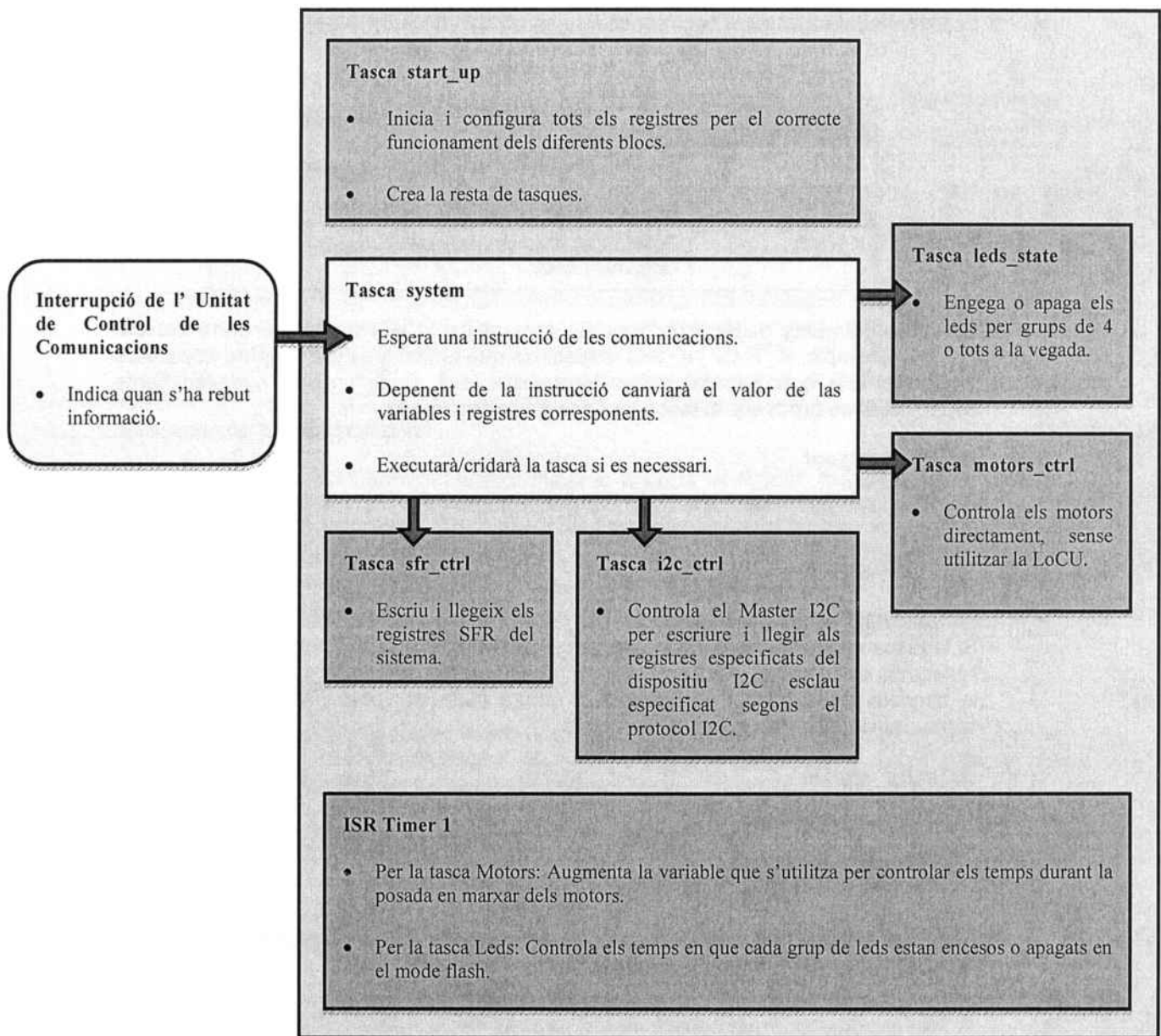


Figura 6.58: Esquema de l'estructuració de les tasques del RTOS.

6.2.3 Tasques del software de control

❖ Tasca start_up

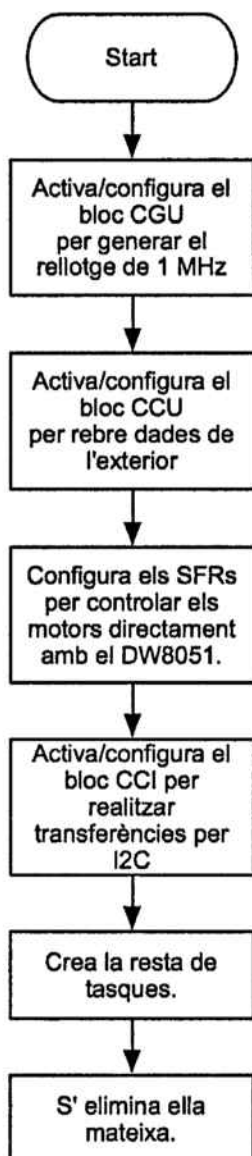


Figura 6.59: Diagrama de flux de la tasca *start_up*.

❖ **Tasca system**

La *tasca system* és el cor de tot el sistema, és on estan descrites totes les operacions que s'han de fer segons la instrucció que arriba per el port de comunicacions.

El sistema està pensat per un màxim de 256 instruccions. Cada instrucció consta de com a mínim 2 bytes i té un màxim de 2N bytes. Això es degut a que el bloc de comunicacions rep cada vegada 2 bytes d'informació. Actualment en té 25 instruccions descrites a la taula 6.17.

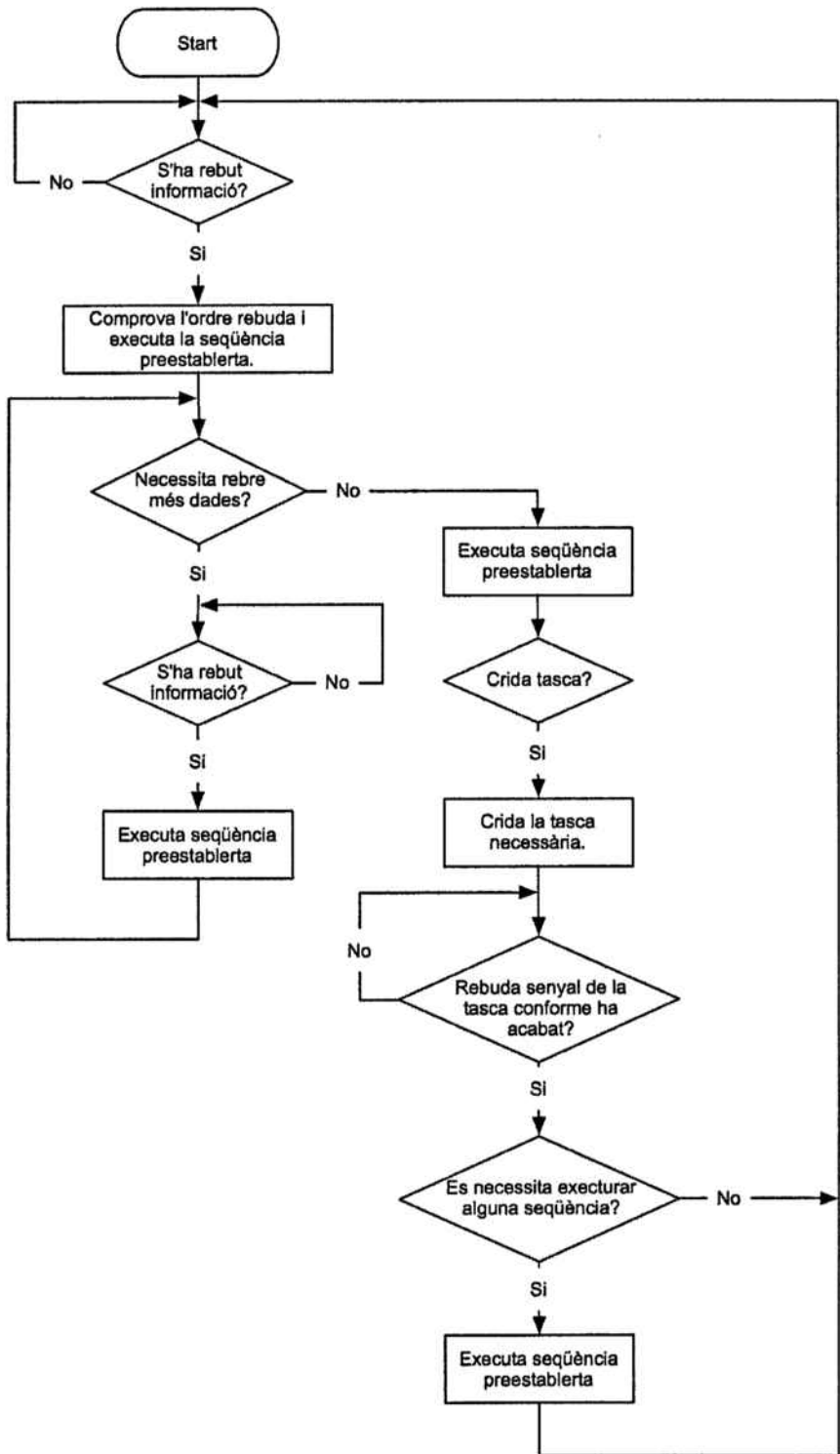


Figura 6.60: Diagrama de flux de la tasca system.

❖ **Tasca *sfr_ctrl***

La tasca *sfr_ctrl* permet controlar tots els blocs que es configuren per mitja dels registres SFR ja que donà accés directe als registres SFR. De manera que es pot reconfigurar qualsevol bloc del sistema en tot moment.

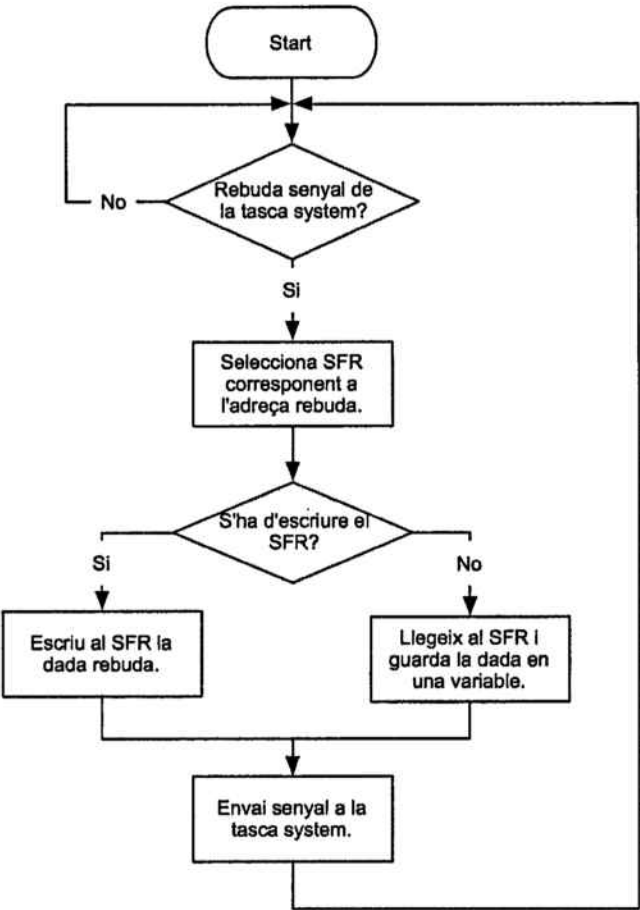


Figura 6.61: Diagrama de flux de la tasca *sfr_ctrl*.

❖ Tasca i2c_ctrl

La tasca *i2c_ctrl* controla el *master* I2C per realitzar les transferències d'informació. La figura 6.62 mostra els protocols de comunicació implementats.

Protocol d'escriptura

S	Slave Addr	W	A	Addr	A	Data	A	P
---	------------	---	---	------	---	------	---	---

Protocol de lectura amb format combinat

S	Slave Addr	W	A	Addr	A	Sr	Slave Addr	R	A	Data	$\overline{A}$	P
---	------------	---	---	------	---	----	------------	---	---	------	----------------	---

Protocol de lectura Standard

S	Slave Addr	R	A	Addr	A	Data	$\overline{A}$	P
---	------------	---	---	------	---	------	----------------	---

- ☐ Del slave al master S = Start condition Sr = repeated Start condition $\overline{A}$ = NACK (SDA high) R = Read (SDA high)
☐ Del master al slave P = Stop condition A = ACK (SDA low) W = Write (SDA low)

Figura 6.62: Protocols I2C.

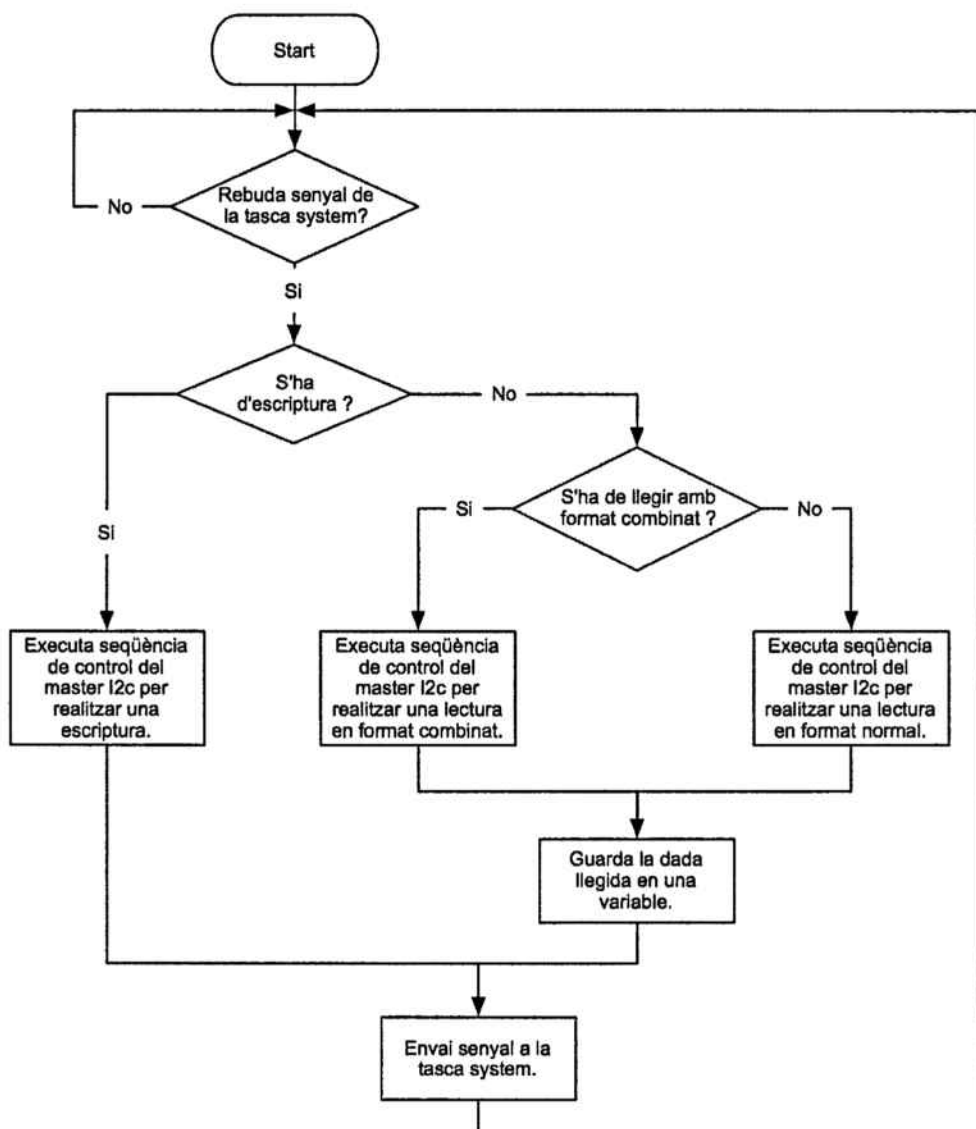


Figura 6.63: Diagrama de flux de la tasca *i2c_ctrl*.

❖ Tasca *motors_ctrl*

La tasca *motors_ctrl* realitzar el control dels motors actuant directament sobre el *driver* analògic. S'utilitza el *Timer1* del DW8051 per augmentar les variables temporals que controlen els temps del *start-up*, veure el diagrama de flux de la figura 6.67. el diagrama de flux de la figura 6.64 mostra el procés per controlar un motor.

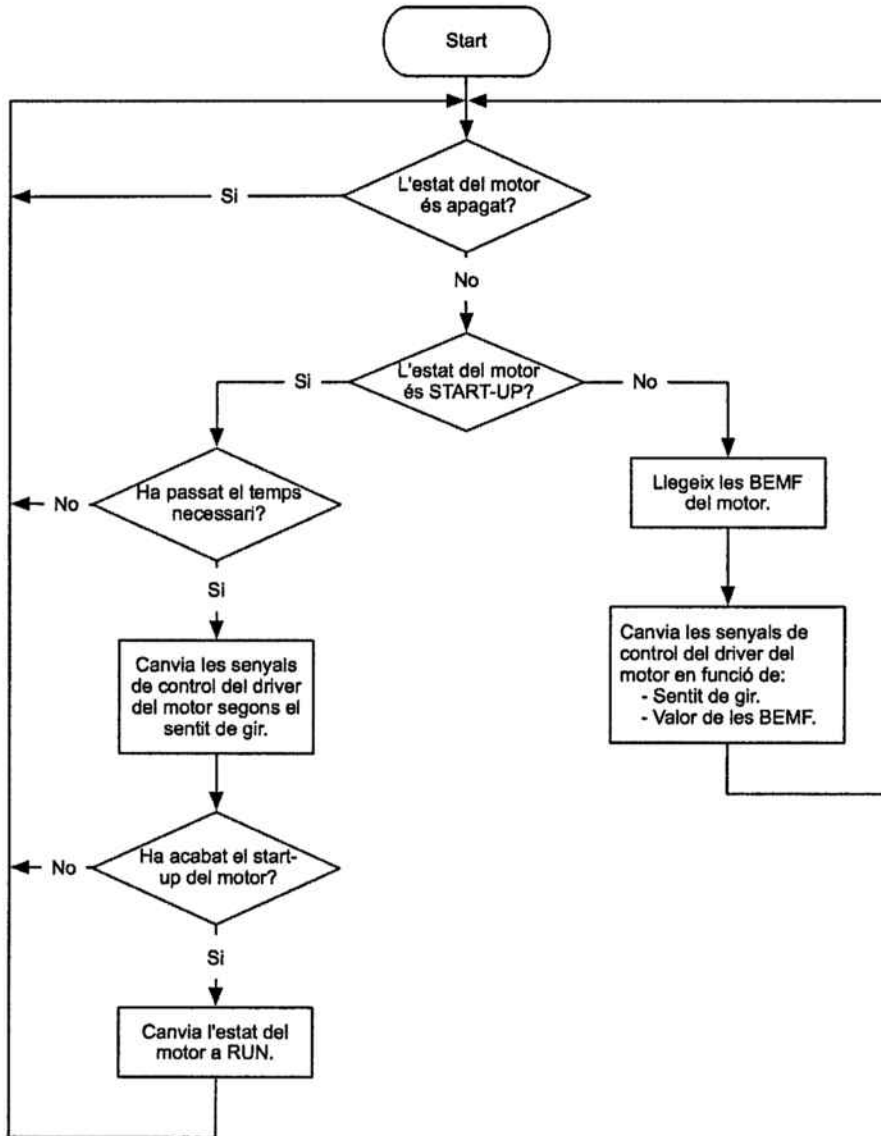


Figura 6.64: Diagrama de flux de la tasca *motors_ctrl*.

❖ Tasca *leds_state*

Els sistema d'il·luminació està format per 16 LEDs dividits en 4 grups de 4 LEDs cadascun.

- 4 LEDs d'infraroig.
- 4 LEDs vermells.
- 4 LEDs verds.
- 4 LEDs blaus.

Els LEDs infrarojos serveixen per detectar tumors, mentre que els LEDs colors serveixen per il·luminar. Per aconseguir una bona il·luminació és necessari encendre i apagar els grups de LEDs en un ordre i uns temps determinats que es mostren a la figura 6.65.

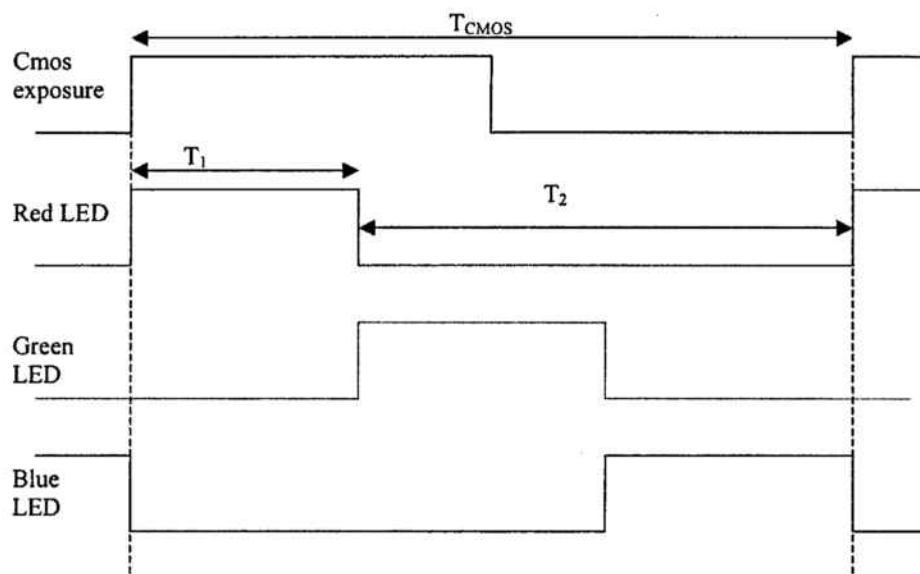


Figura 6.65: Diagrama temporal de il·luminació dels LEDs.

On El T_{CMOS} és 33ms i el cicle de treball dels LEDs és T_1/T_2 . El cicle de treball normalment és 1/3, però s'ha de poder modificar per controlar la llum blanca composta.

Per realitzar el control temporal s'utilitza el *Timer1* del DW8051. Dins la seva rutina d'interrupció s'augmenta una variable que depenent del seu valor fa que els LEDs s'encenguin o s'apaguin.

En el diagrama de flux de la figura 6.66 es mostra el funcionament de la tasca *leds_state* que permet encendre i apagar els leds en grups de 4 o tots a la vegada.

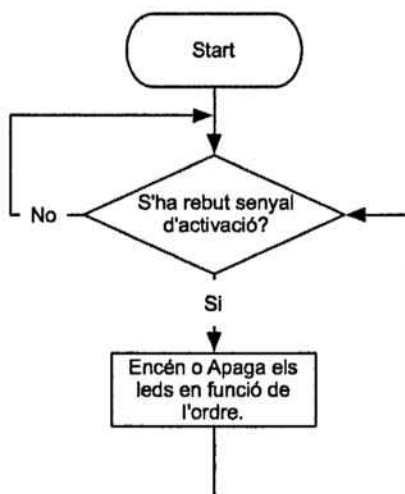


Figura 6.66: Diagrama de flux de la tasca *leds_state*.

❖ Rutines d'interruptió (ISRs)

El *TIMER 1* està configurat generar una interrupció periòdica, que s'utilitza per tenir una referència temporal en el control del *start-up* dels motors i és on es dur a terme el control dels LEDs en el mode *flash*.

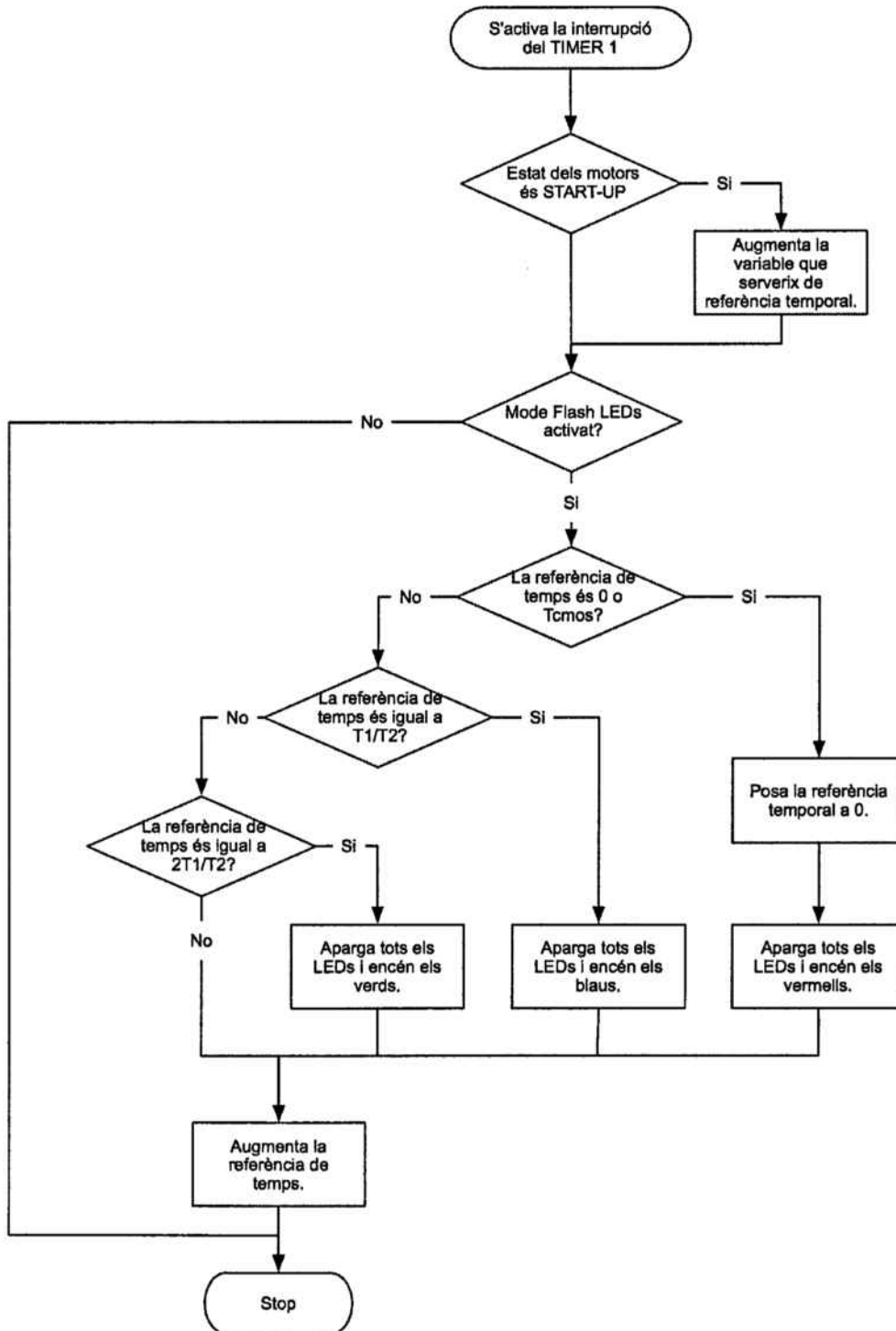


Figura 6.67: Diagrama de flux de la ISR del *TIMER 1*.



Figura 6.68: Diagrama de flux de la ISR de la interrupció de les comunicacions.

Taula 6.17: Instruccions disponibles per controlar el sistema.

Codi	Acrònim	#	MSB	LSB	Descripció
0x00	NO_OPERATION	1	0x00	0x00	No realitza cap operació
0x01	CAMERA_WR	1	0x00	0x01	Escriu al registre de la càmera amb adreça <i>addr</i> el valor <i>data</i>
		2	<i>data</i>	<i>addr</i>	
0x02	CAMERA_RD	1	<i>addr</i>	0x02	Llegeix el valor del registre amb adreça <i>addr</i> de la càmera.
0x11	MOTORS_OFF	1	0x00	0x11	Apaga els motors.
0x12	MOTOR1_ON_R	1	0x00	0x12	Activa el motor 1. Gir horari
0x13	MOTOR1_ON_L	1	0x00	0x13	Activa el motor 1. Gir antihorari.
0x14	MOTOR2_ON_R	1	0x00	0x14	Activa el motor 2. Gir horari.
0x15	MOTOR2_ON_L	1	0x00	0x15	Activa el motor 2. Gir antihorari.
0x21	LEDS_FLASH_NORMAL	1	<i>leds_period_ticks</i>	0x21	Activa el mode flash preestablert amb el període especificat i la intensitat dels LEDs indicada.
		2	<i>red_intensity</i>	<i>ir_intensity</i>	
		3	<i>blue_intensity</i>	<i>green_intensity</i>	
0x22	LEDS_FLASH_CUSTOM	1	0x00	0x22	Activa el mode flash amb el període, <i>duty cycle</i> i intensitat dels LEDs especificats.
		2	<i>leds_period_ticks</i>	<i>duty_cycle</i>	
		3	<i>red_intensity</i>	<i>ir_intensity</i>	
		4	<i>blue_intensity</i>	<i>green_intensity</i>	
0x23	LEDS_IR_ON	1	<i>ir_intensity</i>	0x23	Encén els LEDs infrarojos amb la intensitat especificada.
0x24	LEDS_RED_ON	1	<i>red_intensity</i>	0x24	Encén els LEDs vermells amb la

					intensitat especificada.
0x25	LEDS_GREEN_ON	1	green_intensity	0x25	Encén els LEDs verds amb la intensitat especificada.
0x26	LEDS_BLUE_ON	1	blue_intensity	0x26	Encén els LEDs blaus amb la intensitat especificada.
0x27	LEDS_IR_OFF	1	0x00	0x27	Apaga els LEDs infrarojos.
0x28	LEDS_RED_OFF	1	0x00	0x28	Apaga els LEDs vermells.
0x29	LEDS_GREEN_OFF	1	0x00	0x29	Apaga els LEDs verds.
0x2A	LEDS_BLUE_OFF	1	0x00	0x2A	Apaga els LEDs blaus.
0x2B	LEDS_ALL_ON	1	0x00	0x2B	Encén tots els LEDs amb la intensitat especificada per cada color.
		2	red_intensity	ir_intensity	
		3	blue_intensity	green_intensity	
0x2C	LEDS_ALL_OFF	1	0x00	0x2C	Apaga tots els LEDs
0x41	SFR_WRITE	1	sfr_addr	0x41	Escriu al registre SFR amb direcció <i>sfr_addr</i> el valor <i>sfr_data</i> .
		2	0x00	sfr_data	
0x42	SFR_READ	1	sfr_addr	0x42	Llegeix al registre SFR amb direcció <i>sfr_addr</i> . Mostra el valor llegit pel SFRGP.
0x43	I2C_WRITE	1	i2c_slave_addr	0x43	Escriu el registre de direcció <i>i2c_reg_addr</i> de l'esclau amb direcció <i>i2c_slave_addr</i> el valor especificat per <i>i2c_wr_data</i> .
		2	i2c_wr_data	i2c_reg_addr	
0x44	I2C_READ_NORMAL	1	i2c_slave_addr	0x44	Llegeix el registre de direcció <i>i2c_reg_addr</i> de l'esclau amb direcció <i>i2c_slave_addr</i> . El valor llegit es mostra pel SFRGP.
		2	0x00	i2c_reg_addr	
0x45	I2C_READ_RESTART	1	i2c_slave_addr	0x45	Llegeix el registre de direcció <i>i2c_reg_addr</i> de l'esclau amb direcció <i>i2c_slave_addr</i> . El valor llegit es mostra pel SFRGP. Mitjançant transferència combinada.
		2	0x00	i2c_reg_addr	

El codi font del software de control realitzat es troba a l'annex 11.8.

7 Viabilitat tècnica i econòmica

7.1 Especificacions i característiques tècniques

Les eines utilitzades són les següents:

- Programari:
 - *coreConsultant* de Synopsis
 - IP DW8051 de Synopsis
 - *Design Compiler* de Synopsis
 - *Quartus II* d'Altera
 - *uVision3* de Keil amb el PK51.
- Hardware/equip de laboratori:
 - PC on corre els programes
 - Oscil·loscopi: *Agilent infiniium* MSO8104A.
 - Generador de funcions
 - Placa DE2 (conté una FPGA *Cyclon II* d'Altera)

7.2 Costos d'inversió

Els costos d'inversió del projecte es divideixen en dues parts importants. La primera part correspon al cost de l'enginyer que ha desenvolupat el prototipatge i el software de control, i la segona part inclou el cost dels equip necessaris per la realització del projecte.

Taula 8.1: Costos del projecte.

Concepte	Preu
Equip informàtic	2.000 €
Equip de laboratori	350€
Llicències de software	3000€
Hores d'enginyer	6.000 €
TOTAL	11.350€

7.3 Costos d'explotació

Aquest projecte s'emmarca dins del projecte VECTOR, que actualment està en fase de prototip. El cost d'explotació es pot considerar nul, ja que al ser el software de control, un cop dissenyat només s'ha d'introduir a la memòria del processador, procés que es pot dur a terme durant la fabricació o posteriori.

7.4 Finançament

El disseny del software de control així com el prototipatge en la FPGA es troba dins del projecte *Versatile Endoscopic Capsule for Gastrointestinal tumor recognition and therapy* (VECTOR) recolzat per la Unió Europea dins el 6è *EU Framework Programme for Research and Technological Development (FP6)*.

8 Cronograma d'execució

8.1 Definició de tasques i temps

El temps previst per l'execució del projecte és de sis mesos. Les tasques que han de realitzar-se en aquest temps són les que es descriuen a continuació:

- **Fase de concepció.** Es realitzarà un estudi dels diferents bloc de l'electrònica de control, per tal de determinar-les possibles solucions. Aquesta etapa tindrà una duració d'uns 2 mesos.
- **Fase d'implementació i depuració.** En aquesta fase es realitzarà la implementació de l'electrònica de control en una FPGA, així com dels diferents blocs necessaris per comprovar la correcta interacció. Aquí també s'inclou la realització de petit programes per comprovar el correcte funcionament. Aquesta etapa tindrà una duració d'uns 2 mesos.
- **Fase de disseny del software.** Es desenvoluparan les tasques individuals que controlarà cada bloc del sistema, primer de forma individual i després es comprovarà el correcte funcionament col·lectiu. Aquesta etapa tindrà una duració d'uns 3 mesos.
- **Fase de test.** Es comprova que funciona tot correctament i es realitzen les tasques com era d'esperar. Aquesta etapa tindrà una duració d'un mes.

8.2 Fases i objectius

En cada una de les fases esmentades anteriorment hi ha un objectiu o objectius concrets que cal complir:

- **Fase de concepció.** S'han de realitzar les següents tasques:
 - Estudi dels diferents bloc de l'electrònica de control per determinar les possibles solucions.
 - Estudi de funcionament dels RTOS.
- **Fase d'implementació i depuració.** S'han de realitzar les següents tasques:
 - Generar la *netlist* del DW8051 amb la tecnologia de la FPGA *Cyclon II*.
 - Implementar el DW8051 a la FPGA i comprovar el seu correcte funcionament.
 - Implementar tota l'electrònica digital de control a la FPGA, canviant les memòries pels models de la FPGA.
 - Desenvolupar petits programes i blocs digitals necessaris per comprovar el correcte funcionament dels diferents blocs de l'electrònica digital de control.
 - Corregir els errors trobats en l'electrònica digital de control.
- **Fase de disseny del software.** S'han de realitzar les següents tasques:
 - Disseny de les tasques necessàries per el control de cada bloc del sistema.
 - Comprovació del correcte funcionament de cada tasca.
 - Unió de totes les tasques determinant prioritat i correspondència entre tasques en els cassos necessaris.
- **Fase de test.** Es comprova el correcte funcionament de tot el sistema de control dissenyat. Dins el prototip fabricat.

8.3 Assignació de responsabilitats

El projectista realitzarà totes les labors assignades amb l'ajuda del tutor.

El tutor del projecte facilitarà el material necessari i els medis per poder realitzar el projecte. El mateix tutor serà el que determinarà la validesa o no dels resultats obtinguts en cada fase.

8.4 Cronograma

A l'esquema següent es presenta la previsió del cronograma d'execució de les diferents tasques que constitueixen el projecte, des del moment de la seva assignació fins la data prevista per l'entrega.

	1er Mes	2on Mes	3er Mes	4rt Mes	5é Mes	6é Mes
Assignació						
Fase de concepció						
Fase implementació i depuració						
Fase de disseny del software						
Fase test						
Realització de la memòria tècnica						

Figura 8.1: Cronograma d'execució.

9 Conclusions

9.1 Conclusions

El projecte de prototipatge i depuració de l'electrònica digital de control i el disseny del software de control d'aquesta basat en un RTOS empotrat en el processador DW8051, es troba englobat dins el projecte VECTOR i més concretament en el disseny de l'ASIC de dimensions reduïdes que contindrà l'electrònica de control.

Concretament s'ha sintetitzat tota l'electrònica digital de control en una FPGA per poder disposar d'una plataforma on executar i depurar el software de control dissenyat en temps real abans de disposar de l'ASIC del xip VECTOR V1. D'aquesta manera s'ha pogut desenvolupar el software juntament amb el hardware, de manera que abans de tenir el xip final s'ha pogut comprovar que tot funciona correctament gràcies a les funcions definides al software.

A més, el prototipatge ha permès la depuració dels diferents blocs del sistema. Gràcies als tests realitzats s'ha pogut detectar un error en la interfície de connexió de la memòria RAM interna del sistema i s'ha corregit. Aquest error impedia l'accés indirecte de la memòria en qüestió, fet que hauria impedit desenvolupar el software de control de manera normal, ja que s'haurien hagut d'evitar l'ús de adreçaments indirectes a la memòria RAM interna. Cal remarcar que aquest error es va detectar un cop el xip ja estava fabricat. Degut això, el software desenvolupat amb *RTX Tiny* no s'ha pogut fer corre en el xip VECTOR V1. Tot i això les tasques dissenyades s'han pogut provar en el xip final, per mitja de transformar el codi fet amb el *RTX Tiny* en codi C normal. S'ha retocat el codi just per eliminar la presència del *RTX Tiny* però mantenint la mateixa estructura i funcions.

El software de control generat permet controlar la capacitat de locomoció, visió i il·luminació des del exterior. S'ha establert un sistema d'instruccions que es reben per el port de comunicacions, de manera que segons l'ordre que es rep el sistema realitza una acció determinada. A més, el software de control permet configurar tots els registres de control i configuració dels blocs que formen el sistema permetent un control total des de l'exterior.

Per altra banda, s'ha comprovat que la utilització d'un sistema operatiu en temps real facilita la creació d'aplicacions que requereixen una espera mínima d'interrupció i una espera mínima de commutació de tasca. Proporciona una resposta ràpida a esdeveniments externs previstos, el que el converteix en una eina ideal a l'hora de dissenyar el software de control de petits sistemes que requereixen d'una resposta en temps real a esdeveniments previsibles, com és el cas d'una capsula endoscòpica on s'han de controlar constantment els sistemes d'il·luminació, visió, locomoció i comunicacions.

9.2 Millores proposades

Les millores que es poden realitzar del projecte es centren en el software de control, ja que aquest només consta de les tasques bàsiques per controlar els diferents blocs del sistema mitjançant instruccions que es reben des del exterior. Es poden afegir instruccions més específiques segons les necessitats sorgides quan es disposi de la versió final de l'electrònica que contindrà la càpsula VECTOR.

10 Bibliografia i fonts d'informació

❖ Capsules endoscòpiques

<http://www.vector-project.com>

<http://www.ciberhabitat.gob.mx/hospital/capsula/>

http://www.olympus-europa.com/endoscopy/2001_5491.htm

http://www.olympusamerica.com/msg_section/download_brochures/EndoCapsule_Brochure.pdf

<http://www.omom.us/main.php?sLAN=en>

<http://www.intromedic.com/en/product/productInfo.asp>

<http://www.givenimaging.com/en-us/patients/pages/CapsuleEndoscopy.aspx>

<http://www.rfamerica.com/sayaka/index.html>

<http://www.research.philips.com/newscenter/backgrounders/081111-IntelliCap.html>

<http://www.nano-tera.ch/nanoterawiki/WirelessEndoscopeCamera>

http://www.olympusamerica.com/presspass/press_pass_cut/documents/OlympusMedicalInnovations.pdf

<http://www.capsuloendoscopia.com/central.htm>

Arkady Glukhovsky "Wireless capsule endoscopy." *Sensor Review* Volume 23 Number 2 2003 pp.128–133

Andrea Moglia¹, Arianna Menciassi and Paolo Dario "Recent Patents on Wireless Capsule Endoscopy" *Recent Patents on Biomedical Engineering* 2008, 1, 24-33

❖ RTOS

<http://www.keil.com/rtx51/specs.asp>

http://micrium.com/newmicrium/uploads/file/datasheets/ucosii_datasheet.pdf

http://es.wikipedia.org/wiki/Sistema_operativo_de_tiempo_real

http://en.wikipedia.org/wiki/Real-time_operating_system

<http://www.on-time.com/rtos-32-docs/>

❖ Manuals i Datasheets

Vector_v1 datasheet Rev.1.1 Rel.27/05/09.

Vector2 datasheet Rel 01/2008.

The I2C-Bus Specification Version 2.1 January 2000.

Altera DE2 Board User Manual.

Synopsys DesignWare DW8051 MacroCell Databook Rel June 20, 2005 Version 3.80a.

Manual de *RTX51 Tiny* : http://www.keil.com/support/man/docs/tr51/tr51_overview.htm

José Adolfo González Vázquez, "Introducción a los microcontroladores. Hardware – Software - Aplicaciones", Ed Mc Graw Hill, 1992.

❖ **Altres**

<http://boe.es/boe/dias/2009/11/06/pdfs/BOE-A-2009-17607.pdf>

<http://www.boe.es/boe/dias/2009/11/06/pdfs/BOE-A-2009-17606.pdf>

<http://www.emergogroup.com/files/EUROPE-CONSOLIDATED-90-385-EEC.pdf>

<http://www.emergogroup.com/files/EUROPE-CONSOLIDATED-MDD-93-42-EEC.pdf>

<http://www.emergogroup.com/files/In-Vitro-Diagnostics-Directive-98-79-EC-v2.pdf>

11 Annexes

11.1 Generació *netlist* del DW8051 amb *coreConsultant*

Per poder executar el *coreConsultant* prèviament s'ha de executar la següent comanda,

- `source .csrhc_snps`

Executem el *coreConsultant* que permet generar la *netlist* del DW8051 amb la comanda,

- `coreConsultant &`

Creem un nou directori de treball des del menú **File -> New Workspace**, on escollim

- Nom de la configuració, per exemple `FPGA_SYN_CYCLONE2`.
- Directori arrel (o de treball) on es genera una carpeta amb el nom de la configuració escollit i que contindrà tots els fitxers generats en el procés.
- Especificar on hi ha el **corekid**:

`/data/soft/synopsys/DW8051-3.70a_download/DW8051/3.70a`

Un cop creat el nou espai de treball, seguim els passos descrits en l'**Activity List**.

Escollir especificació:

- | | |
|--------------------------------------|-------------|
| • Size of Internal RAM: | 256 bytes |
| • Timer 2: | No Included |
| • Internal ROM Add Bus Width (bits): | 0 |
| • Number of Serial Ports: | One |
| • Interrupt Unit Type: | No Extended |

Un cop escollida la configuració les apliquem pitjant al botó **Apply**. El següent pas és especificar la tecnologia que volem, figura 11.1. Per fer-ho hem d'indicar on es troben les llibreries de la tecnologia.

A l'apartat `dc_shell search_path variable`, hem d'afegir el següent `path`:

- `/data/soft/altera/quartus5.0/eda/synopsys/dc/syn/cycloneii/lib`

L'inclusió del `path` es fa prement a l'icona de **Insert New item** , i es pot escriure directament o navegar pel sistema de fitxers fins la carpeta desitjada.

Un cop posat el `path`, hem de incloure les llibreries necessàries de la mateixa manera que el `path` però aquest cop seleccionarem d'una llista que ens apareixerà al pitjar sobre la icona de **Insert New item**:

- | | |
|--|---------------------------|
| • <code>dc_shell target_library variable:</code> | <code>cycloneii.db</code> |
| • <code>dc_shell link_library variable:</code> | <code>cycloneii.db</code> |

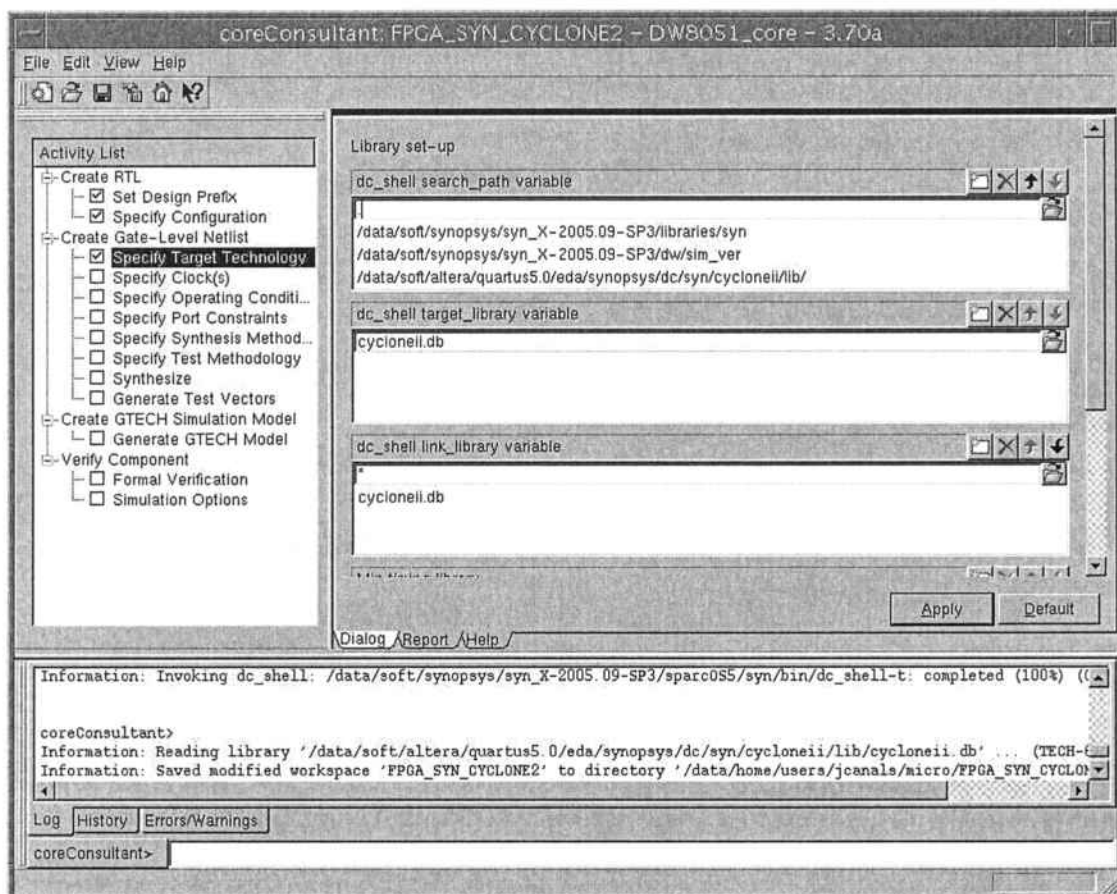


Fig. 11.1: Interfície coreConsultant.

Un cop aplicada la tecnologia, el següent pas és especificar la freqüència de funcionament que volem, és a dir, el rellotge del nostre sistema. En el nostre cas serà un rellotge de 20MHz, per tant, té un cicle de 50ns.

- CycleTime (ns): 50

La resta de paràmetres es deixen els valors per defecte. Els passos següents fins arribar a *Synthesize* no cal modificar cap paràmetre. Els punts no modificats serveixen per millorar el disseny amb restriccions (*constrains*), per evitar problemes amb retards en senyals, prioritzar àrea, entre d'altres.

Un cop acabat el pas de *Synthesize*, el fitxer amb la *netlist* generada es troba en el directori,

- ./directori_treball/nom_config/syn/final/db/DW8051_core.v

L'últim pas abans de poder importar la *netlist* al Quartus II és passar el **DW8051_core.v** resultant a **DW8051_core.edif**, ja que el fitxer en format verilog no és directament sintetitzable a la FPGA.

Primer s'ha de crear una nova carpeta on haurem de crear un fitxer amb el nom, **.synopsys_dc.setup**, que ha de contenir el següent,

```
define_design_lib work -path ./WORK
set search_path {/data/soft/altera/quartus5.0/eda/synopsys/dc/syn/cycloneii/lib};
set target_library {cycloneii.db};
set symbol_library {altera.sdb};
set synthetic_library {cycloneii.sldb};
set link_library {cycloneii.sldb cycloneii.db};
define_design_lib DW_CYCLONEII -path /data/soft/altera/quartus5.0/eda/synopsys/dw/lib ;
define_design_lib altera -path /data/soft/altera/quartus5.0/eda/synopsys/mf/src ;
set edifout_netlist_only "true";
```

```

set edifout_netlist_only "true";
set edifout_power_and_ground_representation "net";
set edifout_power_net_name "VDD";
set edifout_ground_net_name "GND";
set edifout_no_array "false";
set edifin_power_net_name "VDD";
set edifin_ground_net_name "GND";
set compile_fix_multiple_port_nets "true"
set bus_naming_style "%s<%d>";
set bus_dimension_separator_style "><";
set bus_inference_style "%s<%d>";

```

Un cop generat aquest fitxer i guardat en el directori escollit ja podem executar el Design Compiler dins el directori on hem guardat `.synopsys_dc.setup`. Per fer-ho executem la commanda:

- `design_vision -tcl_mode -db_mode`

Per convertir el fitxer verilog en un fitxe edif només hem de fer els següent:

- *File->Read:* `DW8051_core.v`
- *File->Save as:* **`DW8051_core.edif`**

Ara ja tenim la *netlist* preparada per ser importada al Quartus II.

11.2 Creació del sistema bàsic basat en el DW8051 i sintetització del mateix en una FPGA amb Quartus II

Primer hem de crear un directori de treball on guardar els nostres fitxers i futurs fitxers generats.

C:\SysBasic

Dins d'aquest directori de treball creem dues carpetes:

- **C:\SysBasic\src**: contindrà el codi font del nostre sistema
- **C:\SysBasic\syn_fpga**: contindrà els fitxers generats de la síntesis del projecte amb el *Quartus II*.

Un cop definit el nostre sistema d'arxius, cal generar el codi font del sistema basat en el DW8051 que es vol implementar a l'FPGA. El sistema a implementar consta de:

- **DW8051** de *Synopsys* generat en l'annex 11.1.
- **Power On Reset**: encarregat de generar el reset del sistema a partir d'un reset extern asíncron. El codi font que el descriu en llenguatge verilog és el següent:

```
/*
 * Sincronitza el reset del sistema i el retarda per
 * mitja de un registre de desplaçament.
 * Permetre que el 8051 faci bé el reset ja que primer ens
 * assegurem que rep bé el rellotge del sistema abans de fer el * reset.
 */

module power_on_reset(// Power-On-Reset
    clk,
    por_i,
    por_o
);

parameter dlenght = 12;

input      clk;           // Rellotge del sistema
input      por_i;         // Reset d'entrada al sistema actiu nivell
output     por_o;         // Reset retardat 12 cicles e rellotge

reg [dlenght-1:0] delay_chain;

// registre de desplaçament
always @(posedge clk or negedge por_i)
    if (!por_i) delay_chain <= 12'd0;
    else begin
        delay_chain[0] <= 1'b1;
        delay_chain[dlenght-1:1] <= delay_chain[dlenght-2:0];
    end

assign #5 por_o = delay_chain[dlenght-1];

endmodule
```

- **Registre SFR**: registre que s'escriu per mitja del bus intern del sistema. El codi font que el descriu en llenguatge verilog és el següent:

```
/*
 * Registre SFR corresponent a l'adreça 80h.
 */

module sfr_reg(
    clk,
    rst_n_i,
    sfr_rd,
    sfr_wr,
    sfr_addr,
    data_in,
    data_out,
    reg80h
);

input      clk;           // Rellotge del sistema
input      rst_n_i;       // Reset del sistema actiu nivell baix
input      sfr_rd;        // Activa lectura del SFR
input      sfr_wr;        // Activa escriptura del SFR
input [7:0] data_in;      // Dada a escriure al SFR
input [7:0] sfr_addr;     // Bus Addr SFR
output [7:0] data_out;    // Dada llegida del SFR
output [7:0] reg80h;      // Dada que conte el SFR
```

```

reg    [7:0] reg80h;           // Registre SFR 80h
wire   cs_80h;                 // Chip Select del registre

assign cs_80h = ((sfr_addr == 8'h80) ? 1'b1 : 1'b0);
// Lectura SFR
assign data_out = ((cs_80h && sfr_rd) ? reg80h : 8'h00);
// Escriptura SFR
always@(posedge clk or negedge rst_n_i)
    if(!rst_n_i)
        reg80h = 8'h00;
    else if(cs_80h && sfr_wr)
        reg80h = data_in;

endmodule

```

- **Memòries:** són les IPs pròpies d'Altera, s'han de generar amb el Quartus II.

Un cop vistes les parts del sistema i el codi que el descriu, el primer que cal fer és **crear un nou projecte** al *Quartus II* l'obrim i seguim els següents passos:

1. **File -> New Project Wizard**, ens apareix una finestra amb una petita introducció, pitgem **Next**.
2. Apareix una finestra com la de la figura 11.2, on hem d'especificar:
 - Directori: **C:\SysBasic\syn_fpga**
 - Nom del projecte: **top_system_8051**
 - Nom del *top-level entity* del nostre projecte: **top_system_8051**

Cal notar que normalment aquest dos darrers punts acostumen a coincidir. Pitgem a **Next**

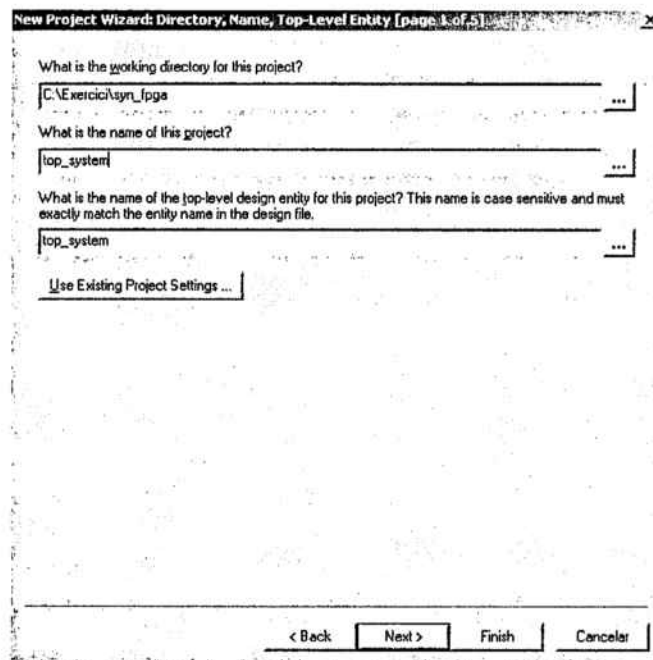


Fig. 11.2: Especificació del directori, nom i top.

3. Apareix la finestra de la figura 11.3, En aquest pas hem d'afegir els fitxers que ja tenim creats que formen part del nostre sistema, en aquest cas la *netlist* del DW8051 creada anteriorment. Per fer-ho pitgem sobre el botó dels punt suspensius, seleccionem el fitxer **DW8051_core.edif** i l'afegim amb el botó **Add**. Pitgem a **Next**.

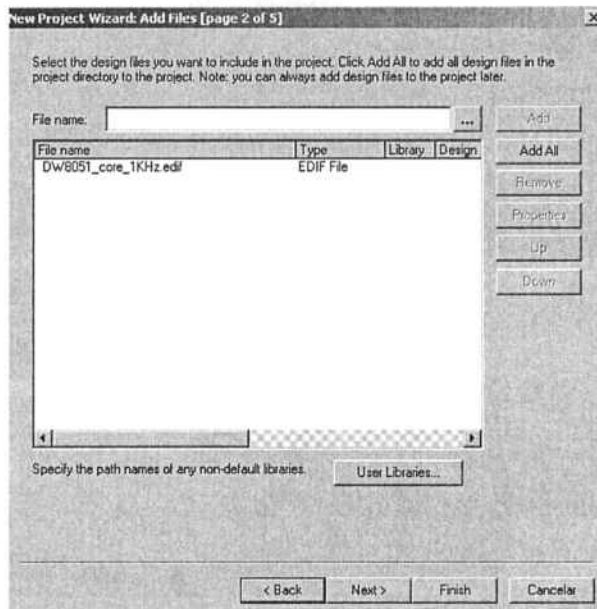


Fig. 11.3: Afegir fitxers al projecte.

4. Apareix la finestra de la figura 11.4 on hem de seleccionar la FPGA a utilitzar. En aquest cas utilitzem la FPGA de la família **Cyclone II** i el model és **EP2C35F672C6**. Un cop seleccionada la FPGA premem sobre **Next**.

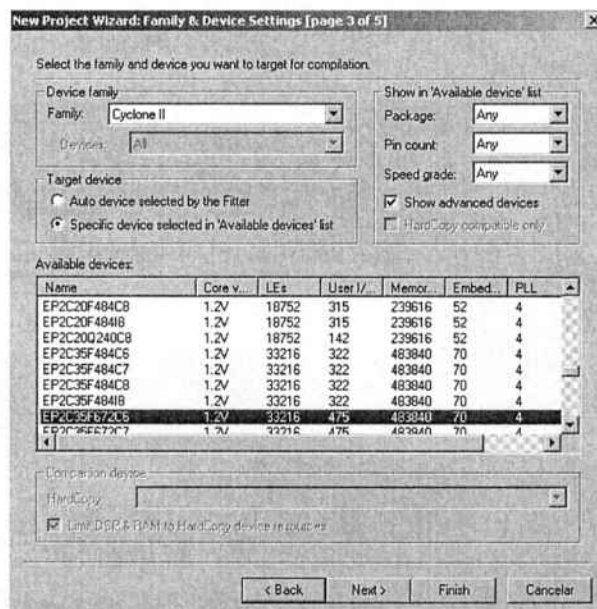


Fig. 11.4: Família i configuració de la FPGA.

5. Ara se'ns donà l'oportunitat d'especificar altres eines de disseny. En el nostre cas com la *netlist* del 8051 la tenim amb format *.edif*, hem d'especificar l'eina de **Design Entry/Synthesis**, que és el **Design Compiler** i el format **EDIF**, tal i com es mostra en la figura 11.5. Pitgem **Next**.

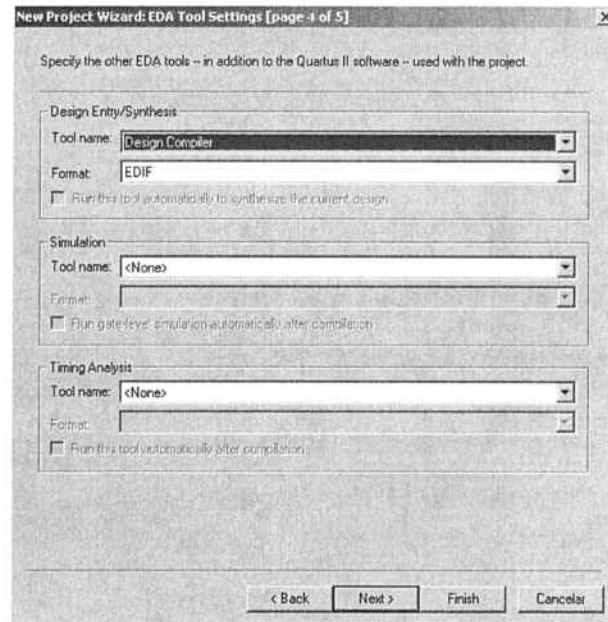


Fig. 11.5: Configuració de l'EDA Tool.

6. Mostra una ultima finestra on hi ha un resum del que hem anat escollint per configurar el nostre projecte, pitgem **Finish**.

Una vegada creat el nou projecte hem de generar les memòries de 2 KB, 256 Bytes i 8 KB amb el *Quartus II*. Per generar la memòria de 2 KB s'han de seguir els següents passos.

1. **File -> New-> Block Diagram/Schematic**. A la nova pestanya que s'obre pitgem amb el botó dret del ratolí i escollim **Insert -> Symbol**, i ens apareix una finestra com la de la figura 11.6.

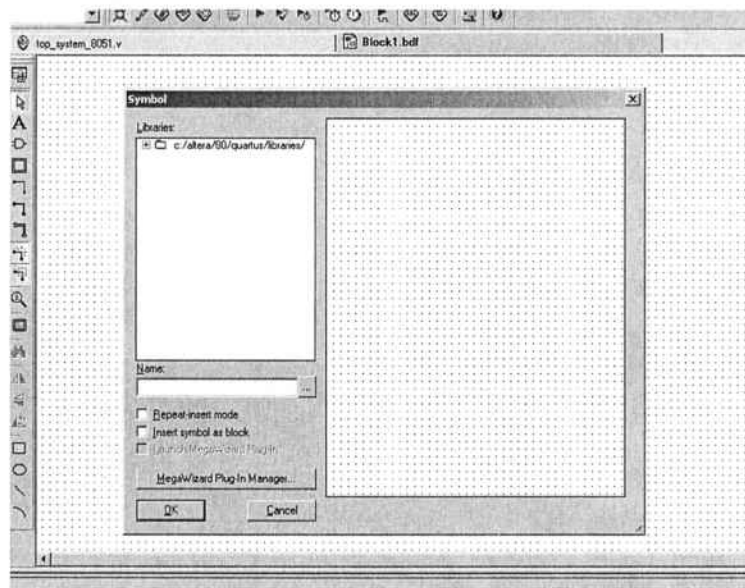


Fig. 11.6: Finestra per afegir crear symbols.

2. Pitgem sobre **MegaWizard Pulg-in-manager**.
3. A la finestra que pareix seleccionem **Create a new custom megafunction variation** pitgem **Next** i ens apareix la finestra, figura 11.7.

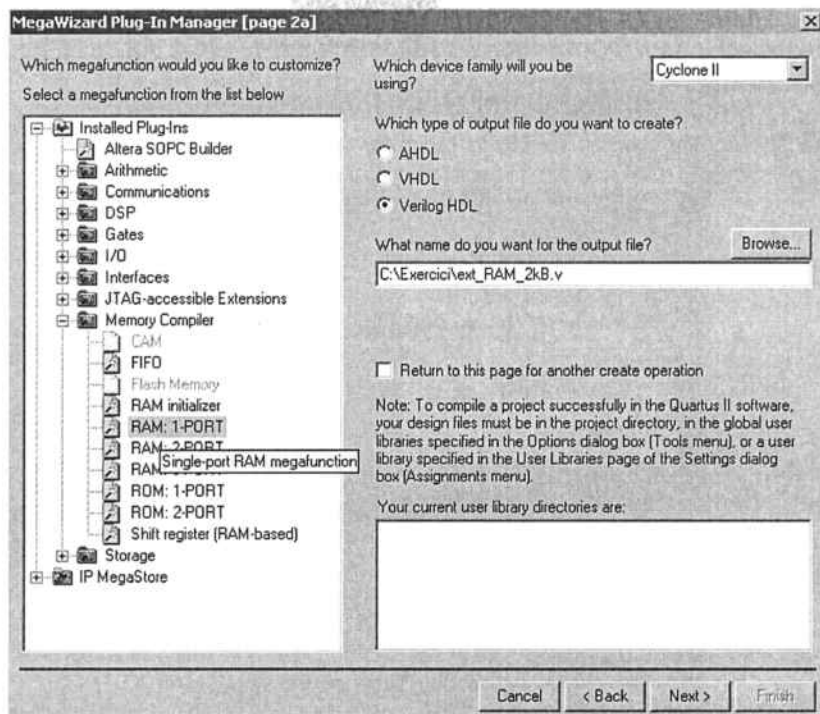


Fig. 11.7: 1er pas del *MegaWizard Plug-In Manager*.

4. Seleccionem **Memory Compiler** -> **RAM:1-PORT** de la llista de *megafuncions* de l'esquerra. A la dreta s'ha d'especificar:
 - o FPGA a utilitzar: **Cyclon II**
 - o Tipus de fitxer de sortida que generarem: **verilog**
 - o Directori on es generaran els fitxers i nom dels mateixos: **C:\SysBasic\src\ext_RAM_2kB.v**
5. Pitgem **Next**. El següent pas es especificar la mida de la memòria, segons el numero de bits per paraula i el nombre de paraules, 8 i 2048 respectivament ja que es vol una memòria RAM externa de 2 KB, figura 11.8. La resta de paràmetres es deixen amb el valor per defecte. Pitgem **Next**.

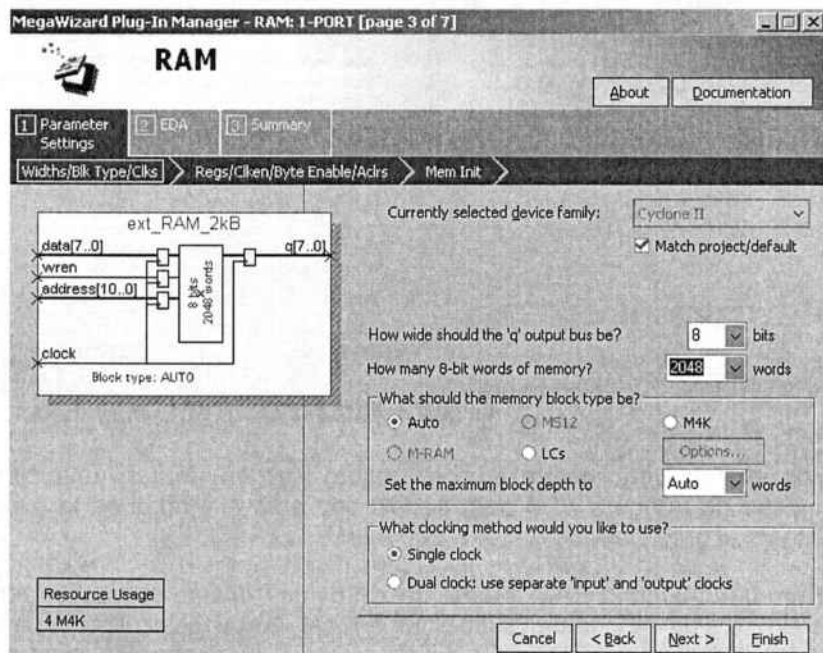


Fig. 11.8: 2on pas del *MegaWizard Plug-In Manager*.

6. Seguidament es pregunta si es vol registrar alguna senyal, figura 11.9, desseleccionem 'q' output port i pitgem **Next**.

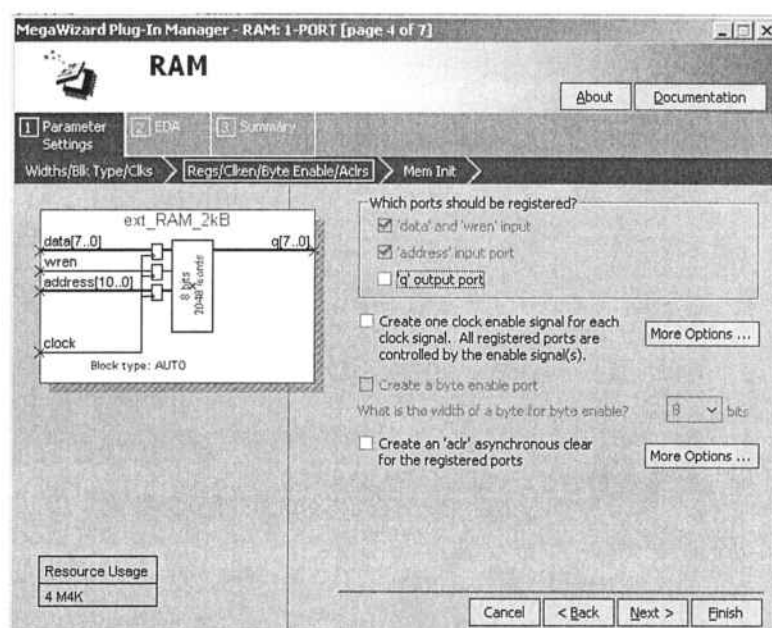


Fig. 11.9: 3er pas del MegaWizard Plug-In Manager.

7. Apareix una nova finestra, figura 11.10, on podem escollir si volem deixar la memòria en blanc o si la volem inicialitzar a uns valors, per mitjà de fitxers .mif o .hex. En aquest cas la deixem en blanc. Pitgem **Finish**.

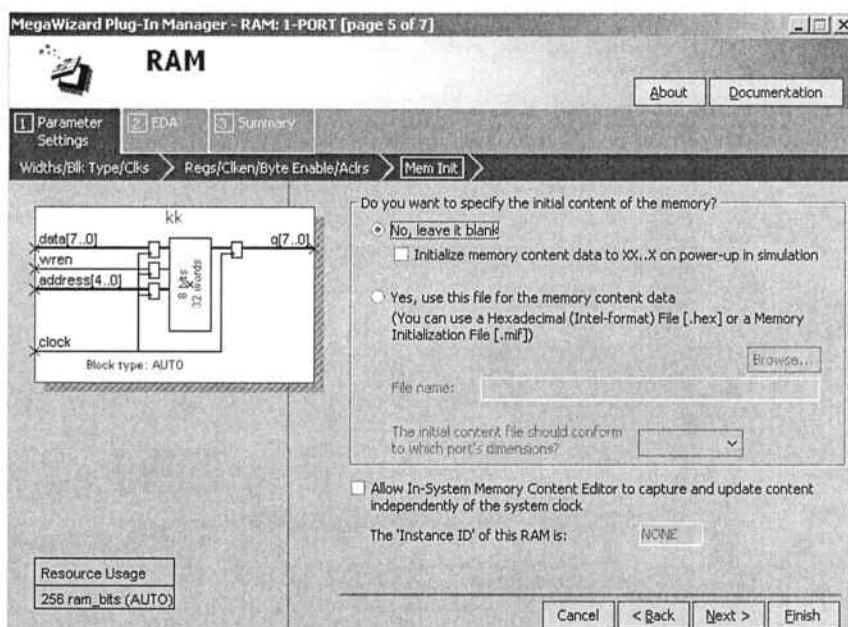


Fig. 11.10: 4rt pas del MegaWizard Plug-In Manager.

8. Ens apareix una finestra amb un resum i pitgem **Finish** novament. Reapareix la finestra de la figura 11.6 però aquest cop amb el símbol de la memòria generada. Pitgem al botó **Cancel**.

Per generem una memòria RAM interna de 256Bytes, repetim el procés, però aquest cop el nom amb el que s'ha de guardar el fitxer és **i_RAM_256.v**, seleccionem paraules de 8 bits i una mida de 256 paraules, la resta de paràmetres es configuren igual que per la RAM externa.

Per generar la memòria de programa (ROM) de 8 KB, el procés a seguir és similar al de les RAMs. Però en el pas 4 hem de seleccionar **ROM:1-PORT** i el nom que li hem de donar és **ext_ROM_8kB.v**. Després com es vol una memòria ROM de 8 KB, les paraules han de ser de 8bits i la mida de la memòria de 8192 paraules. Com en el cas de la RAM s'han de des-registrar les sortides (desseleccionar 'q'). Per últim en el pas 8, si que hem de dir que volem inicialitzar-la carregant-li un fitxer .hex generat prèviament amb *Keil uVision3*, annex 11.3, aquest fitxer es pot canviar sempre que es vulgui .

Un cop creades les memòries, hem de crear els fitxers verilog del **Power On Reset** i del **Registre SFR 80h** que contindran el codi font que hem vist anteriorment.

Per crear un fitxer verilog amb el *Quartus II* s'ha de fer,

File->New->Verilog HDL file

s'obrirà una nova finestra on podrem descriure en llenguatge verilog el sistema digital desitjat. En el nostre cas copiarem el codi font del power on reset i el guardarem,

File->Save As

amb el mateix nom que el modul més l'extensió **power_on_reset.v** a la carpeta **C:\SysBasic\src**.

Farem el mateix per genera el fitxer verilog del registre SFR 80h, però amb els nom **sfr_reg.v**.

L'últim fitxer a generar és el top, on instanciaré tots els blocs (memòries, DW8051, power on reset i el registre sfr) que formen part del sistema bàsic, i les entrades i sortides que tindrà aquest. Per fer-ho hem de crear un fitxer verilog seguint els mateixos passos que amb els dos anteriors i de nom **top_system_8051.v** que contindrà els següent codi:

```
module top_system_8051(
    //inputs
    clk_i,
    rst_n_i,
    //outputs
    t0_o,
    reg80h_o
);

input        clk_i;
input        rst_n_i;

output       t0_o;
output [7:0] reg80h_o;

wire [7:0] iram_addr;
wire [7:0] sfr_data_in;
wire [7:0] mem_data_in;
wire [7:0] iram_data_out;
wire [7:0] irom_data_out;
wire [7:0] iram_data_in;
wire [7:0] sfr_addr;
wire [7:0] sfr_data_out;
wire [7:0] mem_data_out;
wire [15:0] irom_addr;
wire [15:0] mem_addr;
wire        test_mode_n, mem_ea_n, pfi, wdti, rxd0_in, rxd1_in, t0,
t1, t2, t2ex;
wire        rst_out_n, stop_mode_n, por_n, idle_mode_n, sfr_wr,
sfr_rd, mem_wr_n, mem_rd_n, mem_pswr_n, mem_psr_n,
mem_ale, rxd0_out, txd0, rxd1_out, txd1, t0_out, t1_out,
t2_out, port_pin_reg_n, p0_mem_reg_n, p0_addr_data_n,
p2_mem_reg_n, iram_rd_n, iram_we1_n, iram_we2_n,
irom_rd_n, irom_cs_n;

wire        clk_inv;
wire [7:0] mem_data_in_8192, mem_data_in_2048;

assign clk_inv      = !clk_i;
assign mem_data_in  = (mem_rd_n ? mem_data_in_8192:mem_data_in_2048);
assign irom_data_out = 0;
assign test_mode_n  = 1'b1;

DW8051_core DW8051(.clk            (clk_i),
                   .por_n          (por_n),
                   .rst_in_n       (por_n),
                   .rst_out_n      (rst_out_n),
                   .test_mode_n    (test_mode_n),
                   .stop_mode_n    (stop_mode_n),
```

```

        .idle_mode_n      (idle_mode_n),
        .sfr_addr         (sfr_addr),
        .sfr_data_out     (sfr_data_out),
        .sfr_data_in      (sfr_data_in),
        .sfr_wr           (sfr_wr),
        .sfr_rd           (sfr_rd),
        .mem_addr         (mem_addr),
        .mem_data_out     (mem_data_out),
        .mem_data_in      (mem_data_in),
        .mem_wr_n         (mem_wr_n),
        .mem_rd_n         (mem_rd_n),
        .mem_pswr_n       (mem_pswr_n),
        .mem_psrn_n       (mem_psrn_n),
        .mem_ale          (mem_ale),
        .mem_ea_n         (1'b0),
        .int0_n           (1'b1),
        .int1_n           (1'b1),
        .int2             (1'b0),
        .int3_n           (1'b1),
        .int4             (1'b0),
        .int5_n           (1'b1),
        .pfi              (1'b0),
        .wdti             (1'b0),
        .rxdo_in          (1'b0),
        .rxdo_out         (),
        .txdo             (),
        .rxdi_in          (1'b0),
        .rxdi_out         (),
        .txdi             (),
        .t0               (1'b0),
        .t1               (1'b0),
        .t2               (1'b0),
        .t2ex             (1'b0),
        .t0_out           (t0_o),
        .t1_out           (),
        .t2_out           (),
        .port_pin_reg_n   (port_pin_reg_n),
        .p0_mem_reg_n     (p0_mem_reg_n),
        .p0_addr_data_n   (p0_addr_data_n),
        .p2_mem_reg_n     (p2_mem_reg_n),
        .iram_addr        (iram_addr),
        .iram_data_out     (iram_data_out),
        .iram_data_in     (iram_data_in),
        .iram_rd_n        (iram_rd_n),
        .iram_we1_n       (iram_we1_n),
        .iram_we2_n       (iram_we2_n),
        .irom_addr        (irom_addr),
        .irom_data_out     (irom_data_out),
        .irom_rd_n        (irom_rd_n),
        .irom_cs_n        (irom_cs_n)
    );

    ext_ROM_8kB EXTROM(.address      (mem_addr[12:0]),
                      .clock         (clk_inv),
                      .q             (mem_data_in_8192)
    );

    ext_RAM_2kB EXTRAM(.address      (mem_addr[10:0]),
                      .clock         (clk_inv),
                      .data          (mem_data_out),
                      .wren          (!mem_wr_n),
                      .q             (mem_data_in_2048)
    );

    i_RAM_256 iRAM(.address      (iram_addr),
                  .clock         (clk_inv),
                  .data          (iram_data_in),
                  .wren          (!iram_we2_n),
                  .q             (iram_data_out)
    );

    power_on_reset POR(.clk        (clk_i),
                      .por_i       (rst_n_i),
                      .por_o       (por_n)
    );

    sfr_reg SFRREG(.clk          (clk_i),
                  .rst_n_i       (por_n),
                  .sfr_rd        (sfr_rd),
                  .sfr_wr        (sfr_wr),
                  .sfr_addr      (sfr_addr),
                  .data_in       (sfr_data_out),
                  .data_out      (sfr_data_in),
                  .reg80h        (reg80h_o)
    );
endmodule

```

Un cop generats els fitxers que descriuen el nostre sistema, comprovem que tots estan inclosos en el nostre projecte, per fer-ho fem

Project -> Add/Remove files in Project

S'obrirà una finestra de l'estil de la figura 11.3 des d'on podrem veure els fitxers que formen part del projecte i en cas de faltar-ne algun el podrem afegir. La llista de fitxers ha de ser

- *DW8051_core.edif*
- *ext_RAM_2kB.v*
- *ext_ROM_8kB.v*
- *i_RAM_256.v*
- *sfr_reg.v*
- *power_on_reset.v*
- *top_system_8051.v*

Un cop generats tots els fitxers necessaris procedim a compilar el codi generat.

Processing -> Start Compilation

Veiem que no ens dona cap error. Per tant ja podem procedir a fer l'assignació d'entrades i sortides amb els pins de la FPGA.

Assignments -> Assignment Editor

A la pestanya de categoria escollim PIN i omplim la taula de sota amb les següents correspondències, taula 11.1:

Taula 11.1: Correspondència de pins.

Nom del PIN (To)	Nom al data sheet de la FPGA	Número de Pin (Location)
clk_i	EXT CLK	PIN_P26
rst_n_i	SW0	PIN_N25
t0_o	GPIO0[8] o JP1[8]	PIN_F25
reg80h_o[7]	GPIO0[7] o JP1[7]	PIN_J20
reg80h_o[6]	GPIO0 [6] o JP1[6]	PIN_J21
reg80h_o[5]	GPIO0 [5] o JP1[5]	PIN_F23
reg80h_o[4]	GPIO0 [4] o JP1[4]	PIN_F24
reg80h_o[3]	GPIO0 [3] o JP1[3]	PIN_E25
reg80h_o[2]	GPIO0 [2] o JP1[2]	PIN_E26
reg80h_o[1]	GPIO0 [1] o JP1[1]	PIN_J22
reg80h_o[0]	GPIO0 [0] o JP1[0]	PIN_D25

Tornem a compilar per últim cop el projecte i només queda bolcar el codi a la FPGA. **Tools ->Programmer** seleccionem el **fitxer top_system_8051.sof**, ens assegurem de que està seleccionada la casella **Program/Configure** i de que ens ha detectat la placa pel USB-Blaster, en el cas que no l'hagi detectat ens hem d'assegurar de que els drivers estan instal·lats i un cop instal·lats anar a **Hardware Setup** on podrem escollir **USB-Blaster**.

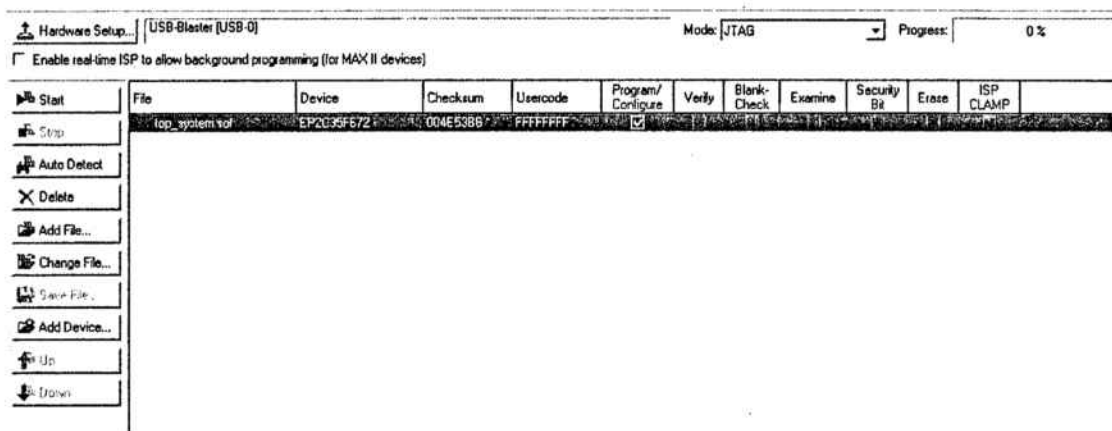


Fig. 11.11: Interfície del *Programmer* de la FPGA

Un cop carregat el programa a la FPGA es qüestió de fer el reset i observar a l'oscil·loscopi els pins del GPIO, per on surten les senyals.

Cal notar que la FPGA s'hi ha de connectar un senyal quadrat de 20MHz de 3.3V, que utilitzarem com a rellotge del nostre sistema. Per generar aquest senyal es necessita un generador de freqüències que ens permeti genera un senyal de 20MHz.

11.3 Keil uVision3

Keil uVision3 és un compilador de codi 'C' i Assembler per microcontroladors de la família ARM, RL-ARM, Cx51 y C166. Dels quals aquest document es centre en la família Cx51.

Primer de tot hem de crear un directori on guardarem el projecte, per exemple,

C:\SysBasic\soft.

Un cop generat el directori de treball ja podem començar a utilitzar el uVision3 per desenvolupar el nostre codi.

1. Creem un nou projecte **Project->New->uVision Project** on seleccionem el directori de treball creat anteriorment, i el nom del projecte per exemple **write_sfr**.
2. Seleccionem el fabricant i el model de processador, que en el nostre cas és **Synopsys** i el model és el **DW8051**. Pitgem a acceptar i se'ns pregunta si volem afegir el codi de **Startup** al nostre projecte, acceptem.
3. Creem un nou fitxer de text **File -> New** on escrivim el següent codi font.

```
#include<reg51.h> //llibreria de registres del 8051

/* BYTE Register */
sfr CKCON = 0x8E; // definim registre del 8051

int i = 0; // variable global inicialitzada a zero

// rutina d'interruptió
void timer0 (void) interrupt 1
{
    i = i + 1; //augmentem la variable
    if(i==0) {P0 = 0x00;}
    if(i==1) {P0 = 0x01;}
    if(i==2) {P0 = 0x02;}
    if(i==3) {P0 = 0x03;}
    if(i==4) {P0 = 0x04;}
    if(i==5) {P0 = 0x10;}
    if(i==6) {P0 = 0x20;}
    if(i==7) {P0 = 0x30;}
    if(i==8) {P0 = 0x40; i = 0;}
}

void main (void)
{
    // inicialitzem el sistema configurem els registres necessaris
    // Timer 0 -- Mode 8-bit counter auto-reload, GATE = 0
    TMOD = 0x02;
    // Enable timer 0 interrupt IE.1 = 1
    IE = 0x82;
    // Timer 0 uses (clk/4 if 0x09), (clk/12 if 0x01). Clk = 10 MHz
    CKCON = 0x01;
    // Disable counting on Timer 0.
    TCON = 0x01;
    TL0 = 0x00;
    TH0 = 0x00;
    // Enable counting on Timer 0
    TCON = 0x11;
    // Initialization SFR 80h
    P0 = 0x00;
    // "bucle" infinit necessari per treballar amb interrupcions
    while(1);
}
```

4. El guardarem amb l'extensió **.c** ja què és codi C. Un cop copiat el codi cal afegir-lo al nostre projecte, tal i com es mostra a la figura 11.12.

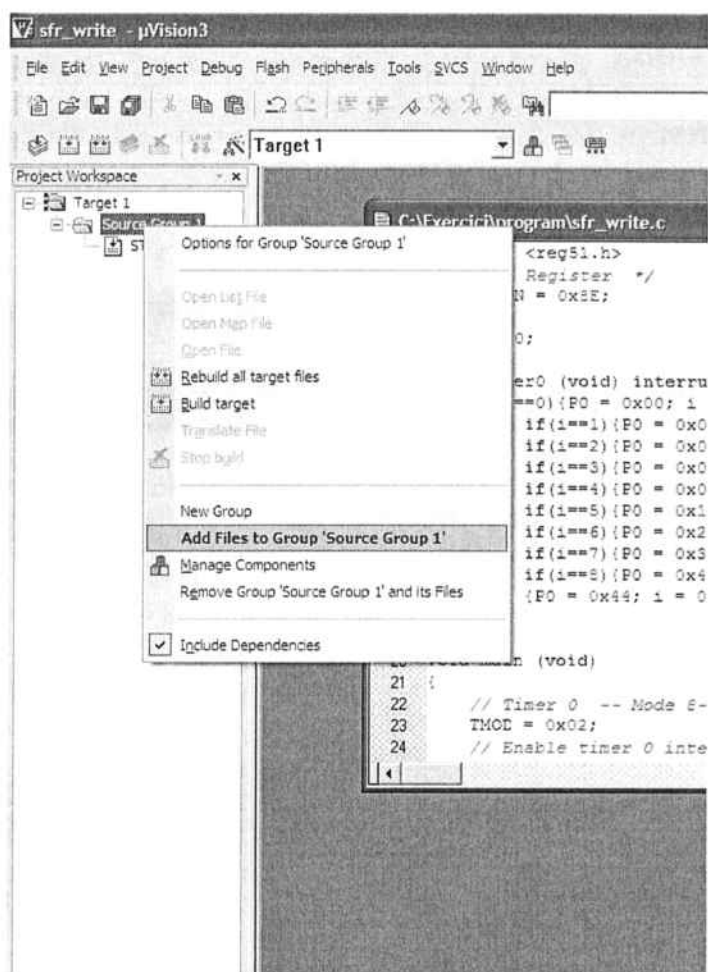


Fig. 11.12: Afegir fitxers al projecte.

5. Definim les opcions de compilació per el nostre projecte. Seleccionem a la finestra del **Project Workspace** el **Target1** i des del menú **Project ->Options for Target 'Target 1'**, s'obre una finestra com la de la figura 11.13, on especifiquem els paràmetres:
 - Freqüència de treball, **20MHz**.
 - Model de memòria, **Small: variables in DATA**.
 - Mida de la memòria de programa, **Small: program 2K or less**.
 - Sistema Operatiu: **None**

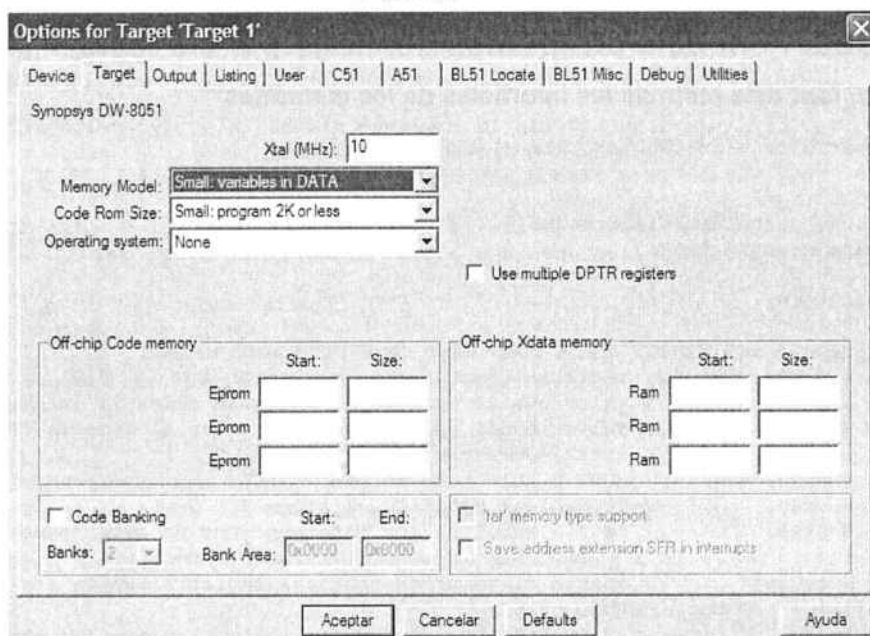


Fig. 11.13: Configuració del model de memòria i freqüència de treball.

6. Pitgem a la **Output**, figura 11.15, on seleccionem la carpeta on volem que vagin el fitxers que es generin, i activem l'opció **Creat HEX file**, que ens generarà un fitxer .hex que contindrà el codi generat llest per carregar a la memòria ROM generada amb el Quartus II. Pitgem Aceptar.

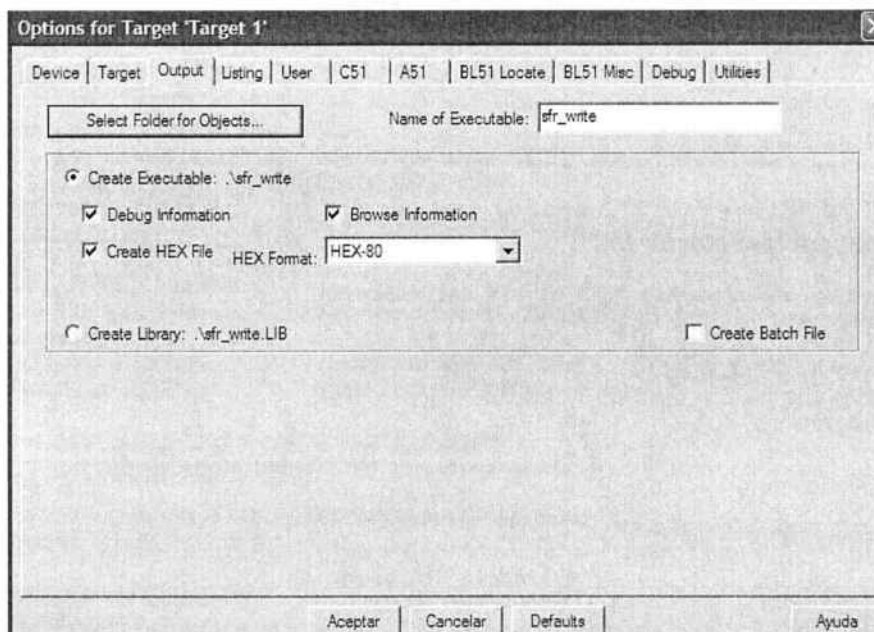


Fig 11.14: Habilitació de la creació del fitxer .hex.

7. Compilem el project a partir del menú **Project->Build Target**.

Per últim, a la **Output window**, s'indica si hi ha hagut algun error i en cas negatiu, ens informa del nombre de Bytes del nostre programa.

11.4 Codi Font dels programes generats per els tests

❖ Codi font dels tests de les interfícies de les memòries

```
// llibreria on s'instancien tots el SFR del sistema
#include "reg51mod.h"

// variable global inicialitzada
char idata count = 0x06;

void main(void)
{
    //Configuració del TIMER1 com a comptador de 8-bits auto-reload
    TL1 = 0x00; // registre comptador inicialitzat a 0
    TH1 = 0x00; // valor que es carrega al TL1 quan salta la interrupció
    CKCON = 0x01; // Selecció del rellotge que utilitza el Timer1 (CKCON.4
                // 1:clk/4 o 0:clk/12).
    TMOD = 0x20; // MODE 2 8bit auto-reload
    TCON = 0x40; // TCON.6 = 1 habilita el timer 1
    IE = 0x88; // IE.7 = Habilita les interrupcions no emmascarades;
            // IE.3 = Habilita la interrupció del timer 1
    SFRGP = count; // escriu el valor de count al registre SFRGP
    while(1){;} // bucle infinit
}

// Rutina d'interrupció del timer 1
void timer1_int (void) interrupt 3
{
    count = count + 1; // augmenta la variable global
    if(count == 0x12)
    {
        SFRGP = count; // escriu el valor de count al registre SFRGP
        count = 0x00; // Posa la variable global a 0
    }
    if(count == 0x05)
    {
        SFRGP = count; // escriu el valor de count al registre SFRGP
    }
}
```

❖ Codi font del test Master I2C

```
// llibreria on s'instancien tots el SFR del sistema
#include "reg51mod.h"

// declaració de constants
#define WRITE_I2C 1
#define READ_I2C 0
char RBIT1 = 0x02;
char RBIT7 = 0x80;

// declaració variables globas
char i2c_wr; // Escull l'acció a realitzar 1: escriu a slave
                // 0:llegeix del slave
char i2c_slave_addr; // Conté l'adreça del slave
char i2c_reg_addr; // Conté la direcció del registre a escriure/llegir
char i2c_wr_data; // Conté la dada a escriure al registre
char i2c_rd_data; // Es on es guarda la dada llegida del slave

// prototip funció
void i2c_ctrl(void);

void main(void)
{
    I2CPRERLO = 0x13; // Configura el clk per un clk_sys de 10MHZ
    I2CPRERHI = 0x00;
    I2CCTRL = 0x80; // Habilita el master I2C

    i2c_slave_addr = 0x58; // Adreça del slave només son els 7 bits més //
                        // significatius
```

```

i2c_reg_addr = 0x9E;          // Addr del registre
i2c_wr = READ_I2C;            // Escull llegir el slave
i2c_ctrl();                   // Crida la funció que controla el master i2c

i2c_wr = WRITE_I2C;           // Escull escriure el slave
i2c_wr_data = 0x33;           // Dada a escriure
i2c_ctrl();                   // Crida la funció que controla el master i2c

i2c_wr = READ_I2C;            // Escull llegir el slave
i2c_ctrl();                   // Crida la funció que controla el master i2c

while(1){;}                   // bucle infinit
}

void i2c_ctrl (void) // Funció que controla el master i2c
{
    if(i2c_wr) // WRITE Slave
    {
        //GENERA START ENVIA ADDR SLAVE
        I2CTXR = i2c_slave_addr; // Put slave addr to TX reg + WRITE bit
        I2CCR = 0x90;             // STA = WR = 1
        while(I2CSR & RBIT1);     // wait transfer is done
        while(I2CSR & RBIT7);     // wait recived ACK form slave

        //ENVIA ADDR DEL REG A ESCRIURE
        I2CTXR = i2c_reg_addr;    // Put reg addr to TX reg
        I2CCR = 0x10;             // WR = 1
        while((I2CSR & RBIT1));    // wait transfer is done
        while(I2CSR & RBIT7);     // wait recived ACK form slave

        //ENVIA DADA A ESCRIURE I GENERA STOP
        I2CTXR = i2c_wr_data;     // Put reg addr to TX reg
        I2CCR = 0x50;             // WR = STO = 1
        while((I2CSR & RBIT1));    // wait transfer is done
        while(I2CSR & RBIT7);     // wait recived ACK form slave
    }
    else // READ SLAVE
    {
        //GENERA START I ENVIA ADDR SLAVE
        I2CTXR = i2c_slave_addr; // Put slave addr to TX reg + WRITE bit
        I2CCR = 0x90;             // STA = WR = 1
        while((I2CSR & RBIT1));    // wait transfer is done
        while(I2CSR & RBIT7);     // wait recived ACK form slave

        //ENVIA L'ADDR A LLEGIR
        I2CTXR = i2c_reg_addr;    // Put reg addr to TX reg
        I2CCR = 0x10;             // WR = 1
        while((I2CSR & RBIT1));    // wait transfer is done
        while(I2CSR & RBIT7);     // wait recived ACK form slave

        //GENERA RESTART I RENVIÀ ADDR SLAVE + READ
        I2CTXR = i2c_slave_addr + 0x01; // Put slave addr to TX reg + READ bit
        I2CCR = 0x90;             // STA = WR = 1
        while((I2CSR & RBIT1));    // wait transfer is done
        while(I2CSR & RBIT7);     // wait recived ACK form slave

        // GENERA EL SENYAL STOP
        I2CCR = 0x60;             // RD = STO = 1; ACK = 0
        while((I2CSR & RBIT1));    // wait transfer is done
        i2c_rd_data = I2CRXR;     // Store data recibed
    }
}

```

❖ Codi font del test de la unitat de control de les comunicacions

```

// llibreria on s'instancien tots el SFR del sistema
#include "regs1mod.h"

void main(void)
{
    // Habilita i configura el clk de 1MHz

```

```

CLK1M      = 0x84;
// Habilita les comunicacions i configura el temps de bit. clk_sys = 10MHz
BITIMERF = 0xD1;
// Habilita i configura la interrupció de les comunicacions
IP        = 0xFF;
TCON      = 0x41;      // TCON.6 = 1 Enable Timer1
IE        = 0x81;      // IE.7 = Habilita les interrupcions no emmascarades
              // IE.9 = Habilita la interrupció int_0 (comunicacions)
while(1){;} // bucle infinit
}

// Rutina d'interrupció de la interrupció int_0 (comunicacions)
void comm_rf (void) interrupt 0
{
    switch(LDATARF)      // Segons el valor de LDATARF escull que fer
    {
        case 0x41:      // Si LDATARF = 0x41
            SFRGP = MDATARF; // Escriu al registre SFRGP el valor rebut en el
              // Byte mes significatiu
            break;

        default:        // En qualsevol altre cas escriu 0xFF al registre
            SFRGP = 0xFF; // SFRGP
            break;
    }
}

```

11.5 Documentació Top test unit

❖ Diagrama de blocs

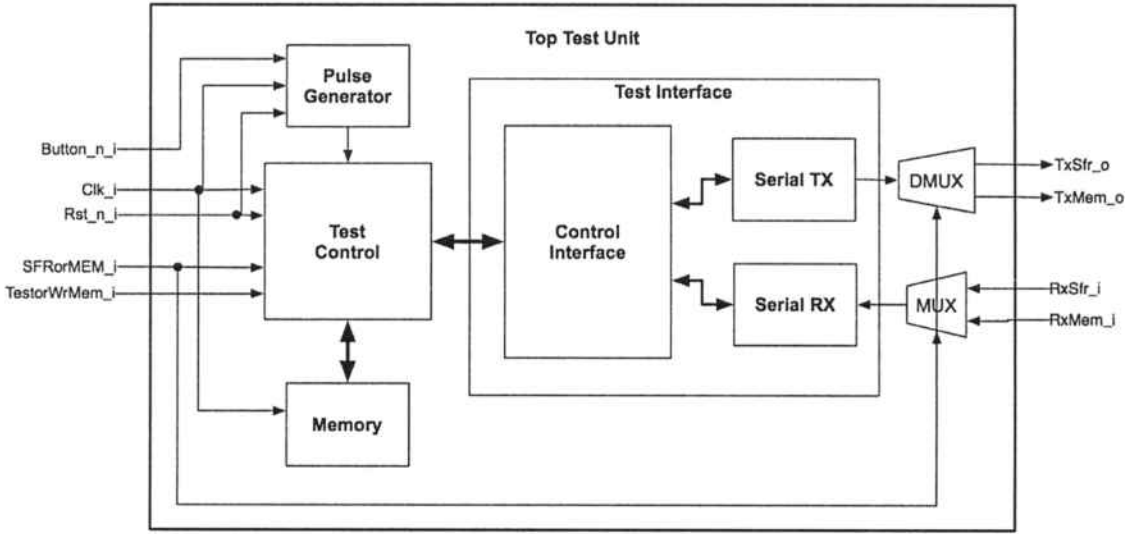


Figura 11.15: Diagrama de blocs de la interfície de test top test unit.

Taula 11.2: Entrades i sortides del top test unit.

Nom	Descripció	I/O	Bits
Clk_i	Relloige del sistema.	I	1
Rst_n_i	Reset del sistema. Actiu nivell baix.	I	1
Button_n_i	Senyal que arriba del botó. Fa que es realitzi una lectura o escriptura o activa el bolcat de memòria. Actiu nivell baix.	I	1
SFRorMEM_i	1: Escriu/llegeix els SFRs. 0: Escriu/llegeix la memòria de programa del xip VECTOR V1.	I	1
TestorWrMem_i	1: Realitza una escriptura o lectura d'una posició de la memòria de programa o de un SFR, depenent de SFRorMEM_i. 0: Fa el bolcat de memòria, quan es pressiona el Button_n_i.	I	1
TxSfr_o	Sortida sèrie quan es realitzen les escriptures o lectures a els SFRs.	O	1
TxMem_o	Sortida sèrie quan es realitzen les escriptures o lectures a memòria.	O	1
RxSfr_i	Entrada sèrie quan es realitzen les escriptures o lectures a SFRs.	I	1
RxMem_i	Entrada sèrie quan es realitzen les lectures a memòria.	I	1

❖ Guia d'utilització

El bloc Top test Unit té 3 modes de funcionament que es seleccionen amb les senyals TestorWrMem_i, SFRorMEM_i.

- **Mode test SFR:** TestorWrMem_i = 1, SFRorMEM_i = 1.

Escriu/llegeix el registre SFR indicat. La memòria memory és una memòria creada amb el Quartus II de 256 posicions de 8 bits cadascuna on es poden carregar fitxers .HEX.

Per realitzar una escriptura, el fitxer .HEX que és carrega a la memòria, ha de contenir el següent.

Adreça	Contingut
00h	Adreça del SFR a escriure.
01h	0x00
02h	Dada a escriure al SFR.
03h	0x02: indica que es realitza una escriptura a SFR.

Sempre que es realitza una escriptura automàticament, es fa una lectura del mateix registre per comprovar que s'ha escrit bé.

Per realitzar una lectura, el fitxer .HEX que es carrega a la memòria, ha de contenir el següent.

Adreça	Contingut
00h	Adreça del SFR a llegir.
01h	0x00
02h	0x00
03h	0x07: indica que es realitza una lectura a SFR.

Cada cop que es pitja el botó es llegeix una posició de la memòria *memory*. Quan s'ha pitjat per 5 cop es realitza l'acció d'escriptura o lectura seguint el protocol. Es pot repetir l'estructura per escriure/llegir varis SFRs

Protocol d'escriptura

Byte 0	Byte 1	Byte 2
01010101	ADDR	DATA

☐ Dades que reben pel *serial*

Protocol de lectura

Byte 0	Byte 1
01010101	ADDR
	11011011
	DATA

☐ Dades que s'envien pel *serial_o*

Byte 0 Byte 1

Figura 11.16: Diagrama temporal per escriure/llegir un SFR.

- **Mode test MEM:** *TestorWrMem_i* = 1, *SFRorMEM_i* = 0.

Escriu/llegeix a la posició de memòria de programa indicada.

Per realitzar una escriptura, el fitxer .HEX que es carrega a la memòria, ha de contenir el següent.

Adreça	Contingut
00h	LSB de l'adreça escriure.
01h	MSB de l'adreça escriure.
02h	Dada a escriure a la memòria.
03h	0x0A: indica que es realitza una escriptura a memòria.

Sempre que es realitza una escriptura automàticament, es fa una lectura de la mateixa posició de memòria per comprovar que s'ha escrit bé.

Per realitzar una lectura, el fitxer .HEX que es carrega a la memòria, ha de contenir el següent.

Adreça	Contingut
00h	LSB de l'adreça escriure.
01h	MSB de l'adreça escriure.
02h	0x00
03h	0x0F: indica que es realitza una lectura a memòria.

Cada vegada que es pitja el botó es llegeix una posició de la memòria *memory*. Quan s'ha pitjat per 5 cop es realitza l'acció d'escriptura o lectura seguint el protocol. Es pot repetir l'estructura per escriure/llegir varies posicions de memòria.

Protocol d'escriptura

Byte 0	Byte 1	Byte 2	Byte 3
01010101	ADDR[7:0]	ADDR[15:8]	DATA

☐ Dades que reben pel *serialmem*

Protocol de lectura

☐ Dades que s'envien pel *serialmem_o*

Byte 0	Byte 1	Byte 2		
01010101	ADDR[7:0]	ADDR[15:8]		
			11011011	DATA
Byte 0	Byte 1			

Figura 11.17: Diagrama temporal per escriure/llegir a/de una posició de la memòria de programa.

- **Mode bolcat memòria:** *TestorWrMem_i = 0, SFRorMEM_i = 0.*

Bolca tot el contingut de la memòria *memory* a la memòria de programa de forma automàtica, seguint el protocol de la figura 11.16. Per tant la memòria *memory* ha de contenir el fitxer .HEX del programa que volem carregar. En aquest cas només cal pitjar el botó un sol cop.

11.6 Documentació Slave I2C

❖ Diagrama de blocs

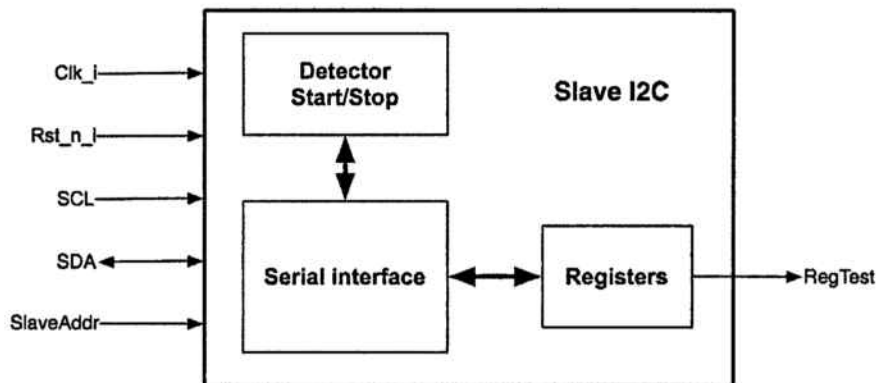


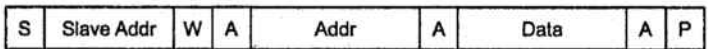
Figura 11.18: Diagrama de blocs del Slave I2C.

Taula 11.4: Entrades i sortides del Slave I2C.

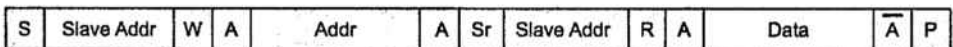
Nom	Descripció	I/O	Bits
Clk_i	Relloige del sistema.	I	1
Rst_n_i	Reset del sistema. Actiu nivell baix.	I	1
SCL	Relloige del bus I2C.	I	1
SDA	Port sèrie per on entre/surten les dades.	I/O	1
RegTest	Mostra el valor del registre 9Eh del slave.	O	8
SlaveAddr	És l'adreça del Slave.	I	7

❖ Guia d'utilització

Protocol d'escriptura



Protocol de lectura



- ☐ Del slave al master
- ☐ Del master al slave
- S = Start condition
- Sr = repeated Start condition
- P = Stop condition
- A = ACK (SDA low)
- $\overline{A}$ = NACK (SDA high)
- W = Write (SDA low)
- R = Read (SDA high)

Figura 11.19: Protocol I2C que segueix el Slave I2C.

Els registres que es poden escriure i llegir són els corresponents a les adreces: 00h, 01h, 02h, 03h, 04h, 9Eh. L'únic que té sortida a l'exterior és el 9Eh.

11.7 Documentació Comm RF

❖ Diagrama de blocs

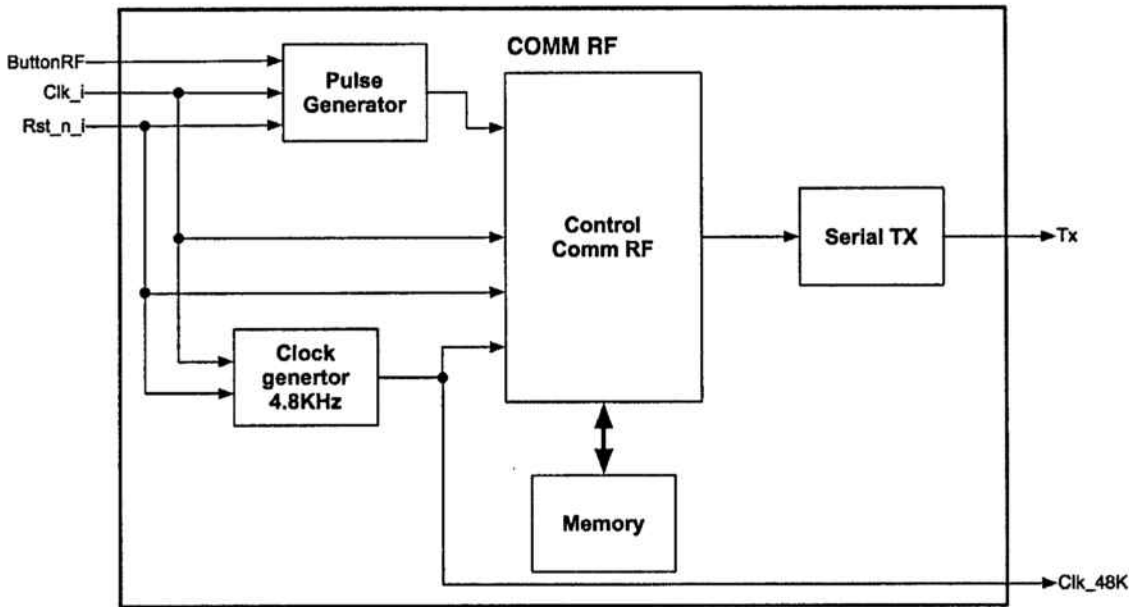


Figura 11.20: Diagrama de blocs del Comm RF.

Taula 11.5: Entrades i sortides del Comm RF.

Nom	Descripció	I/O	Bits
Clk_i	Relotge del sistema.	I	1
Rst_n_i	Reset del sistema. Actiu nivell baix.	I	1
ButtonRF	Senyal que arriba del botó. Fa que hi hagi es llegeixi la memòria i envia el contingut d'una posició per el Tx. Actiu nivell baix.	I	1
Tx	Port sèrie per on s'envien les dades.	O	1
Clk_48K	Relotge que s'utilitza per enviar les dades.	O	1

❖ Guia d'utilització

La memòria *Memory* és una memòria creada amb el *Quartus II* de 256 posicions de 16 bits cadascuna on si poden carregar fitxers .HEX. Cada vegada que es pitja el botó, el bloc *Control Comm RF* llegeix una posició de memòria i envia el seu contingut a través del port sèrie *Tx* seguint el protocol de les comunicacions.

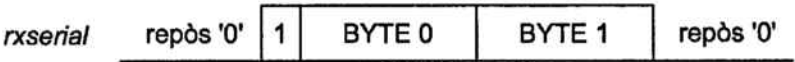


Figura 11.21: Protocol del bloc de comunicacions.

11.8 Codi font del software de control desenvolupat

❖ Fitxer system.c

```
#include <rtx51tny.h>
#include "reg51mod.h"
#include "vars.h"
#include "startup.h"
#include "sfr_config.h"
#include "leds.h"
#include "motors.h"
#include "i2c.h"
#include "interrupts.h"

void system (void) _task_ SYS
{
    while(1)
    {
        // Esperem que arribin ordres per les comm RF
        os_wait (K_SIG,0,0);
        instruction = LDATARF;

        switch(instruction)
        {
            /* I2C INSTRUCTIONS */
            case I2C_WRITE :
                i2c_wr = WRITE;
                i2c_rd_type = RD_I2C_RESTART;
                i2c_slave_addr = MDATARF;
                os_wait (K_SIG,0,0); // wait
                i2c_reg_addr = LDATARF;
                i2c_wr_data = MDATARF;
                os_send_signal (I2C); // send signal to i2c
                os_wait (K_SIG,0,0);
                break;

            case I2C_READ_NORMAL :
                i2c_wr = READ;
                i2c_rd_type = RD_I2C_NORMAL;
                i2c_slave_addr = MDATARF;
                os_wait (K_SIG,0,0); // wait
                i2c_reg_addr = LDATARF;
                os_send_signal (I2C); // send signal to i2c
                os_wait (K_SIG,0,0);
                // TREIEM PEL SFRGP mente no tenim comm rf
                SFRGP = i2c_rd_data;
                // ENVIEM DATA LLEGIDA PEL COMM RF
                // i2c_wr_data = i2c_rd_data;
                // i2c_wr = WRITE;
                // i2c_slave_addr = RFLINK_SLAVE_ADDR;
                // i2c_reg_addr = RFLINK_REG4SEND;
                // os_send_signal (I2C); // send signal to i2c
                // os_wait (K_SIG,0,0);
                break;

            case I2C_READ_RESTART :
                i2c_wr = READ;
                i2c_rd_type = RD_I2C_RESTART;
                i2c_slave_addr = MDATARF;
                os_wait (K_SIG,0,0);
                i2c_reg_addr = LDATARF;
                os_send_signal (I2C); // send signal to i2c
                os_wait (K_SIG,0,0);
                // TREIEM PEL SFRGP mente no tenim comm rf
                SFRGP = i2c_rd_data;
                // ENVIEM DATA LLEGIDA PEL COMM RF
                // i2c_wr_data = i2c_rd_data;
                // i2c_wr = WRITE;
                // i2c_slave_addr = RFLINK_SLAVE_ADDR;
                // i2c_reg_addr = RFLINK_REG4SEND;
                // os_send_signal (I2C); // send signal to i2c
                // os_wait (K_SIG,0,0);
                break;

            /* SFR READ WRITE INSTRUCTIONS */
            case SFR_WRITE :
                sfr_wr = WRITE;
                sfr_addr = MDATARF;
                os_wait (K_SIG,0,0);
                sfr_data = LDATARF;
                os_send_signal (SFR); // send signal to SFR
                os_wait (K_SIG,0,0);
                break;

            case SFR_READ :
                sfr_wr = READ;
                sfr_addr = MDATARF;
                os_send_signal (SFR); // send signal to SFR
                os_wait (K_SIG,0,0);
                // TREIEM PEL SFRGP mente no tenim comm rf
                SFRGP = sfr_data;
                // ENVIEM DATA LLEGIDA PEL COMM RF
                // i2c_wr_data = sfr_data;
                // i2c_wr = WRITE;
```

```

// i2c_slave_addr = RFLINK_SLAVE_ADDR;
// i2c_reg_addr   = RFLINK_REG4SEND;
// os_send_signal (SFR); // send signal to i2c
// os_wait (K_SIG,0,0);
break;

/* CAMERA INSTRUCTIONS */
case CAMERA_WR : // WRITE a camera register
    i2c_wr      = WRITE;
    i2c_rd_type = RD_I2C_RESTART;
    i2c_slave_addr = CAM_SLAVE_ADDR;
    os_wait (K_SIG,0,0);
    i2c_reg_addr = LDATARF;
    i2c_wr_data  = MDATARF;
    os_send_signal (I2C);
    os_wait (K_SIG,0,0);
break;

case CAMERA_RD : // READ a camera register
    i2c_wr      = READ;
    i2c_rd_type = RD_I2C_RESTART;
    i2c_slave_addr = CAM_SLAVE_ADDR;
    i2c_reg_addr = MDATARF;
    os_send_signal (I2C);
    os_wait (K_SIG,0,0);
break;

/* MOTORS INSTRUCTIONS */
case MOTORS_OFF :
    next_motor_run = 0;
    next_direct_ml = 0;
    state_motor1   = IDLE;
break;

case MOTOR1_ON_R :
    next_motor_run = M1;
    next_direct_ml = RIGHT;
    state_motor1   = STARTUP;
break;

case MOTOR1_ON_L :
    next_motor_run = M1;
    next_direct_ml = LEFT;
    state_motor1   = STARTUP;
break;

case MOTOR2_ON_R :
    next_motor_run = M2;
    next_direct_ml = RIGHT;
    state_motor1   = STARTUP;
break;

case MOTOR2_ON_L :
    next_motor_run = M2;
    next_direct_ml = LEFT;
    state_motor1   = STARTUP;
break;

/* LEDS INSTRUCTIONS */
case LEDS_FLASH_NORMAL :
    state_leds      = IDLE;
    leds_period_ticks = MDATARF;
    leds_duty_cycle  = 0x03;
    leds_time_high   = leds_period_ticks / leds_duty_cycle;
    leds_time_high2  = 2 * leds_time_high;
    os_wait (K_SIG,0,0);
    leds_ir_intensity = LDATARF;
    leds_red_intensity = MDATARF;
    os_wait (K_SIG,0,0);
    leds_green_intensity = LDATARF;
    leds_blue_intensity = MDATARF;
    state_leds         = RUN;
break;

case LEDS_FLASH_CUSTOM :
    os_wait (K_SIG,0,0);
    state_leds      = IDLE;
    leds_period_ticks = LDATARF;
    leds_duty_cycle  = MDATARF;
    leds_time_high   = leds_period_ticks / leds_duty_cycle;
    leds_time_high2  = 2 * leds_time_high;
    os_wait (K_SIG,0,0);
    leds_ir_intensity = LDATARF;
    leds_red_intensity = MDATARF;
    os_wait (K_SIG,0,0);
    leds_green_intensity = LDATARF;
    leds_blue_intensity = MDATARF;
    state_leds         = RUN;
break;

case LEDS_IR_ON :
    state_leds      = IDLE;
    leds_ir_intensity = MDATARF;
    leds_instruction = LEDS_IR_ON;
    os_send_signal (LEDS); // send signal to LEDS
    os_wait (K_SIG,0,0);

```

```

break;

case LEDS_RED_ON :
    state_leds          = IDLE;
    leds_red_intensity   = MDATARF;
    leds_instruction     = LEDS_RED_ON;
    os_send_signal (LEDS);
    os_wait (K_SIG,0,0);
break;

case LEDS_GREEN_ON :
    state_leds          = IDLE;
    leds_green_intensity = MDATARF;;
    leds_instruction     = LEDS_GREEN_ON;
    os_send_signal (LEDS);
    os_wait (K_SIG,0,0);
break;

case LEDS_BLUE_ON :
    state_leds          = IDLE;
    leds_blue_intensity  = MDATARF;;
    leds_instruction     = LEDS_BLUE_ON;
    os_send_signal (LEDS);
    os_wait (K_SIG,0,0);
break;

case LEDS_IR_OFF :
    state_leds          = IDLE;
    leds_instruction     = LEDS_IR_OFF;
    os_send_signal (LEDS);
    os_wait (K_SIG,0,0);
break;

case LEDS_RED_OFF :
    state_leds          = IDLE;
    leds_instruction     = LEDS_RED_OFF;
    os_send_signal (LEDS);
    os_wait (K_SIG,0,0);
break;

case LEDS_GREEN_OFF :
    state_leds          = IDLE;
    leds_instruction     = LEDS_GREEN_OFF;
    os_send_signal (LEDS);
    os_wait (K_SIG,0,0);
break;

case LEDS_BLUE_OFF :
    state_leds          = IDLE;
    leds_instruction     = LEDS_BLUE_OFF;
    os_send_signal (LEDS);
    os_wait (K_SIG,0,0);
break;

case LEDS_ALL_ON:
    os_wait (K_SIG,0,0);
    state_leds          = IDLE;
    leds_ir_intensity    = LDATARF;
    leds_red_intensity    = MDATARF;
    os_wait (K_SIG,0,0);
    leds_green_intensity = LDATARF;
    leds_blue_intensity  = MDATARF;
    leds_instruction     = LEDS_ALL_ON;
    os_send_signal (LEDS);
    os_wait (K_SIG,0,0);
break;

case LEDS_ALL_OFF :
    state_leds          = IDLE;
    leds_instruction     = LEDS_ALL_OFF;
    os_send_signal (LEDS);
    os_wait (K_SIG,0,0);
break;

default : break;
}
}

```

❖ Fitxer startup.h

```

void startup_task (void) _task_ INIT
{
    while(1){
        /* System initalitzation */
        // Init Clk generator
        CLK1M      = 0x84;
        //Habilita i configura temps de bit per un rellotge de 10MHz
        BITIMERF   = 0xD1;
        //MOTOR
        MODEM1     = 0x00;
        MODEM2     = 0x00;
        DACM1      = VREF;
        DACM2      = VREF;
    }
}

```

```

EN_CP5V = 0x01;
CTRLM1 = 0x07;
CTRLM2 = 0x07;
// Init I2C
// configurem el clk per un clk_sys de 10MHZ
I2CPRERLO = 0x13;
I2CPRERHI = 0x00;
// habilitem el master
I2CCTRL = 0x80;
// INTERRUPTS CONFIG
TL1 = 0x00;
TH1 = 0x00;
CKCON = 0x01; // Timer 1 uses (depending CKCON.4 1:clk/4 or 0:clk/12).
TMOD = 0x20; // MODE 2 8bit auto-reload
TCN = 0x41; // TCN.6 = 1 Enable Timer1
IE = 0x89; // IE.7 = Enable all unmasked interrupt;
//IE.3 = enable int1 ; IE.0 int_0

// Creat other tasks
os_create_task (SYS); // Create System controller */
os_create_task (I2C); // Create I2C controller */
os_create_task (LEDS); // Create LEDS controller */
os_create_task (MOTORS); // Create MOTORS controller */
os_create_task (SFR); // Create SFR config */
// Delete startup_task
os_delete_task (INIT);
}
}

```

❖ Fitxer sfr_config.h

```

void sfr_ctrl (void) _task_ SFR
{
    while(1)
    {
        os_wait (K_SIG,0,0);
        if(sfr_wr) // WRITE SFR
        {
            switch(sfr_addr)
            {
                /* General propose SFR */
                case 0xED : SFRGP = sfr_data; break;
                /* Motor control Registers */
                case 0x9E : DACM1 = sfr_data; break;
                case 0x9F : DACM2 = sfr_data; break;
                case 0xA0 : CTRLM1 = sfr_data; break;
                case 0xA1 : LTBSM1 = sfr_data; break;
                case 0xA2 : MTBSM1 = sfr_data; break;
                case 0xA3 : LFLIM1 = sfr_data; break;
                case 0xA4 : MPLIM1 = sfr_data; break;
                case 0xA5 : DCCTRL = sfr_data; break;
                case 0xA6 : MODEM1 = sfr_data; break;
                case 0xA7 : MODEM2 = sfr_data; break;
                case 0xA9 : COMBEMP = sfr_data; break;
                case 0xAA : CTRLM2 = sfr_data; break;
                case 0xAB : LTBSM2 = sfr_data; break;
                case 0xAC : MTBSM2 = sfr_data; break;
                case 0xAD : LFLIM2 = sfr_data; break;
                case 0xAE : MPLIM2 = sfr_data; break;
                /* Clks Control Registers */
                case 0xAF : CLK1M = sfr_data; break;
                case 0xB7 : LPER100K = sfr_data; break;
                case 0xB9 : LPER100KH = sfr_data; break;
                case 0xBA : MPER100 = sfr_data; break;
                case 0xBC : CTRLPWM = sfr_data; break;
                /* I2C Registers */
                case 0xB0 : I2CPRERLO = sfr_data; break;
                case 0xB1 : I2CPRERHI = sfr_data; break;
                case 0xB2 : I2CCTRL = sfr_data; break;
                case 0xB3 : I2CTXR = sfr_data; break;
                case 0xB4 : I2CCR = sfr_data; break;
                /* Communication Control Reg */
                case 0xBD : BITIMERF = sfr_data; break;
                case 0xBE : MDATARF = sfr_data; break;
                case 0xBF : LDATARF = sfr_data; break;
                /* LEDS Registers */
                case 0xC2 : LED0 = sfr_data; break;
                case 0xC3 : LED1 = sfr_data; break;
                case 0xC4 : LED2 = sfr_data; break;
                case 0xC5 : LED3 = sfr_data; break;
                case 0xC6 : LED4 = sfr_data; break;
                case 0xC7 : LED5 = sfr_data; break;
                case 0xC9 : LED6 = sfr_data; break;
                case 0xCE : LED7 = sfr_data; break;
                case 0xCF : LED8 = sfr_data; break;
                case 0xD1 : LED9 = sfr_data; break;
                case 0xD2 : LED10 = sfr_data; break;
                case 0xD3 : LED11 = sfr_data; break;
                case 0xD4 : LED12 = sfr_data; break;
                case 0xD5 : LED13 = sfr_data; break;
                case 0xD6 : LED14 = sfr_data; break;
                case 0xD7 : LED15 = sfr_data; break;
                /* Time Stamp Registers */
                case 0xD9 : TSPER1 = sfr_data; break;
                case 0xDA : TSPER2 = sfr_data; break;
            }
        }
    }
}

```

```

        case 0xDB : TSPER3 = sfr_data; break;
        /* BYTE Registers MICRO */
        case 0xE : CKCON = sfr_data; break;
        case 0x88 : TCON = sfr_data; break;
        case 0x89 : TMOD = sfr_data; break;
        case 0x8A : TL0 = sfr_data; break;
        case 0x8B : TL1 = sfr_data; break;
        case 0x8C : TH0 = sfr_data; break;
        case 0x8D : TH1 = sfr_data; break;
        case 0xA8 : IE = sfr_data; break;
        default : break;
    }
}
else // READ SFR
{
    switch(sfr_addr)
    {
        /* General propose SFR */
        case 0xED : sfr_data = SFRGP ; break;
        /* Motor control Registers */
        case 0x9E : sfr_data = DACM1 ; break;
        case 0x9F : sfr_data = DACM2 ; break;
        case 0xA0 : sfr_data = CTRLM1 ; break;
        case 0xA1 : sfr_data = LTBSM1 ; break;
        case 0xA2 : sfr_data = MTBSM1 ; break;
        case 0xA3 : sfr_data = LFLIM1 ; break;
        case 0xA4 : sfr_data = MFLIM1 ; break;
        case 0xA5 : sfr_data = DCCTRL ; break;
        case 0xA6 : sfr_data = MODEM1 ; break;
        case 0xA7 : sfr_data = MODEM2 ; break;
        case 0xA9 : sfr_data = COMBEMF ; break;
        case 0xAA : sfr_data = CTRLM2 ; break;
        case 0xAB : sfr_data = LTBSM2 ; break;
        case 0xAC : sfr_data = MTBSM2 ; break;
        case 0xAD : sfr_data = LFLIM2 ; break;
        case 0xAE : sfr_data = MFLIM2 ; break;
        /* Clks Control Registers */
        case 0xAF : sfr_data = CLK1M ; break;
        case 0xB7 : sfr_data = LPER100K ; break;
        case 0xB9 : sfr_data = LPER100KH ; break;
        case 0xBA : sfr_data = MPER100 ; break;
        case 0xBC : sfr_data = CTRLPWM ; break;
        /* I2C Registers */
        case 0xB0 : sfr_data = I2CPRELO ; break;
        case 0xB1 : sfr_data = I2CPREHI ; break;
        case 0xB2 : sfr_data = I2CCTRL ; break;
        case 0xB3 : sfr_data = I2CTXR ; break;
        case 0xB4 : sfr_data = I2CCR ; break;
        case 0xB5 : sfr_data = I2CRXR ; break;
        case 0xB6 : sfr_data = I2CSR ; break;
        /* Communication Control Reg */
        case 0xBD : sfr_data = BITIMERF ; break;
        case 0xBE : sfr_data = MDATARF ; break;
        case 0xBF : sfr_data = LDATARF ; break;
        /* LEDs Registers */
        case 0xC2 : sfr_data = LED0 ; break;
        case 0xC3 : sfr_data = LED1 ; break;
        case 0xC4 : sfr_data = LED2 ; break;
        case 0xC5 : sfr_data = LED3 ; break;
        case 0xC6 : sfr_data = LED4 ; break;
        case 0xC7 : sfr_data = LED5 ; break;
        case 0xC9 : sfr_data = LED6 ; break;
        case 0xCE : sfr_data = LED7 ; break;
        case 0xCF : sfr_data = LED8 ; break;
        case 0xD1 : sfr_data = LED9 ; break;
        case 0xD2 : sfr_data = LED10 ; break;
        case 0xD3 : sfr_data = LED11 ; break;
        case 0xD4 : sfr_data = LED12 ; break;
        case 0xD5 : sfr_data = LED13 ; break;
        case 0xD6 : sfr_data = LED14 ; break;
        case 0xD7 : sfr_data = LED15 ; break;
        /* Time Stamp Registers */
        case 0xD9 : sfr_data = TSPER1 ; break;
        case 0xDA : sfr_data = TSPER2 ; break;
        case 0xDB : sfr_data = TSPER3 ; break;
        /* BYTE Register MICRO */
        case 0xE : sfr_data = CKCON ; break;
        case 0xD0 : sfr_data = PSW ; break;
        case 0xE0 : sfr_data = ACC ; break;
        case 0xF0 : sfr_data = B ; break;
        case 0x81 : sfr_data = SP ; break;
        case 0x82 : sfr_data = DPL ; break;
        case 0x83 : sfr_data = DPH ; break;
        case 0x87 : sfr_data = PCON ; break;
        case 0x88 : sfr_data = TCON ; break;
        case 0x89 : sfr_data = TMOD ; break;
        case 0x8A : sfr_data = TL0 ; break;
        case 0x8B : sfr_data = TL1 ; break;
        case 0x8C : sfr_data = TH0 ; break;
        case 0x8D : sfr_data = TH1 ; break;
        case 0xA8 : sfr_data = IE ; break;
        case 0xB8 : sfr_data = IP ; break;
        case 0x98 : sfr_data = SCON ; break;
        case 0x99 : sfr_data = SBUF ; break;
        default : break;
    }
}

```

```

    }
    os_send_signal (SYS);
}
}

```

❖ Fitxer leds.h

```

void leds_state (void) _task_LEDS
{
    while(1)
    { // encén o apaga els leds en funció de la instrucció rebuda
      // i amb la intensitat indicada
      os_wait (K_SIG,0,0);
      switch(leds_instruction)
      {
        case LEDES_IR_ON :
          LED0 = leds_ir_intensity;
          LED1 = leds_ir_intensity;
          LED2 = leds_ir_intensity;
          LED3 = leds_ir_intensity;
          break;

        case LEDES_RED_ON :
          LED4 = leds_red_intensity;
          LED5 = leds_red_intensity;
          LED6 = leds_red_intensity;
          LED7 = leds_red_intensity;
          break;

        case LEDES_GREEN_ON :
          LED8 = leds_green_intensity;
          LED9 = leds_green_intensity;
          LED10 = leds_green_intensity;
          LED11 = leds_green_intensity;
          break;

        case LEDES_BLUE_ON :
          LED12 = leds_blue_intensity;
          LED13 = leds_blue_intensity;
          LED14 = leds_blue_intensity;
          LED15 = leds_blue_intensity;
          break;

        case LEDES_IR_OFF :
          LED0 = LED_OFF;
          LED1 = LED_OFF;
          LED2 = LED_OFF;
          LED3 = LED_OFF;
          break;

        case LEDES_RED_OFF :
          LED4 = LED_OFF;
          LED5 = LED_OFF;
          LED6 = LED_OFF;
          LED7 = LED_OFF;
          break;

        case LEDES_GREEN_OFF :
          LED8 = LED_OFF;
          LED9 = LED_OFF;
          LED10 = LED_OFF;
          LED11 = LED_OFF;
          break;

        case LEDES_BLUE_OFF :
          LED12 = LED_OFF;
          LED13 = LED_OFF;
          LED14 = LED_OFF;
          LED15 = LED_OFF;
          break;

        case LEDES_ALL_ON :
          // infrared leds
          LED0 = leds_ir_intensity;
          LED1 = leds_ir_intensity;
          LED2 = leds_ir_intensity;
          LED3 = leds_ir_intensity;
          // red leds
          LED4 = leds_red_intensity;
          LED5 = leds_red_intensity;
          LED6 = leds_red_intensity;
          LED7 = leds_red_intensity;
          // green leds
          LED8 = leds_green_intensity;
          LED9 = leds_green_intensity;
          LED10 = leds_green_intensity;
          LED11 = leds_green_intensity;
          // blue leds
          LED12 = leds_blue_intensity;
          LED13 = leds_blue_intensity;
          LED14 = leds_blue_intensity;
          LED15 = leds_blue_intensity;
          break;
      }
    }
}

```

```

    case LEDS_ALL_OFF :
        // infrared leds
        LED0 = LED_OFF;
        LED1 = LED_OFF;
        LED2 = LED_OFF;
        LED3 = LED_OFF;
        // red leds
        LED4 = LED_OFF;
        LED5 = LED_OFF;
        LED6 = LED_OFF;
        LED7 = LED_OFF;
        // green leds
        LED8 = LED_OFF;
        LED9 = LED_OFF;
        LED10 = LED_OFF;
        LED11 = LED_OFF;
        // blue leds
        LED12 = LED_OFF;
        LED13 = LED_OFF;
        LED14 = LED_OFF;
        LED15 = LED_OFF;
        break;

    default : break;
}
os_send_signal (SYS);
}
}

```

❖ Fitxer motors.h

```

void motors_ctrl (void) _task_ MOTORS
{
    while(1)
    {
        if(state_motor1 != IDLE)
        {
            if(next_direct_m1 != direct_m1 || next_motor_run != motor_run)
            {
                direct_m1 = next_direct_m1;
                motor_run = next_motor_run;
            }

            // MOTOR START-UP
            if(state_motor1 == STARTUP)
            {
                if(time_motor_1 == TICKS_M1)
                {
                    time_motor_1 = 0;
                    startup_count_m1 = startup_count_m1 + 1;
                    if(direct_m1 == RIGHT)
                    {
                        curr_step_m1 = curr_step_m1 + 1;
                        if(curr_step_m1 == 7) curr_step_m1 = S1;
                    }
                    else
                    {
                        curr_step_m1--;
                        if(curr_step_m1 == 0) curr_step_m1 = S6;
                        if(curr_step_m1 == -1) curr_step_m1 = S6;
                    }
                    if(startup_count_m1 == STARTUP_TICKS)
                    {
                        if(direct_m1 == RIGHT) {curr_step_m1 = S3;}
                        else {curr_step_m1 = S4;}
                        state_motor1 = RUN;
                        startup_count_m1 = 0;
                    }
                }
            }

            // MOTOR RUNS..
            if(state_motor1 == RUN)
            {
                // READ the comparators signals
                if(motor_run == M1)
                {
                    bemf_m14 = COMBEMF & RBIT3;
                    bemf_m25 = COMBEMF & RBIT4;
                    bemf_m36 = COMBEMF & RBIT2;
                }
                if(motor_run == M2)
                {
                    bemf_m14 = COMBEMF & RBIT0;
                    bemf_m25 = COMBEMF & RBIT1;
                    bemf_m36 = COMBEMF & RBIT5;
                }
                if(curr_step_m1 == S1 || curr_step_m1 == S3 || curr_step_m1 == S5)
                {
                    //BEMF transition HIGH to LOW if motor turns right or LOW to HIGH
                    //if motor turns left
                    if(direct_m1 == RIGHT)
                    {
                        if(bemf_m14 == LOW && curr_step_m1 == S1)
                        {curr_step_m1++;}
                        else if(bemf_m36 == LOW && curr_step_m1 == S3)

```

```

        {curr_step_m1++;}
    else if(bemf_m25 == LOW && curr_step_m1 == S5)
    {curr_step_m1++; }
}
else
{
    if(motor_run == M1)
    {
        if(bemf_m14 == 0x08 && curr_step_m1 == S1)
        {
            curr_step_m1--;
            if(curr_step_m1 == 0) curr_step_m1 = S6;
        }
        else if(bemf_m36 == 0x04 && curr_step_m1 == S3)
        { curr_step_m1--;}
        else if(bemf_m25 == 0x10 && curr_step_m1 == S5)
        { curr_step_m1--;}
    }
    if(motor_run == M2)
    {
        if(bemf_m14 == 0x01 && curr_step_m1 == S1)
        {
            curr_step_m1--;
            if(curr_step_m1 == 0) curr_step_m1 = S6;
        }
        else if(bemf_m36 == 0x20 && curr_step_m1 == S3)
        { curr_step_m1--;}
        else if(bemf_m25 == 0x02 && curr_step_m1 == S5)
        { curr_step_m1--;}
    }
}
}
else
{
    //BEMF transition LOW to HIGH if motor turns
    // right or HIGH to LOW if motor turns left
    if(direct_m1 == RIGHT)
    {
        if(motor_run == M1)
        {
            if(bemf_m14 == 0x08 && curr_step_m1 == S4)
            {curr_step_m1++;}
            else if(bemf_m36 == 0x04 && curr_step_m1 == S6)
            {
                curr_step_m1++;
                if(curr_step_m1 == 0x07) curr_step_m1 = S1;
            }
            else if(bemf_m25 == 0x10 && curr_step_m1 == S2)
            { curr_step_m1++;}
        }
        if(motor_run == M2)
        {
            if(bemf_m14 == 0x01 && curr_step_m1 == S4)
            {curr_step_m1++;}
            else if(bemf_m36 == 0x20 && curr_step_m1 == S6)
            {
                curr_step_m1++;
                if(curr_step_m1 == 0x07) curr_step_m1 = S1;
            }
            else if(bemf_m25 == 0x02 && curr_step_m1 == S2)
            {curr_step_m1++;}
        }
    }
    else
    {
        if(bemf_m14 == LOW && curr_step_m1 == S4)
        {curr_step_m1--;}
        else if(bemf_m36 == LOW && curr_step_m1 == S6)
        {curr_step_m1--;}
        else if(bemf_m25 == LOW && curr_step_m1 == S2)
        {curr_step_m1--;}
    }
}
}

// Change control signals
if(motor_run == M1)
{
    // WLow = 0; WHigh = 0; VLow = 1; VHigh = 0; ULow = 0; UHigh = 1
    if(curr_step_m1 == S1){MODEM1 = STEP1; MODEM2 = STEPbis;}
    // WLow = 1; WHigh = 0; VLow = 0; VHigh = 0; ULow = 0; UHigh = 1
    else if(curr_step_m1 == S2){MODEM1 = STEP2; MODEM2 = STEPbis;}
    // WLow = 1; WHigh = 0; VLow = 0; VHigh = 1; ULow = 0; UHigh = 0
    else if(curr_step_m1 == S3){MODEM1 = STEP3; MODEM2 = STOP;}
    // WLow = 0; WHigh = 0; VLow = 0; VHigh = 1; ULow = 1; UHigh = 0
    else if(curr_step_m1 == S4){MODEM1 = STEP4; MODEM2 = STEP5;}
    // WLow = 0; WHigh = 1; VLow = 0; VHigh = 0; ULow = 1; UHigh = 0
    else if(curr_step_m1 == S5){MODEM1 = STEP5; MODEM2 = STEP5;}
    // WLow = 0; WHigh = 1; VLow = 1; VHigh = 0; ULow = 0; UHigh = 0
    else if(curr_step_m1 == S6){MODEM1 = STEP6; MODEM2 = STOP;}
    else
    {MODEM1 = STOP; MODEM2 = STOP;}
}
if(motor_run == M2)
{
    // WLow = 0; WHigh = 0; VLow = 1; VHigh = 0; ULow = 0; UHigh = 1;}
    if(curr_step_m1 == S1){MODEM2 = STEP1; MODEM1 = STEPbis;}
    // WLow = 1; WHigh = 0; VLow = 0; VHigh = 0; ULow = 0; UHigh = 1;}
    else if(curr_step_m1 == S2){MODEM2 = STEP2; MODEM1 = STEPbis;}
    // WLow = 1; WHigh = 0; VLow = 0; VHigh = 1; ULow = 0; UHigh = 0;}
}

```

```

        else if(curr_step_ml == S3){MODEM2 = STEP3; MODEM1 = STOP;}
        // WLow = 0; WHigh = 0; VLow = 0; VHigh = 1; ULow = 1; UHigh = 0;}
        else if(curr_step_ml == S4){MODEM2 = STEP4; MODEM1 = STEP5;}
        // WLow = 0; WHigh = 1; VLow = 0; VHigh = 0; ULow = 1; UHigh = 0;}
        else if(curr_step_ml == S5){MODEM2 = STEP5; MODEM1 = STEP5;}
        // WLow = 0; WHigh = 1; VLow = 1; VHigh = 0; ULow = 0; UHigh = 0;}
        else if(curr_step_ml == S6){MODEM2 = STEP6; MODEM1 = STOP;}
        else
            {MODEM1 = STOP; MODEM2 = STOP;}
    }
}
else
{
    curr_step_ml = S0;
}
}
}

```

❖ Fitxer i2c.h

```

void i2c_ctrl (void) _task_ I2C
{
    while(1)
    {
        os_wait (K_SIG,0,0);
        if(i2c_wr) // WRITE Slave
        {
            //ENVIA ADDR SLAVE
            I2CTXR = i2c_slave_addr;
            I2CCR = 0x90;
            while(I2CSR & RBIT1);
            while(I2CSR & RBIT7);
            //ENVIA ADDR DEL REG A ESCRIURE
            I2CTXR = i2c_reg_addr;
            I2CCR = 0x10;
            while((I2CSR & RBIT1));
            while(I2CSR & RBIT7);
            //ENVIA DADA A ESCRIURE
            I2CTXR = i2c_wr_data;
            I2CCR = 0x50;
            while((I2CSR & RBIT1));
            while(I2CSR & RBIT7);
        }
        else // READ SLAVE
        {
            switch(i2c_rd_type)
            {
                case RD_I2C_NORMAL:
                    //ENVIA ADDR SLAVE + READ
                    I2CTXR = i2c_slave_addr + 0x01;
                    I2CCR = 0x90;
                    while((I2CSR & RBIT1));
                    while(I2CSR & RBIT7);
                    //ENVIA L'ADDR A LLEGIR
                    I2CTXR = i2c_reg_addr;
                    I2CCR = 0x10;
                    while((I2CSR & RBIT1));
                    while(I2CSR & RBIT7);
                    //REP DADA DEL REG
                    I2CCR = 0x60;
                    while((I2CSR & RBIT1));
                    i2c_rd_data = I2CRXR;
                    break;

                case RD_I2C_RESTART:
                    //ENVIA ADDR SLAVE
                    I2CTXR = i2c_slave_addr;
                    I2CCR = 0x90;
                    while((I2CSR & RBIT1));
                    while(I2CSR & RBIT7);
                    //ENVIA L'ADDR A LLEGIR
                    I2CTXR = i2c_reg_addr;
                    I2CCR = 0x10;
                    while((I2CSR & RBIT1));
                    while(I2CSR & RBIT7);
                    // EL RESTART I RENVIA ADDR SLAVE + READ
                    I2CTXR = i2c_slave_addr + 0x01;
                    I2CCR = 0x90;
                    while((I2CSR & RBIT1));
                    while(I2CSR & RBIT7);
                    //REP DADA DEL REG
                    I2CCR = 0x60;
                    while((I2CSR & RBIT1)); // wait transfer is done
                    i2c_rd_data = I2CRXR;
                    break;

                default: break;
            }
        }
        os_send_signal (SYS);
    }
}

```

❖ Fitxer interrupts.h

```
void comm_rf (void) interrupt 0 using 2
{
    isr_send_signal (SYS);
}

void timer1_int (void) interrupt 3
{
    if(state_motor1 == STARTUP) {time_motor_1 = time_motor_1 + 1;}
    else {time_motor_1 = 0;}

    if(state_leds == RUN)
    {
        // Flas mode
        if( time_leds == 0 || time_leds == leds_period_ticks )
        {
            // red leds
            LED4 = leds_red_intensity;
            LED5 = leds_red_intensity;
            LED6 = leds_red_intensity;
            LED7 = leds_red_intensity;
            // green leds
            LED8 = LED_OFF;
            LED9 = LED_OFF;
            LED10 = LED_OFF;
            LED11 = LED_OFF;
            // blue leds
            LED12 = LED_OFF;
            LED13 = LED_OFF;
            LED14 = LED_OFF;
            LED15 = LED_OFF;
            // posa la variable temporal a 0
            time_leds = 0;
        }
        if( time_leds == leds_time_high )
        {
            // red leds
            LED4 = LED_OFF;
            LED5 = LED_OFF;
            LED6 = LED_OFF;
            LED7 = LED_OFF;
            // green leds
            LED8 = leds_green_intensity;
            LED9 = leds_green_intensity;
            LED10 = leds_green_intensity;
            LED11 = leds_green_intensity;
            // blue leds
            LED12 = LED_OFF;
            LED13 = LED_OFF;
            LED14 = LED_OFF;
            LED15 = LED_OFF;
        }
        if( time_leds == leds_time_high2 )
        {
            // red leds
            LED4 = LED_OFF;
            LED5 = LED_OFF;
            LED6 = LED_OFF;
            LED7 = LED_OFF;
            // green leds
            LED8 = LED_OFF;
            LED9 = LED_OFF;
            LED10 = LED_OFF;
            LED11 = LED_OFF;
            // blue leds
            LED12 = leds_blue_intensity;
            LED13 = leds_blue_intensity;
            LED14 = leds_blue_intensity;
            LED15 = leds_blue_intensity;
        }
        time_leds = time_leds + 1; //Agumenta referencia temporal
    }
    else{time_leds = 0;}
}
```

❖ Fitxer vars.h

```
/******
VARIABLES INITIALIZATION FUNCTIONS
*****/
// TASKS DEF
#define INIT      0          /* task number of task:  start up      */
#define SYS       1          /* task number of task:  command sys   */
#define I2C       2          /* task number of task:  i2c control   */
#define SFR       3          /* task number of task:  SFR control   */
#define LEDS      4          /* task number of task:  LEDS control  */
#define MOTORS    5          /* task number of task:  MOTORS control */

// DEFINES INSTRUCC
#define NO_OPERATION 0x00

#define CAMERA_WR    0x01
#define CAMERA_RD    0x02
```

```

#define MOTORS_OFF          0x11
#define MOTOR1_ON_R         0x12
#define MOTOR1_ON_L         0x13
#define MOTOR2_ON_R         0x14
#define MOTOR2_ON_L         0x15

#define LEDS_FLASH_NORMAL 0x21
#define LEDS_FLASH_CUSTOM 0x22
#define LEDS_IR_ON         0x23
#define LEDS_RED_ON         0x24
#define LEDS_GREEN_ON       0x25
#define LEDS_BLUE_ON        0x26
#define LEDS_IR_OFF         0x27
#define LEDS_RED_OFF        0x28
#define LEDS_GREEN_OFF      0x29
#define LEDS_BLUE_OFF       0x2A
#define LEDS_ALL_ON         0x2B
#define LEDS_ALL_OFF        0x2C

#define SFR_WRITE           0x41
#define SFR_READ            0x42

#define I2C_WRITE            0x43
#define I2C_READ_NORMAL     0x44
#define I2C_READ_RESTART   0x45

// masks for write 1 on bit position (with | operator)
// masks for read one bit position with & operator
unsigned char code RBIT0 = 0x01;
unsigned char code RBIT1 = 0x02;
unsigned char code RBIT2 = 0x04;
unsigned char code RBIT3 = 0x08;
unsigned char code RBIT4 = 0x10;
unsigned char code RBIT5 = 0x20;
unsigned char code RBIT6 = 0x40;
unsigned char code RBIT7 = 0x80;
// Masks for write 0 on bit position (with & operator)
unsigned char code WBIT0 = 0xFE;
unsigned char code WBIT1 = 0xFD;
unsigned char code WBIT2 = 0xFB;
unsigned char code WBIT3 = 0xF7;
unsigned char code WBIT4 = 0xEF;
unsigned char code WBIT5 = 0xDF;
unsigned char code WBIT6 = 0xBF;
unsigned char code WBIT7 = 0x7F;

char instruction = 0x00;

/*****
MOTOR
*****/
// CONSTANTS
#define HIGH 0x01
#define LOW 0x00
#define ON 0x01
#define OFF 0x00
#define RIGHT 0x01
#define LEFT 0x00
#define M1 0x01
#define M2 0x02

// Step of motor movement
#define STOP 0x00 // motor stoped
#define STEP1 0xC4 // UV -->W(H-L)
#define STEP2 0xC1 // UW -->V(L-H)
#define STEP3 0xD1 // VW -->U(H-L)
#define STEP4 0xD0 // VU -->W(L-H)
#define STEP5 0xC8 // WU -->V(H-L)
#define STEP6 0xCC // WV -->U(L-H)
#define STEPbis 0xE0

#define S0 0x00 // motor stoped
#define S1 0x01 // UV -->W(H-L)
#define S2 0x02 // UW -->V(L-H)
#define S3 0x03 // VW -->U(H-L)
#define S4 0x04 // VU -->W(L-H)
#define S5 0x05 // WU -->V(H-L)
#define S6 0x06 // WV -->U(L-H)

// Tensio referencia
#define VREF 0x7F

// States motor
#define IDLE 0x00
#define STARTUP 0x01
#define RUN 0x02

// temps motor
#define TICKS_M1 10
#define STARTUP_TICKS 26

// MOTOR VARIABLES-----
char time_motor_1 = 0; // Time reference for each motor
char startup_count_m1 = 0; // the startup count variable of the motors
char curr_step_m1 = S0; // stores current step of the motors

```

```

char motor_run      = M1; // indicates the motor is running now
char next_motor_run = M1;
char direct_m1      = RIGHT; // indicates de direction of rotation
char next_direct_m1 = RIGHT;
char state_motor1   = IDLE; // indicates the state of the motor

char bemf_m14       = 0; // stores BEMF of free fase
char bemf_m25       = 0;
char bemf_m36       = 0;

/*****
                                LEDS
*****/
#define LED_OFF 0x00

char state_leds      = IDLE;
char leds_instruction = 0;
// control de time refrence
unsigned char time_leds = 0;
// duty cylce of led = Time signal is high / Time signal is low
unsigned char leds_time_high = 0;
unsigned char leds_time_high2 = 0;
unsigned char leds_duty_cycle = 0;
// contains de number of ticks that corresponds a Tcoms = 33ms
unsigned char leds_period_ticks = 0;

char leds_ir_intensity = 0;
char leds_red_intensity = 0;
char leds_green_intensity = 0;
char leds_blue_intensity = 0;

/*****
                                SFR & I2C
*****/
// SFR VARIABLES
char sfr_wr = 0;
char sfr_addr = 0;
char sfr_data = 0;

// I2C CONSANTS
#define WRITE 1
#define READ 0
//types of RD
#define RD_I2C_NORMAL 1
#define RD_I2C_RESTART 2

// Slave Addr
#define RFLINK_SLAVE_ADDR 0xFF // A DEFINIR
#define RFLINK_REG4SEND 0xCC // A DEFINIR
#define CAM_SLAVE_ADDR 0x40 // SIGN = 1 => addr = 0x50
// SIGN = 0 => addr = 0x40

// I2C VARIABLES-----
char i2c_wr = 0; //1: write, 0: read
char i2c_slave_addr = 0;
char i2c_reg_addr = 0;
char i2c_wr_data = 0;
char i2c_rd_data = 0;
char i2c_rd_type = 0;

```

❖ Fitxer reg51mod.h

```

/-----
REG51MOD.H modiflicated

Header file for generic 80C51 and 80C31 microcontroller.
Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software, Inc.
All rights reserved.
-----*/

#ifndef __REG51_H__
#define __REG51_H__

/* General propose SFR */
sfr SFRGP = 0xED;

/* Motor control Regiters */
sfr DACM1 = 0x9E;
sfr DACM2 = 0x9F;
sfr CTRLM1 = 0xA0;
sbit START_M1 = CTRLM1^0;
sbit TURN_M1 = CTRLM1^1;
sbit EN_CP5V = CTRLM1^2;
sfr LTBSM1 = 0xA1;
sfr MTBSM1 = 0xA2;
sfr LPLIM1 = 0xA3;
sfr MPLIM1 = 0xA4;
sfr DCCTRL = 0xA5;
sfr MODEM1 = 0xA6;
sfr MODEM2 = 0xA7;
sfr COMBEMF = 0xA9;
sfr CTRLM2 = 0xAA;
sfr LTBSM2 = 0xAB;
sfr MTBSM2 = 0xAC;

```

```

sfr LFLIM2 = 0xAD;
sfr MFLIM2 = 0xAE;

/* Clks Control Registers */
sfr CLKLM = 0xAF;
sfr LPER100K = 0xB7;
sfr LPER100KH = 0xB9;
sfr MPER100 = 0xBA;
sfr CTRLPWM = 0xBC;

/* I2C Registers */
sfr I2CPRERLO = 0xB0;
sfr I2CPRERHI = 0xB1;
sfr I2CCTRL = 0xB2;
sfr I2CTXR = 0xB3;
sfr I2CCR = 0xB4;
sfr I2CRXR = 0xB5;
sfr I2CSR = 0xB6;

/* Communication Control Reg */
sfr BITIMERF = 0xBD;
sfr MDATAF = 0xBE;
sfr LDATF = 0xBF;

/* LEDs Registers */
sfr LED0 = 0xC2;
sfr LED1 = 0xC3;
sfr LED2 = 0xC4;
sfr LED3 = 0xC5;
sfr LED4 = 0xC6;
sfr LED5 = 0xC7;
sfr LED6 = 0xC9;
sfr LED7 = 0xCE;
sfr LED8 = 0xCF;
sfr LED9 = 0xD1;
sfr LED10 = 0xD2;
sfr LED11 = 0xD3;
sfr LED12 = 0xD4;
sfr LED13 = 0xD5;
sfr LED14 = 0xD6;
sfr LED15 = 0xD7;

/* Time Stamp Registers */
sfr TSPER1 = 0xD9;
sfr TSPER2 = 0xDA;
sfr TSPER3 = 0xDB;

/* BYTE Register MICRO */
sfr CKCON = 0xE8;
sfr PSW = 0xD0;
sfr ACC = 0xE0;
sfr B = 0xF0;
sfr SP = 0x81;
sfr DPL = 0x82;
sfr DPH = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0 = 0x8A;
sfr TL1 = 0x8B;
sfr TH0 = 0x8C;
sfr TH1 = 0x8D;
sfr IE = 0xA8;
sfr IP = 0xB8;
sfr SCON = 0x98;
sfr SBUP = 0x99;

/* BIT Register */
/* PSW */
sbit CY = 0xD7;
sbit AC = 0xD6;
sbit F0 = 0xD5;
sbit RS1 = 0xD4;
sbit RS0 = 0xD3;
sbit OV = 0xD2;
sbit P = 0xD0;

/* TCON */
sbit TF1 = 0x8F;
sbit TR1 = 0x8E;
sbit TF0 = 0x8D;
sbit TR0 = 0x8C;
sbit IE1 = 0x8B;
sbit IT1 = 0x8A;
sbit IE0 = 0x89;
sbit IT0 = 0x88;

/* IE */
sbit EA = 0xAF;
sbit ES = 0xAC;
sbit ET1 = 0xAB;
sbit EX1 = 0xAA;
sbit ET0 = 0xA9;
sbit EX0 = 0xA8;

/* IP */

```

```

sbit PS    = 0xBC;
sbit PT1   = 0xBB;
sbit PX1   = 0xBA;
sbit PT0   = 0xB9;
sbit PX0   = 0xB8;

```

```

/* P3 */
sbit RD    = 0xB7;
sbit WR    = 0xB6;
sbit T1    = 0xB5;
sbit T0    = 0xB4;
sbit INT1  = 0xB3;
sbit INT0  = 0xB2;
sbit TXD   = 0xB1;
sbit RXD   = 0xB0;

```

```

/* SCON */
sbit SM0   = 0x9F;
sbit SM1   = 0x9E;
sbit SM2   = 0x9D;
sbit REN   = 0x9C;
sbit TB8   = 0x9B;
sbit RB8   = 0x9A;
sbit TI    = 0x99;
sbit RI    = 0x98;

```

```

#endif

```