



# 3915-1: Optimització del Procés d'Arrencada d'un Sistema Multiprocessador

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Hugo Lluelles Martínez i dirigit per Borja Martínez Huerta.

Bellaterra, Febrer de 2012

El firmant, Borja Martínez Huerta , professor del Departament de Microelectrònica i Sistemes Electrònics de la Universitat Autònoma de Barcelona

CERTIFICA:

Que la present memòria ha sigut realitzada sota la seva direcció per Hugo Lluelles Martínez

Bellaterra, Febrer de 2012

---

Firmat: Borja Martínez Huerta

# Índex

# Índex de figures

# Capítol 1

# Introducció

## 1.1 Marc del Projecte

Un sistema encastat és un sistema de computació amb una funció molt específica. A diferència d'un PC en el que es poden executar infinitat de programes diferents, en la majoria d'ocasions, un sistema encastat està pensat per a que només executi un únic programa durant tota la seva vida útil. Això fa que no calgui un dispositiu d'emmagatzematge per al codi d'aplicació, ni gaire gran, ni fet pensant en que s'hi han de canviar dades molt sovint (com és el cas d'un disc dur en un PC, al que s'hi accedeix gairebé constantment), sinó que amb una memòria no volàtil integrada a la placa ja n'hi ha prou per a desar-hi el programa i per a que l'element de processament vagi a buscar el codi que ha d'executar un cop s'hagi alimentat el sistema.

Una altra de les diferències principals amb un PC, és que en un sistema encastat es prioritza el preu i el consum molt per sobre del rendiment. Això és perquè els sistemes encastats tenen un preu de fabricació molt més baix, i qualsevol petit estalvi en algun component suposa un percentatge de baixada molt important en el preu de fabricació.

Aquest projecte intenta donar un pas més enllà en la optimització del cost associat a la memòria no volàtil d'arrencada, que com es veurà més endavant, en alguns casos particulars ni tan sols és necessària.

## 1.2 Motivació i Plantejament del Problema

Cada cop més sovint, els sistemes encastats estan tendint a tenir arquitectures amb diferents elements de còmput (ja siguin microcontroladors, microprocessadors, FPGAs, DSPs, etc.) amb tasques específiques, i cadascun d'aquests elements necessita tenir una memòria no volàtil associada, com ara una EEPROM o una Flash, on hi tingui emmagatzemat el codi que ha d'executar per iniciar el procés d'arrencada. En general, aquests dispositius no poden compartir la mateixa memòria, ja sigui perquè necessiten interfícies diferents per a comunicar-s'hi, o perquè, tot i fer servir la mateixa interfície, els dos dispositius van a buscar el codi d'arrencada a la mateixa posició de memòria (típicament a la posició 0).

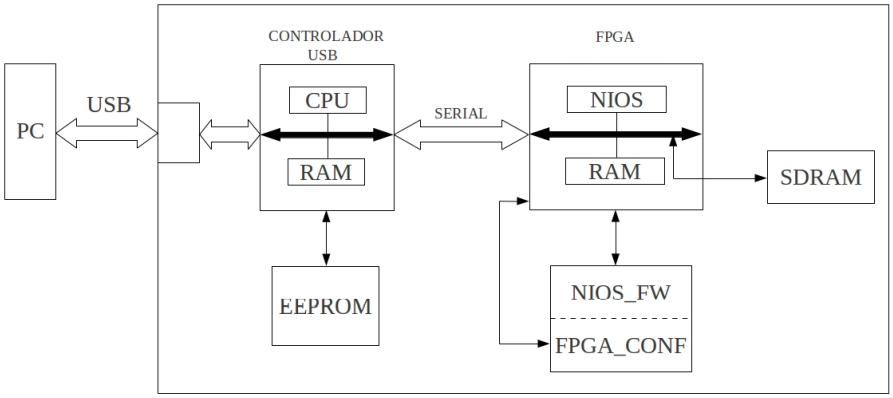


Figura 1.1: Exemple de Sistema Encastat Multiprocessador

El sistema representat a la figura 1.1 és un exemple de sistema multiprocessador encastat. Aquest sistema està format per un controlador d'USB basat en un processador de 8 bits, i una FPGA que conté un SoftCore, que és un processador implementat amb un llenguatge de descripció de Hardware.

El xip del controlador USB només conté memòria volàtil integrada, cosa que fa que sigui necessària una memòria no volàtil externa (normalment una EEPROM) que s'ha de posar a la placa per allotjar el codi del programa que haurà d'executar.

Per part de la FPGA, ens trobem amb que el xip només conté elements pro-

gramables i que inicialment està "buida". Un cop alimentada, una FPGA va a buscar la informació necessària per a carregar la seva configuració a alguna memòria no volàtil que hi tingui connectada (típicament una Flash). A més, en el cas d'un SoftCore, el Firmware pot anar implícit amb la informació de configuració si s'ha d'executar el codi des de la memòria interna de la FPGA. El problema és que és una memòria molt limitada per la mida de la FPGA, per tant és aconsellable emmagatzemar el Firmware o bé en una altra memòria no volàtil, o bé reutilitzant la mateixa memòria on es guarda la configuració de la FPGA.

Per tant, en un sistema multiprocessador hi trobem múltiples memòries, i de diferents tipus segons la seva funcionalitat, i aquestes memòries, en el cas particular de que el sistema encastat funcioni sempre connectat a un PC via USB, podrien no ser necessàries si s'aprofita aquesta connexió per a fer arribar els Firmwares als seus respectius elements de còmput.

## 1.3 Objectius

Prenent com a referència un sistema com el de l'exemple de la figura 1.1, tenim una placa connectada a un PC via USB, i l'objectiu d'aquest projecte és aprofitar aquesta connexió per a que quan el PC detecti el dispositiu i carregui el driver, aquest s'encarregui d'enviar el Firmware necessari per a arrencar cadascun dels elements de la placa, i d'aquesta manera poder estalviar-nos totes les memòries no volàtils.

Més concretament aquest objectiu es pot dividir en les tres parts següents:

- Eliminar la memòria que conté el Firmware del controlador USB: Com es veu a la figura 1.1 hi ha una memòria EEPROM on està guardat el codi del controlador USB, i l'objectiu és emmagatzemar aquest codi al PC i descarregar-lo en iniciar el sistema directament a la RAM interna del controlador.

- Configuració de la FPGA: El fitxer de configuració de la FPGA està guardat a una part de la memòria Flash de la figura 1.1. El segon objectiu és fer

arribar la configuració a través de l'USB al controlador, i fer servir el seu processador per a configurar la FPGA.

- Eliminar la memòria que conté el Firmware del SoftCore: En el cas de la figura 1.1, el segon processador del sistema és el SoftCore implementat a la FPGA, i el Firmware que ha d'executar està guardat a una part de la memòria Flash. Per tant, l'objectiu és transferir el codi des del PC al controlador USB, i del controlador USB a la memòria SDRAM externa a través d'una connexió sèrie entre els dos processadors.

## 1.4 Planificació Temporal

Per poder realitzar una bona gestió i control del projecte és necessari una planificació temporal que ens permeti dividir el projecte en tasques i objectius.

La primera part del projecte consisteix en l'estudi dels fonaments teòrics bàsics tant del controlador USB, com dels mètodes de configuració de la FPGA, com dels SoftCores.

Un cop es té un visió general, es pot començar a aprofundir en cadascuna de les parts del projecte i començar amb la implementació de les aplicacions. En aquest cas es començarà pel disseny del sistema NIOSII que anirà configurat a la FPGA, i en la implementació del protocol que servirà per transferir el Firmware a executar pel NIOSII, des del PC a la SDRAM externa de la placa d'Altera.

El següent pas serà el d'estudiar una mica més el sistema d'arrencada del controlador FX2, i aconseguir descarregar el Firmware a través de les Vendor Requests de Cypress, i implementar alguna Vendor Request pròpia.

Un cop dominat tot el necessari referent a l'FX2, s'ha de procedir a estudiar el sistema d'arrencada d'una FPGA, entendre el protocol Passive Serial, i dissenyar les Vendor Requests que faran possible que el processador contingut al controlador USB configuri una FPGA amb aquest protocol.

La memòria s'haurà d'anar implementant a mesura que s'avança en el projecte, per a que al final només s'hagin d'afegir detalls, i fer algunes correccions.

| **Tasques** | **Duració** | **Inici** | **Fi** |
|---|---|---|---|
| *Informe previ* | | | |
| - Redacció | 5 dies | 03/01/11 | 07/01/11 |
| - Entrega | 1 dia | 10/01/11 | 10/01/11 |
| *Estudi de conceptes bàsics* | | | |
| - Funcionament bàsic d'un controlador d'USB | 14 dies | 21/02/11 | 6/03/11 |
| - Conceptes bàsics d'una FPGA | 7 dies | 07/03/11 | 13/03/11 |
| *Sistema NIOSII* | | | |
| - Disseny del Sistema | 14 dies | 14/03/11 | 27/03/11 |
| - Disseny del Protocol | 21 dies | 28/03/11 | 17/04/11 |
| - Elaboració de la memòria | 14 dies | 18/04/11 | 01/05/11 |
| *FX2* | | | |
| - Descarregar el Firmware | 21 dies | 02/05/11 | 22/05/11 |
| - Implementació de Vendor Requests | 14 dies | 23/05/11 | 05/06/11 |
| - Elaboració de la memòria | 14 dies | 06/06/11 | 19/06/11 |
| *Placa Final* | | | |
| - Vendor Request per al Passive Serial | 21 dies | 20/06/11 | 10/07/11 |
| - Adaptació del codi anterior | 21 dies | 11/07/11 | 31/07/11 |
| - Elaboració de la memòria | 14 dies | 01/08/11 | 14/08/11 |
| *Mesura de temps* | 7 dies | *15/08/11* | *21/08/11* |
| *Elaboració de la memòria* | 14 dies | *22/08/11* | *04/09/11* |

Taula 1.1: Planificació inicial del projecte.

# Capítol 2

# Conceptes Tècnics

## 2.1 USB

El Bus USB és una connexió cada vegada més i més utilitzada, no només per a connectar perifèrics a un PC, sinó també per a tot tipus de dispositius com ara equips de música o televisors. Aquesta popularitat va creixent perquè és una connexió ràpida i senzilla, i fa que els dispositius que l'incorporen siguin veritables Plug-and-Play, ja que un dispositiu USB es pot connectar al PC en qualsevol moment i amb prou feines en uns segons el dispositiu ja està llest per a fer-se servir. Això és així perquè quan el sistema operatiu del Host detecta que hi ha un dispositiu USB, l'interroga per conèixer de quin tipus de dispositiu es tracta i quines característiques té, aleshores el sistema carrega el driver específic per a aquell aparell deixant-ho tot preparat per a la comunicació. De la mateixa manera, quan es detecta la desconnexió del dispositiu, el sistema operatiu descarrega el driver del sistema automàticament.

Un concepte fonamental per entendre els dispositius USB és que aquests sempre fan el paper d'Slave en la comunicació, mentre que el Host és el que fa sempre de Master. Això vol dir que sempre és el Host el que inicia la comunicació a base d'enviar comandes al dispositiu USB.

Les comunicacions entre el Host i el dispositiu USB es duen a terme mitjançant unes cues FIFO anomenades EndPoints. Tot i que cada dispositiu pot

tenir un nombre diferents d'EndPoints, no tots aquests són del mateix tipus. Hi ha uns quants que s'utilitzen per a la transmissió de dades de l'usuari i sempre hi ha com a mínim un de reservat per a la comunicació directa entre el Host i el dispositiu USB.

### 2.1.1 FX2

L'FX2 és un controlador USB de Cypress. La CPU d'aquest controlador és un derivat del microcontrolador 8051, amb un port USB 2.0 de 480 Mbps i amb una sola memòria volàtil per a emmagatzemar el codi a executar i les dades. Per això, generalment es posa una memòria EEPROM al costat per a emmagatzemar-ho tot.

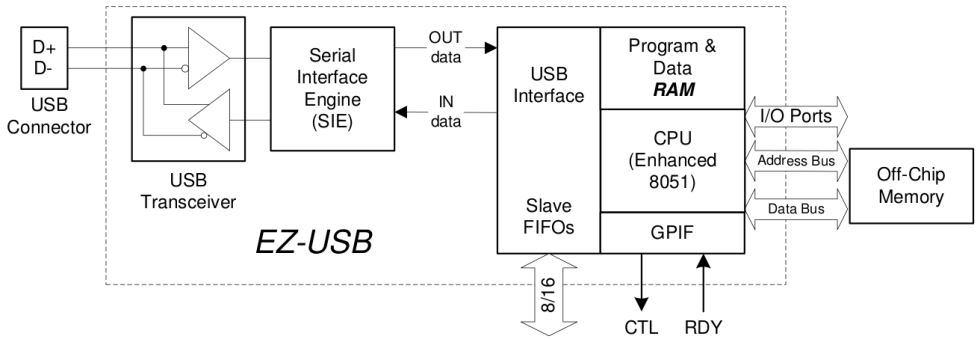


Figura 2.1: Diagrama de Blocs de l'FX2

#### EndPoints

L'especificació d'USB defineix els EndPoints com a font o magatzem de dades, i com un USB és un bus sèrie, això queda reflectit en cues FIFO. Per a referir-se a cadascuna d'aquestes cues es fa servir una direcció de 4 bits, de manera que un USB pot tenir fins a 32 EndPoints.

L'FX2 disposa de 3 EndPoints de 64 Bytes més 4 KBytes d'espai d'emmagatzematge per altres EndPoints. Els 3 primers són l'EP0, l'EP1IN i l'EP1OUT. L'EP0 és l'EndPoint de control per defecte, i és bidireccional. Els EP1IN i EP1OUT

són dos buffers de 64 Bytes, un per a cada sentit de la comunicació. Aquests EndPoints només són accessibles des del Firmware del controlador.

Per a fer servir els altres 4 KBytes d'espai, es poden fer servir fins a 4 EndPoints (EP2, EP4, EP6 i EP8), i hi ha 12 configuracions possibles, segons el nombre d'EndPoints que s'utilitzin i la seva mida. La configuració que es vulgui fer servir es defineix al principi del Firmware del controlador, i si no s'especifica, la configuració per defecte consisteix en assignar 1 Kbyte per a cada EndPoint, és a dir, 512 Bytes per a cada sentit de la comunicació.

La figura 2.2 representa una sistema on una FPGA està connectada a un PC via USB (El PC és el Host), i hi ha un exemple de configuració dels EndPoints on es veu que el PC pot escriure dades a l'EP2OUT, i a l'EP4OUT, que són dels que la FPGA pot llegir. I els que serveixen per a que la FPGA escrigui dades per al PC que són l'EP2IN i l'EP4IN. A més està representat l'EP0, al que la FPGA no pot accedir de cap manera donat que és únicament per a la comunicació entre l'FX2 i el PC.

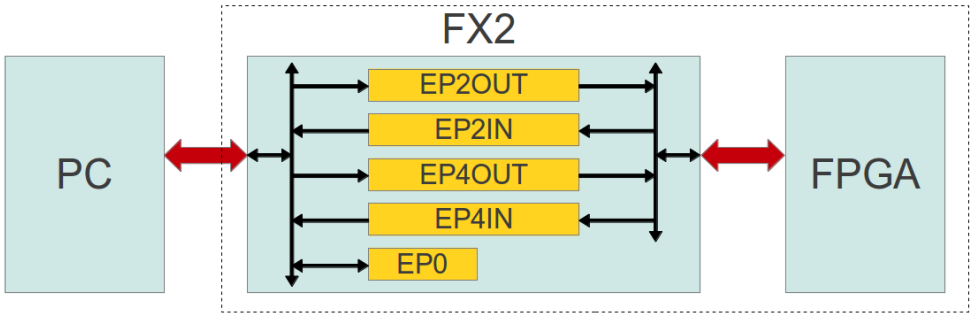


Figura 2.2: EndPoints

**EndPoint 0 (EP0)**

L'EndPoint 0 és un EndPoint de control obligatori per a qualsevol dispositiu USB, donat que és per on el Host envia totes les comandes al controlador. Hi ha 3 tipus de comandes, les Standard Request, les Class Request i les Vendor Request.

Les Standard Request són comandes que han de reconèixer tots els dispositius

USB que hi ha al mercat i poden servir, per exemple, per demanar l'estat del dispositiu, la seva identificació o per canviar algun paràmetre de configuració.

A més segons el tipus de dispositiu que sigui, pot tenir unes comandes més específiques, aquestes són les Class Request. Per exemple, els dispositius d'emmagatzematge massiu tenen una sèrie de Class Request que no tenen el dispositius de comunicacions.

I per últim les Vendor Request són comandes totalment lliures, és a dir, que l'usuari les pot programar per a que el seu dispositiu reconegui una comanda i executi la funció que ell vulgui.

Cadascuna d'aquestes comandes està formada per 2 o 3 fases, depenen de si hi ha dades addicionals a enviar o no. Aquestes fases són la de SETUP, la de DATA i la d'STATUS.

Cadascuna d'aquestes etapes entre el Host i l'USB estan formades per 3 tipus de paquets. El primer és un Token, que és una capçalera que anuncia el tipus de transacció, la direcció i l'EndPoint que es farà servir. El segon és un paquet de dades, i el tercer és un paquet de HandShake, és a dir, de confirmació.

A l'etapa de SETUP [figura 2.3] s'envia un Token indicant que és una transacció de control, un paquet de dades amb el tipus de comanda que s'està enviant (Standard, Class o Vendor), quina comanda en concret i algunes dades més pròpies de cada comanda, i per últim el paquet de confirmació.

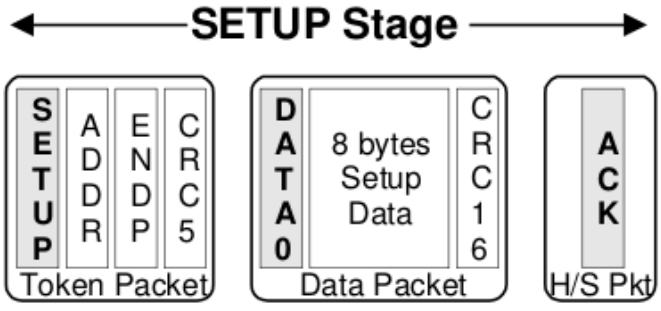


Figura 2.3: Etapa de SETUP

L'etapa de dades [figura 2.4] és opcional, només es fa servir si la comanda que s'està processant ha de moure més dades que les que caben a l'etapa de SETUP.

El Token que s'envia indica si les dades s'han d'enviar del Host a l'USB (OUT) o a l'inrevés (IN), la direcció i l'EndPoint. Després ve el paquet de dades i per últim el de confirmació. Aquesta etapa es repeteix tantes vegades com sigui necessari fins que totes les dades arribin a la seva destinació.

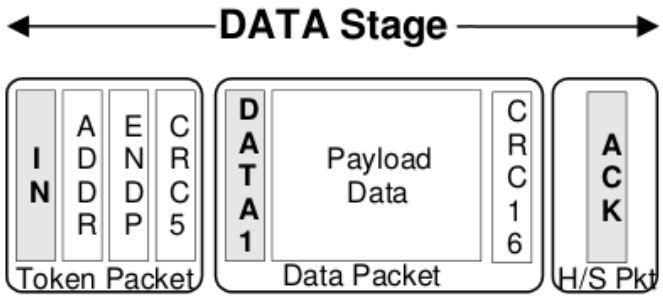


Figura 2.4: Etapa de Dades

Finalment l'etapa d'STATUS indica si s'ha acabat la transacció de forma correcta.

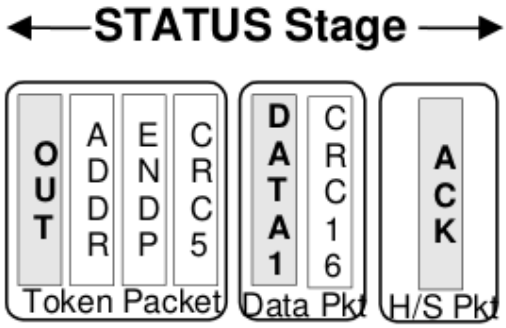


Figura 2.5: Etapa d'STATUS

### Arrencada

El primer que fa el Host quan detecta que s'ha connectat un dispositiu USB a algun dels seus ports és demanar-li les dades d'identificació, concretament el VID (Identificació del fabricant) i el PID (Identificació del producte) per tal de carregar

el driver necessari per fer servir aquest dispositiu. D'aquest procés d'identificació se'n diu Enumeració. Hi ha alguns casos en que aquestes dades d'identificació canvien mentre el dispositiu USB està connectat, de manera que fa que el host hagi de carregar uns nous drivers. Aquesta segona identificació s'anomena Re-enumeració.

L'FX2, un cop alimentat, posa en funcionament una màquina d'estats que és capaç de dur a terme algunes funcions bàsiques de forma autònoma, sense cap tipus d'intervenció del processador 8051 que incorpora. El primer que farà aquesta màquina serà mirar si té una memòria no volàtil connectada, i depenen de si hi és o no, i del que hi hagi a la primera direcció de memòria arrencarà en un d'aquests 3 modes:

- Mode C2: Si el primer Byte de la memòria no volàtil és 0xC2 vol dir que a aquesta memòria hi trobarà tant les dades per a la Enumeració, com el codi del Firmware que haurà de carregar a la RAM. Per tant, s'Enumerarà amb les dades corresponents i agafarà el Firmware de la memòria no volàtil per a guardar-lo a la RAM i començar a executar-lo.

- Mode C0: Si la memòria hi és, però el seu primer Byte és 0xC0, significa que les 16 posicions següents contenen les dades d'identificació i que la memòria no conté res més. Per tant farà l'Enumeració amb les dades que ha trobat a la memòria i esperarà a que el Host li enviï el Firmware que haurà d'executar. De totes maneres, durant aquesta espera, el dispositiu continuarà executant la màquina d'estats, de manera que podrà atendre algunes comandes que li arribin des del Host.

- Mode "No EEPROM": En el cas de que no trobi cap memòria no volàtil, o que les dades siguin invàlides (El primer Byte no sigui ni 0xC0 ni 0xC2) s'enumerarà amb el PID=0x8613 i el VID=0x04B4, que són les dades d'identificació d'un dispositiu genèric d'USB. Igual que en el mode C0, l'USB continuarà en funcionament, i si el Host li envia el Firmware a executar amb uns nous PID i VID, simularà una desconnexió i una reconnexió, provocant

així que el Host descarregui el driver genèric i faci que el dispositiu USB es Re-enumeri per tal de carregar el driver corresponent.

Aquesta màquina d'estats té implementades unes Vendor Requests creades per Cypress que permeten guardar dades a la memòria interna del controlador, fent possible que si s'arrenca el sistema en mode "No EEPROM" o "C0", el Host pugui enviar el Firmware de l'FX2 a través de l'EP0 i executar aquest nou codi.

## 2.2 FPGA

Una FPGA és un dispositiu que conté blocs de lògica programables i la interconnexió dels quals és configurable, de manera que es poden implementar des de portes lògiques o circuits combinacionals fins a sistemes sencers incloent processadors, perifèrics i memòries. A la figura 2.6 es pot veure un d'aquests blocs lògics, anomenats Logic Elements (LE) o Logic Cells (LC). Un LE és l'element bàsic configurable, a partir del qual, es pot implementar qualsevol disseny digital. Un LE està format per una Look Up Table (LUT), un Flip-Flop D i un multiplexor. Una LUT és un component que té una part de RAM (en el cas de la figura 2.6 4 bits) on es posen els valors de sortida corresponents a la taula de la veritat que es vol implementar. Per exemple, a la figura 2.6 està implementada una porta AND de dues entrades. El bit de selecció del Multiplexor de la figura serveix per a decidir si es farà servir el resultat de la LUT o del Flip-Flop, de manera que tant aquest bit de selecció com el bits corresponents a les possibles sortides de la LUT són els bits que s'han d'assignar per a realitzar la configuració de la FPGA, i tots ells estan escrits en memòria RAM, de manera que s'han de configurar cada vegada que s'alimenta el sistema.

Segons el tipus de tecnologia que fa servir per a la programació del seu contingut, una FPGA pot ser no volàtil (basada en ROM), és a dir, que després de configurar-la, un cop li treus l'alimentació manté el sistema, o bé volàtil, de manera que quan es deixa d'alimentar es desconfigura.

Les FPGAs volàtils, necessiten carregar el circuit implementat cada vegada que s'encén el sistema. Segons el fabricant i el model, cada FPGA pot tenir difer-

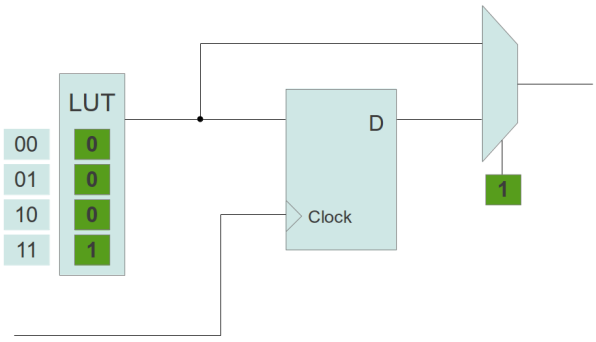


Figura 2.6: Logic Element

ents mecanismes per a carregar la configuració, però el més usual és tenir una memòria no volàtil al costat (típicament una Flash) on la FPGA vagi a buscar la configuració que ha de carregar.

### 2.2.1 Configuració de la CycloneII

La CycloneII és una FPGA d'Altera, i aquesta té 3 sistemes per a carregar la configuració.

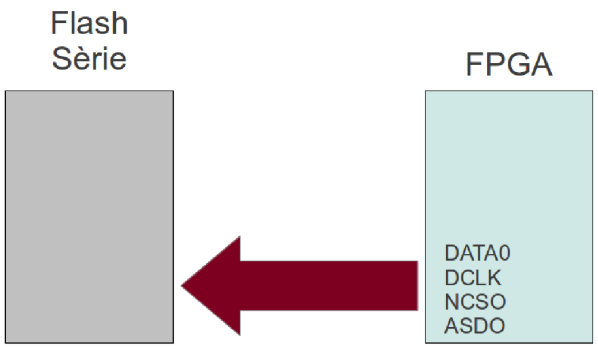


Figura 2.7: Connexió de l'Active Serial

L'Active Serial [figura 2.7] consisteix en connectar una memòria Flash a la FPGA i anar llegint els bits de configuració fins a tenir tots els elements a punt. Aquest mètode es diu Active Serial perquè és l'FPGA la que, un cop alimentada,

va automàticament a buscar la configuració a aquesta memòria i s'encarrega de gestionar el rellotge que marca la transferència de la informació bit a bit de forma sèrie.

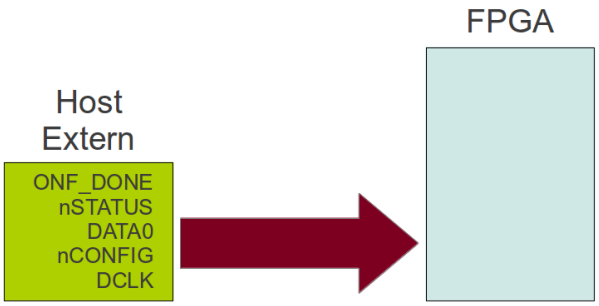


Figura 2.8: Connexió del Passive Serial

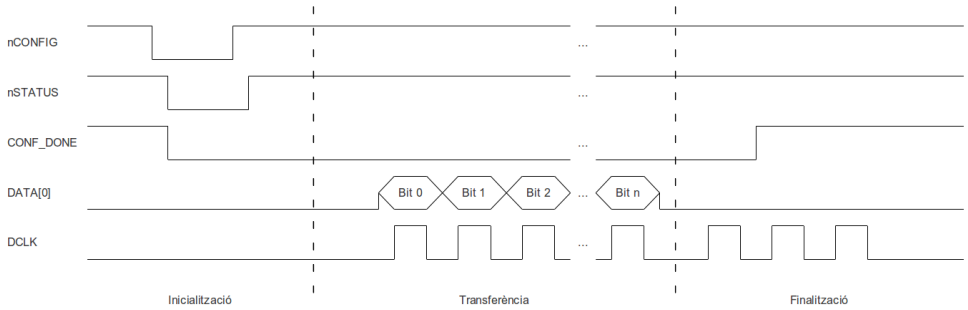


Figura 2.9: Diagrama d'Ones del Passive Serial

En el Passive Serial [figura 2.8] el que s'encarrega de configurar el contingut de l'FPGA és un Host extern.

Com es veu a la figura 2.9, el Passive Serial es pot dividir en 3 parts, la d'Inicialització, on tant la FPGA com el Host es posen a punt per a començar, la fase de Configuració, que és on s'envien els bits en sèrie a través d'una de les línies al ritme marcat pel rellotge generat pel Host, i per últim la fase de Finalització, on la FPGA indica que ha sigut tot correcte.

La configuració per JTAG [figura 2.10] es fa servir sobretot per a fer tests i depurar el sistema implementat. Consisteix en connectar la FPGA a un port de

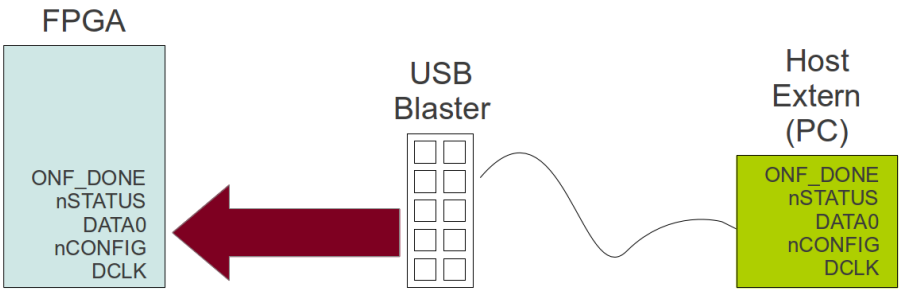


Figura 2.10: Connexió per la configuració per JTAG

configuració (un USB Blaster), i fer-lo servir per a descarregar el fitxer de configuració directament a l'FPGA mitjançant una eina del Software d'Altera QuartusII.

## 2.3 SoftCores

Un SoftCore és un microprocessador on la seva arquitectura i el seu comportament està totalment descrit amb un lleguatge d'alt nivell de descripció de Hardware (HDL Hardware Description Language), i aquest SoftCore es pot sintetitzar després en un circuit ASIC o bé en algun dispositiu de lògica programable, com ara una FPGA.

La utilització dels SoftCores proporciona alguns avantatges per al disseny de circuits encastats. En primer lloc, un processador SoftCore és flexible i es pot adaptar a diferents aplicacions d'una forma relativament senzilla. A més, donat que estan descrits amb llenguatges d'alt nivell, fa que siguin més fàcils d'entendre i de modificar que els processadors descrits a nivell lògic, de manera que triguen més en quedar obsolets.

Avui en dia la gran majoria d'FPGAs contenen implementats SoftCores, i la implementació de circuits pràcticament només es fa servir per a testejar-los abans de muntar un ASIC.

### 2.3.1 NIOSII

Altera proporciona algunes eines per a muntar un sistema sencer basat en el Soft-Core NIOSII per a fer-lo funcionar en una de les seves FPGAs.

El Software d'Altera QuartusII permet dissenyar circuits en llenguatges de descripció de Hardware, sintetitzar-los i descarregar-los en els seus productes. A més, conté una eina anomenada SOPC Builder que permet anar afegint mòduls d'una forma simple al voltant del Softcore NIOSII. La figura 2.11 és un diagrama de blocs d'un sistema basat en el NIOSII, com es pot veure, la CPU està interconnectada amb els mòduls d'Entrada/Sortida, les memòries internes per a dades i codi, i els mòduls per a gestionar els perifèrics, tot mitjançant un bus propi d'Altera anomenat Avalon.

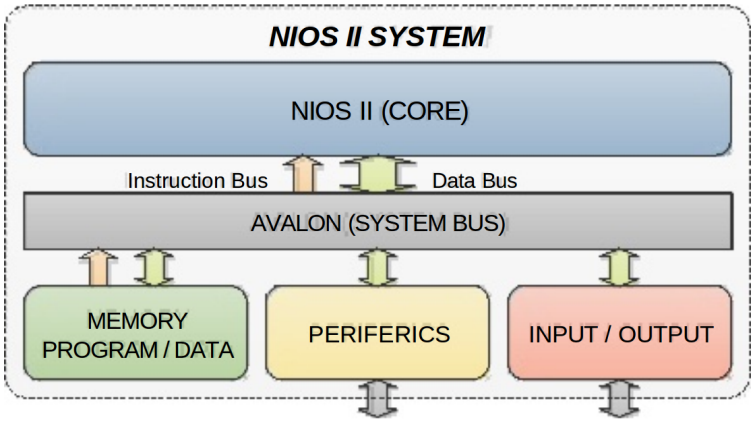


Figura 2.11: Sistema basat en un NIOSII

La memòria interna per a codi que tenen els sistemes implementats en FPGAs solen ser bastant petites degut al seu cost, de manera que un Firmware relativament elaborat no hi cap. Per això és normal trobar una memòria externa a on s'ha de fer arribar el Firmware final per a executar-lo en última instància.

# Capítol 3

# Implementació i Desenvolupament

La implementació del projecte té com a objectiu optimitzar l'arrencada d'un sistema com el de l'esquema de l'exemple de la figura 1.1 esposat a la introducció.

Per fer això s'ha dividit el projecte en 3 parts: La primera consisteix en descarregar el Firmware del controlador FX2 directament pel port USB, la segona en la configuració del sistema que allotjarà la FPGA, en aquest cas un Sofcore NIOSII, i per últim l'enviament del Firmware final que executarà aquest NIOSII, donant així per acabada l'arrencada del sistema.

## 3.1 Firmware USB

En aquesta primera fase, l'objectiu és descarregar, des del PC a la memòria interna de l'FX2, un fitxer HEX que conté el Firmware que haurà d'executar el controlador mitjançant les Vendor Requests corresponents.

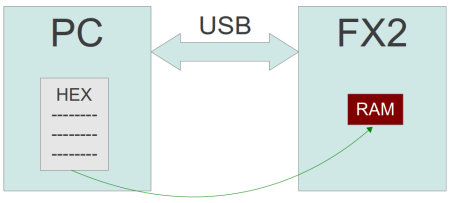


Figura 3.1: Primera Fase

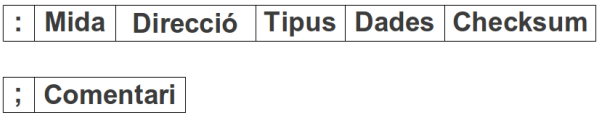


Figura 3.2: Format de les línies d'un fitxer HEX

### 3.1.1 Fitxer HEX

Un fitxer HEX és un fitxer de text que conté la representació en memòria del codi que es vol executar a l'FX2. Cada línia d'un d'aquests fitxers té un dels dos formats de la figura 3.2 depenen del primer caràcter.

Si el primer caràcter són ':' indica que és un fitxer amb contingut, i que conté el següents camps:

- Mida: Són dos caràcters en hexadecimal que indiquen la mida del camp de dades en Bytes.
- Direcció: Quatre caràcters amb la direcció corresponent al primer Byte de dades.
- Tipus de línia: Són dos caràcters que indiquen el tipus de línia que s'està llegint. En aquest cas, el compilador de l'entorn Keil uVision2 genera un fitxer HEX només amb línies del tipus 00. Els altres tipus serveixen per a fer servir direccions de més de 16 bits.
- Dades: Conté les dades que s'han de guardar a la memòria del controlador FX2.
- Checksum: Són dos caràcters que serveixen per comprovar que les dades són correctes. Per a calcular aquest checksum, es fa la suma Byte a Byte de tots els camps anteriors (excepte el primer caràcter), s'agafa l'últim Byte del resultat de la suma i es fa el complement a dos.

En cas de que el primer caràcter sigui un ';', indica que és una línia de comentari, i tot el que ve després és text sense significat per al controlador.

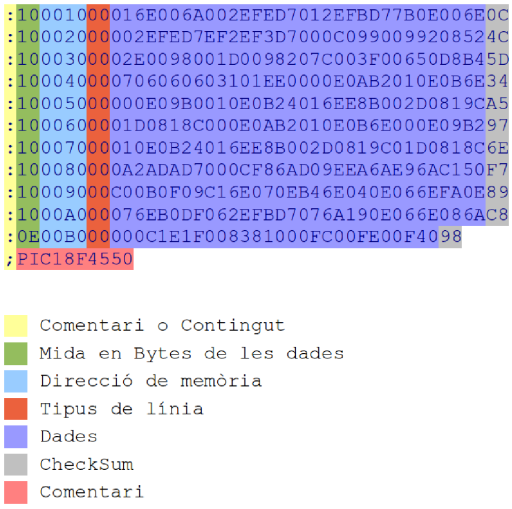


Figura 3.3: Fitxer HEX

A la figura 3.3 es pot veure un exemple d'un fitxer HEX.

La primera línia indica, amb els ':', que és una línia amb contingut i amb 16 Bytes (10 en hexadecimal) de dades que han d'anar a partir de la direcció de memòria 0x0010. Després indica que és una línia de dades pel codi 00, i el segueixen els 16 Bytes de dades, i per acabar el checksum, que es calcula de la següent manera:

Primer es fa la suma Byte a Byte de tots el camps (5F4), s'agafa l'últim Byte del resultat (F4) i es fa el complement a 2, i el resultat és 0C.

Finalment veiem que l'última línia comença amb un ';', que ens indica que és un comentari, i ve seguit per un text sense significat per al controlador.

### 3.1.2 Enviament des del PC a l'FX2 del fitxer HEX

Com s'exposa al capítol 2, Cypress ha implementat unes Vendor Requests a la màquina d'estats interna del controlador FX2, que permeten descarregar dades a qualsevol part de la seva memòria. En aquest cas un fitxer amb extensió HEX que conté el Firmware a executar. El protocol que permet enviar aquest fitxer té tres passos:

| bmRequest | bRequest | wValue | wIndex | DATA | wLength |
|---|---|---|---|---|---|
| 0x40 | 0xA0 | 0xE600 | 0x00 | 0x01 | 0x01 |
| 0x40 | 0xA0 | Direcció de la línia de l'HEX | 0x00 | Dades de la línia de l'HEX | Mida de la línia de l'HEX |
| 0x40 | 0xA0 | 0xE600 | 0x00 | 0x00 | 0x01 |

Taula 3.1: Vendor Requests de la primera part

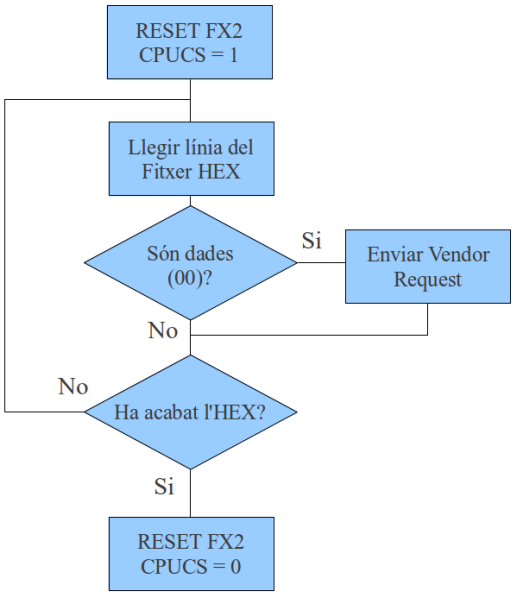


Figura 3.4: Execució del PC per a enviar el fitxer HEX

El primer consisteix en posar el controlador en mode RESET per a preparar-lo per a la recepció de les dades. Per això es fa servir la primera Vendor Request de la taula 3.1, que el que fa es posar un 1 al registre CPUCS, que correspon a la direcció de memòria 0xE600.

El segon pas consisteix en, un cop el controlador està en mode de RESET, enviar-li el fitxer HEX. Es pot enviar mitjançant una Vendor Request amb les dades de la segona línia de la taula 3.1.

I per acabar s'ha de tornar a arrencar el controlador, és a dir, treure'l de l'estat de RESET. Això s'aconsegueix posant un 0 al registre CPUCS, que és el que fa

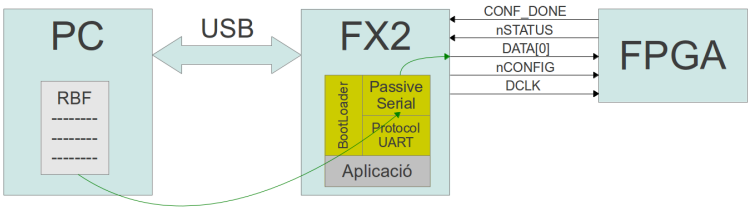


Figura 3.5: Segona Fase

la tercera Vendor Request de la taula 3.1. Quan el controlador FX2 rep aquesta comanda simula una desconnexió i una reconnexió amb el Host i comença a executar el Firmware que s'ha descarregat a la memòria (Re-enumeració).

A la figura 3.4 es pot veure el diagrama de flux que segueix l'aplicació que s'executa al PC per a descarregar el Firmware a l'FX2.

## 3.2 Configuració de la FPGA

Per a configurar la FPGA farem servir el protocol Passive Serial explicat al capítol 2. Per a dur-lo a terme s'ha de fer servir un fitxer amb extensió RBF. Aquest és un fitxer binari que conté la configuració de cadascun dels elements de la FPGA per a implementar el disseny que es vulgui.

En aquest moment, el Firmware de l'FX2 ha de tenir dues parts diferenciades: el BootLoader i el codi d'aplicació [figura 3.5].

El BootLoader és el codi que haurà d'anar executant per arrencar la resta del sistema, i per tant, en aquest cas, té una part que s'encarregarà de la fase 2, en la que es configura la FPGA, i una altra part per a la fase 3, que enviarà el Firmware final que haurà d'executar el sistema implementat a la FPGA.

El codi d'aplicació correspon al Firmware que s'executarà a l'FX2 quan el procés d'arrencada hagi finalitzat.

### 3.2.1 Enviament des del PC a l'FX2 del fitxer RBF

El programa d'enviament que s'executa al host on està connectat l'USB (en aquest cas el PC) llegeix el fitxer RBF de 64 Bytes en 64 Bytes perquè és la mida de la cua de l'EndPoint 0, i l'envia mitjançant unes Vendor Requests implementades expressament per a cadascuna de les fases del Passive Serial. A la figura 3.6 es pot veure el diagrama de flux d'aquest procés.

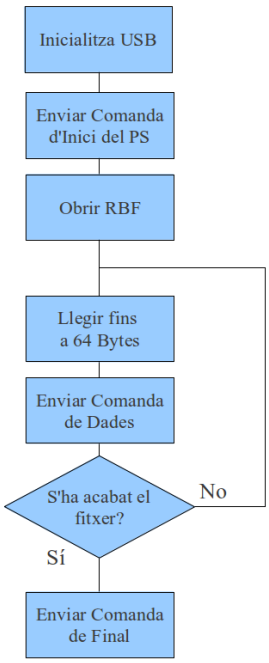


Figura 3.6: Execució del PC per a enviar el fitxer RBF

La primera de les Vendor Requests de la taula 3.2 és la que s'encarrega de dir al controlador FX2 que prepari la FPGA per a començar la seva configuració mitjançant el Passive Serial. Per a executar aquesta comanda l'FX2 no necessita dades addicionals, per això el camp de dades es deixa a NULL.

La segona Vendor Request és la que es fa servir per a enviar les dades de configuració de la FPGA llegides del fitxer RBF.

I per últim, la tercera comanda de la taula 3.2 serveix per a fer els últims passos del Passive serial i l'FX2 tampoc necessita cap tipus de dades per a executar-la.

| bmRequest | bRequest | wValue | wIndex | DATA | wLength |
|---|---|---|---|---|---|
| 0x40 | 0xAE | 0x0000 | 0x00 | NULL | 0x00 |
| 0x40 | 0xAD | 0x0000 | 0x00 | Dades del fitxer RBF | Mida de les dades llegides |
| 0x40 | 0xAF | 0x0000 | 0x00 | NULL | 0x00 |

Taula 3.2: Vendor Requests de la segona part

### 3.2.2 Firmware de l'FX2 per al Passive Serial

El primer que farà l'FX2 serà configurar la capa física amb la que es comunicarà amb la FPGA, i després romandrà a l'espera de les comandes del Host. El controlador USB serà capaç de respondre a 3 Comandes noves per a realitzar cadascuna de les 3 fases del Passive Serial tal i com es veu al diagrama de flux de la figura 3.7.

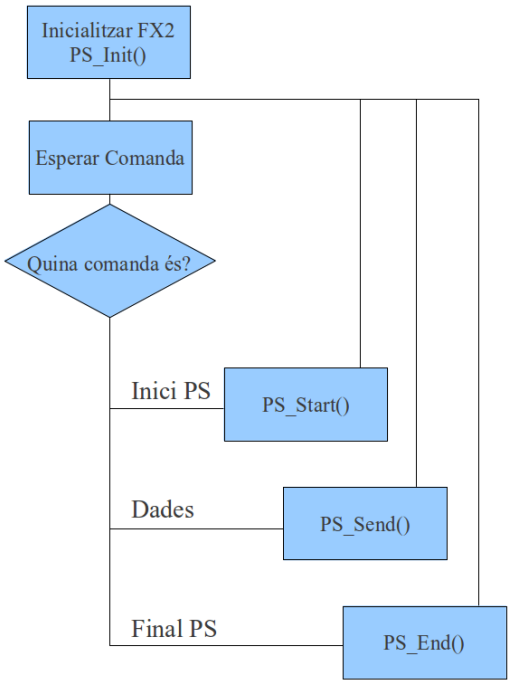


Figura 3.7: Firmware FX2

La comunicació del controlador FX2 amb la FPGA per a dur a terme el Passive

Serial es fa mitjançant 5 senyals: CONF_DONE (Entrada), nSTATUS (Entrada), DATA[0] (Sortida), nCONFIG (Sortida) i DCLK (Sortida), sempre des del punt de vista de l'FX2. Aquests senyals són 5 línies tirades des de ports d'Entrada/Sortida de propòsit general de l'FX2 als ports dedicats de la FPGA.

Durant la part d'inicialització del Firmware de l'FX2 s'executa la funció PS_Init() que s'encarrega de configurar aquestes 5 potes de propòsit general com entrades i sortides segons correspongui. A més ha d'inicialitzar els senyal de sortida DATA[0] i DCLK a 0 i nCONFIG a 1.

A partir d'aquest moment l'FX2 espera a rebre les comandes i les dades necessàries per a configurar la FPGA.

La primera comanda de la taula 3.2 s'encarrega d'executar la funció PS_Start(), que serveix per a indicar a la FPGA que es vol procedir a la seva configuració via Passive Serial. Per a fer això l'FX2 posa a 0 el senyal nCONFIG (Punt 1 de la figura 3.8) i espera a que la FPGA contesti posant l'nSTATUS i el CONF_DONE també a 0 (Punt 2 de la figura 3.8), després d'això el controlador FX2 torna a posar el senyal nCONFIG a 1 (Punt 3 de la figura 3.8) i espera a l'última confirmació de la FPGA indicada posant a 1 el senyal nSTATUS (Punt 4 de la figura 3.8). En aquest moment la FPGA ja està preparada per a rebre les dades de configuració del fitxer RBF.

La segona comanda de la taula 3.2 executa la funció PS_Send(). Aquesta funció s'encarrega d'agafar les dades que li arriben per l'EndPoint 0 i enviar-les bit a bit pel senyal DATA[0].

El primer que s'ha de tenir present per a aquesta fase és que els Bytes de dades s'enviaran bit a bit començant pel bit menys significatiu, és a dir, que si a l'FX2 li arriba la seqüència de Bytes 0x02, 0x1B, 0xEE, 0x01 i 0xFA s'haurà d'enviar la següent combinació de bits:

0100-0000 1101-1000 0111-0111 1000-0000 0101-1111

Per a enviar un bit, primer es posa la línia DATA[0] a 0 o a 1 segons el bit que es vol enviar (Punt 5 de la figura 3.9) i després es puja el senyal de rellotge DCLK a 1 (Punt 6 de la figura 3.9), i es manté la línia DATA[0] fins que arribi el flanc de

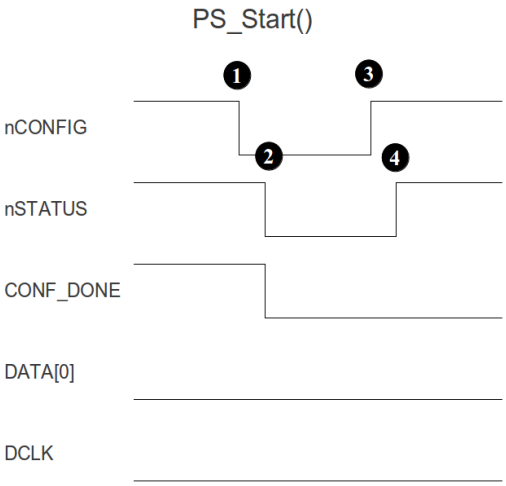


Figura 3.8: Inici del Passive Serial

baixada del rellotge (Punt 7 de la figura 3.9), i així successivament fins a acabar amb totes les dades.

És interessant saber que els períodes no tenen per què ser ni tots iguals, ni regulars (els semiperíodes poden ser de temps diferents), i que no tenen límit màxim de temps. Gràcies a això no cal tenir emmagatzemat el fitxer binari sencer a memòria per tractar-lo tot de cop, sinó que es pot enviar el fitxer en paquets, i atendre cada paquet per separat. A més, si fos necessari, no hi hauria cap problema en fer una pausa a mitja configuració i seguir més endavant on s'havia deixat (sempre i quan el rellotge es mantingui en un valor constant).

Finalment, la tercera i última comanda de la taula 3.2 és la que crida a la funció PS_End() [figura 3.10]. Aquesta funció, un cop acabada la transferència de tot el fitxer RBF, s'encarrega d'observar la línia CONF_DONE fins que es posi a 1, i de mantenir el rellotge actiu fins que faci com a mínim dos flancs de baixada més. A més s'ha de tenir cura de no deixar la línia DATA[0] "flotant", sinó que s'ha de posar a 1 o a 0 segons convingui. Un cop fet això ja tenim la FPGA configurada i en funcionament.

Si en algun moment abans d'acabar la transferència de tot el fitxer, la línia CONF_DONE es posés a 1, indicaria que hi ha hagut algun error en la configu-

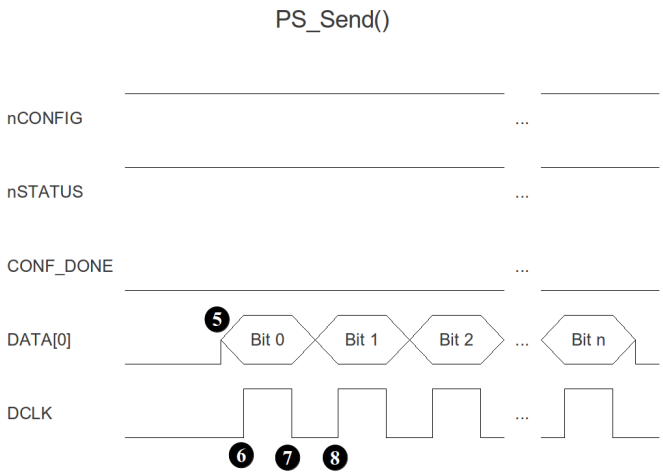


Figura 3.9: Configuració bit a bit del Passive Serial

ració del dispositiu, i s'hauria de tornar a començar des del principi.

## 3.3 NIOSII

Quan s'implementa un NIOSII és necessari implementar-hi també una memòria RAM on hi hagi el seu codi d'arrencada. El problema és que implementar memòria RAM en una FPGA és més costós i limitat que utilitzar una memòria externa, per aquest motiu és interessant fer que el codi d'arrencada del NIOSII estigui escrit per a rebre les dades corresponents al codi a executar per algun port d'Entrada/Sortida i transferir-les a una memòria externa, i un cop descarregat tot el codi a aquesta memòria fer un salt i executar el Firmware final.

En aquesta part el controlador FX2 farà servir la segona part del seu Boot-Loader, rebrà les comandes per l'EndPoint 0 i les enviarà a través del seu port sèrie cap al NIOSII.

### 3.3.1 Fitxer SREC

Un fitxer SREC és un fitxer de text que conté la informació d'un programa Byte a Byte en hexadecimal, de manera que es puguin escriure els valors hexadecimals

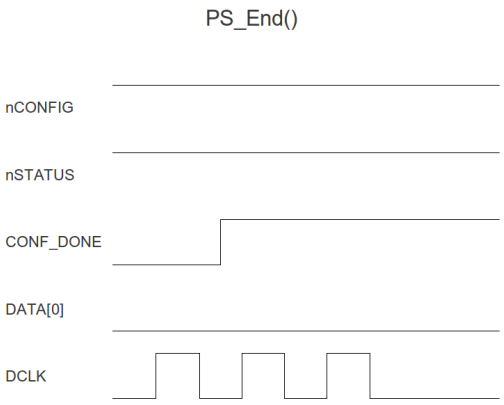


Figura 3.10: Final del Passive Serial

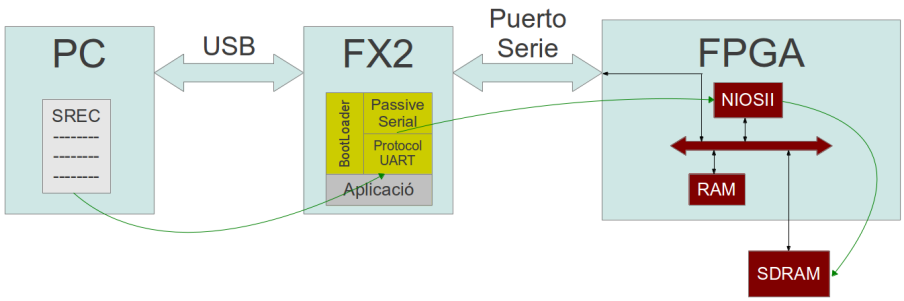


Figura 3.11: Tercera Fase

directament en memòria per a executar el programa que representa. Cada línia d'un fitxer SREC té el format de la figura 3.12.

- Tipus: Indica quin tipus de línia és.
- Mida: 2 caràcters en hexadecimal que indiquen la quantitat de dades en Bytes que conté la línia.

| Tipus | Mida | Direcció | Dades | Checksum |
|---|---|---|---|---|

Figura 3.12: Format de les línies d'un fitxer SREC

- Direcció: En funció del tipus de línia que sigui, és un camp de 4, 6 o 8 caràcters en hexadecimal que indica la direcció del primer Byte de dades de la línia.

- Dades: Conté les dades que s'han de guardar a la memòria.

- Checksum: El checksum dels fitxers SREC es calcula sumant Byte a Byte tots els camps anteriors exceptuant el camp del tipus de línia, agafant l'últim Byte del resultat i fent el complement a 1.

El camp del tipus de línia pot tenir els següents valors:

- S0: Indica l'inici d'un bloc de dades, el camp d'adreça és de 4 xifres i està omplert amb zeros, i les dades que pot portar són el nom del mòdul, la versió, la revisió i la descripció.

- S1, S2 i S3: Contenen dades a carregar en memòria, i el seu camp d'adreça és de 4, 6 o 8 dígits respectivament (adreces de 8, 16 o 32 bits).

- S5: És una línia que no conté dades i fa servir el seu camp d'adreça de 4 dígits per indicar el nombre de línies de tipus S1, S2 i S3 que hi ha per sobre.

- S7, S8 i S9: Aquests últims tres tipus de registre tampoc contenen dades, i indiquen quina és la direcció de memòria on ha de començar a executar-se el programa. El seu camp de direcció és de 4, 6 o 8 dígits respectivament (adreces de 8, 16 o 32 bits).

A la figura 3.13 es pot veure un exemple de fitxer SREC.

La primera línia comença amb S0 indicant que s'inicia un bloc de dades, i l'adreça està tota a zeros. Les 8 línies següents comencen indicant que són del tipus S3. Això vol dir que són línies de dades amb direccions de 32 bits. I la última és una línia del tipus S7, i això vol dir que la direcció que porta al camp d'adreça és la direcció a la que ha de saltar el processador per a executar el codi representat a l'SREC.

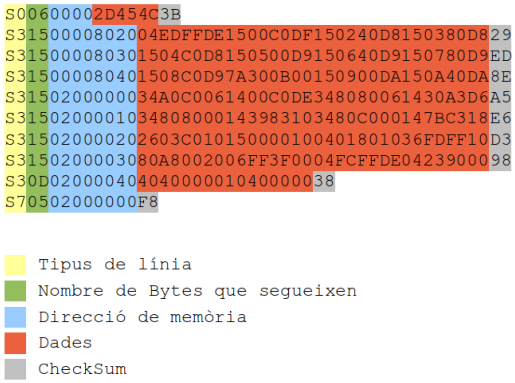


Figura 3.13: Fitxer SREC

| Tipus | Mida | Direcció | Dades | Checksum |
|---|---|---|---|---|

Figura 3.14: Format de comanda del Protocol

A més totes les línies acaben amb el checksum corresponent. Per exemple, el checksum de la primera línia es calcula fent la suma de tots els camps anteriors (C4), agafant l'últim Byte del resultat (C4), i calculant el complement a 1, que és igual a 3B.

### 3.3.2 Enviament des del PC a l'FX2 del fitxer SREC

El programa executat al PC s'encarrega de llegir el fitxer SREC i enviar les dades útils per al NIOS II. Per fer-ho s'ha implementat un protocol de comunicació per a encapsular les dades del fitxer SREC en una sèrie de comandes amb el format de la figura 3.14.

#### Protocol de comunicació

Bàsicament l'aplicació executada al PC fa de Parser del fitxer SREC (el passa de caràcters de text al seu valor numèric, fent que una comanda ocupi la meitat que una línia) de tots els camps excepte el de tipus de registre, i afegeix el seu propi Byte de control al principi de les noves comandes. Aquest Byte de control pot

tenir els següents valors:

- 0: Indica l'inici de la transmissió, i no conté cap altre camp.
- 1: Notifica al receptor que acaba la transmissió. Quan el NIOSII llegeix un paquet, calcula el checksum i veu que és incorrecte, demana una retransmissió, si això passa cinc vegades es dona el fitxer per corrupte i es decideix acabar amb la transmissió. Aquesta comanda tampoc necessita més camps.
- 2: Paquet d'ACK. És un paquet sense dades que envia el receptor a l'emissor per notificar que ha rebut el paquet anterior de forma correcta.
- 3: Paquet de NACK. El receptor rep una comanda, calcula el seu checksum i comprova que no coincideix amb el que ha rebut pel port sèrie, i envia una paquet de tipus 3 per notificar-ho a l'emissor i esperar una retransmissió del paquet. No conté més camps.
- 4: Comanda de dades, a l'SREC una línia que comença amb S3 (En aquest cas les direccions són sempre de 32 bits).
- 5: Direcció de Reset. Implica que el fitxer SREC s'ha acabat i que el NIOSII ja pot saltar a la direcció de memòria indicada a la comanda per a començar a executar el nou Firmware.

**Execució al PC**

L'aplicació que s'encarrega de l'enviament del fitxer SREC va llegint el fitxer SREC línia per línia i les va encapsulant segons el protocol anterior, i va enviant les comandes mitjançant una nova Vendor Request amb les dades mostrades a la taula 3.3.

A la figura 3.15 es pot veure el diagrama de flux d'aquesta l'aplicació.

### 3.3.3 Firware del NIOSII. Recepció

El Firmware del NIOSII està contingut al primer bloc de memòria RAM i va implícit en la configuració de la FPGA. Aquest programa s'encarregarà de rebre

| bmRequest | bRequest | wValue | wIndex | DATA | wLength |
|---|---|---|---|---|---|
| 0x40 | 0xB0 | 0x0000 | 0x00 | Comanda | Mida de la comanda |

Taula 3.3: Vendor Requests de la tercera part

els paquets que li enviï l'emissor i posar-los a la direcció de memòria indicada.

En primer lloc inicialitza el port sèrie (init_uart()) i es queda escoltant de forma bloquejant a l'espera d'un paquet d'inici de transmissió. Un cap rebut envia un ACK i espera el primer paquet de dades. Quan el rep comprova que el checksum és correcte, i si ho és guarda les dades a l'adreça indicada al paquet i envia una confirmació. Si el checksum falla envia un paquet d'error i espera a tornar a rebre el paquet.

En cas de que rebi un paquet de Reset, comprovarà igualment el checksum, enviarà un ACK (si és correcte) i farà un salt incondicional a la direcció de memòria indicada al paquet.

Si el que rep és una comanda de finalització de la transmissió, tornarà al punt inicial del programa a l'espera de que es torni a iniciar una nova transmissió.

La figura 3.16 correspon al diagrama de flux d'aquesta execució.

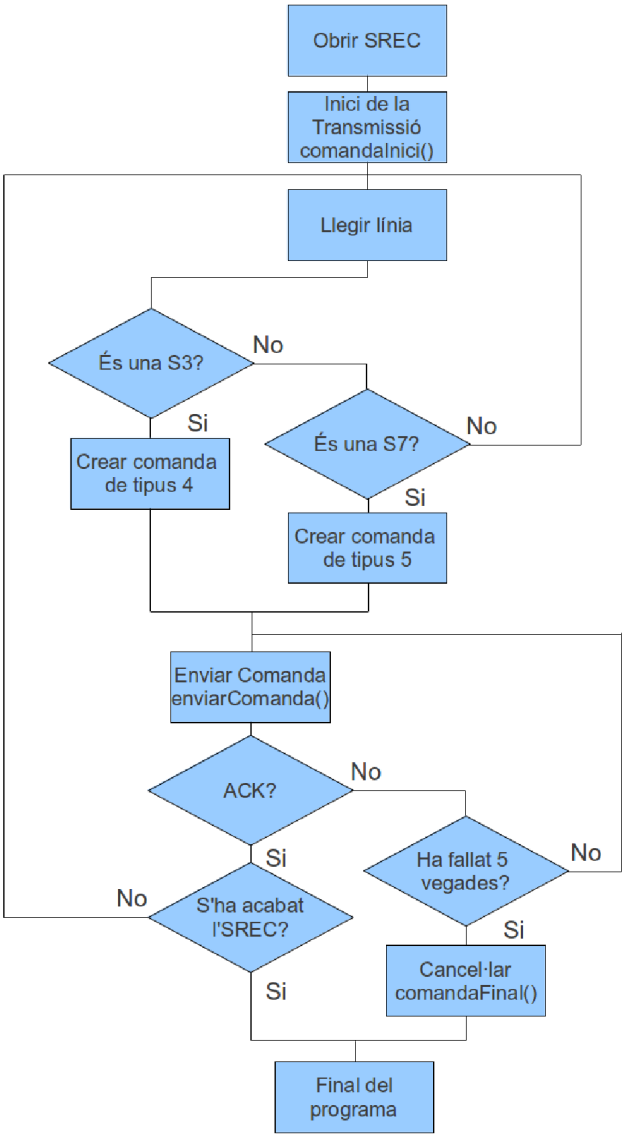


Figura 3.15: Diagrama de Flux del Programa d'Enviament

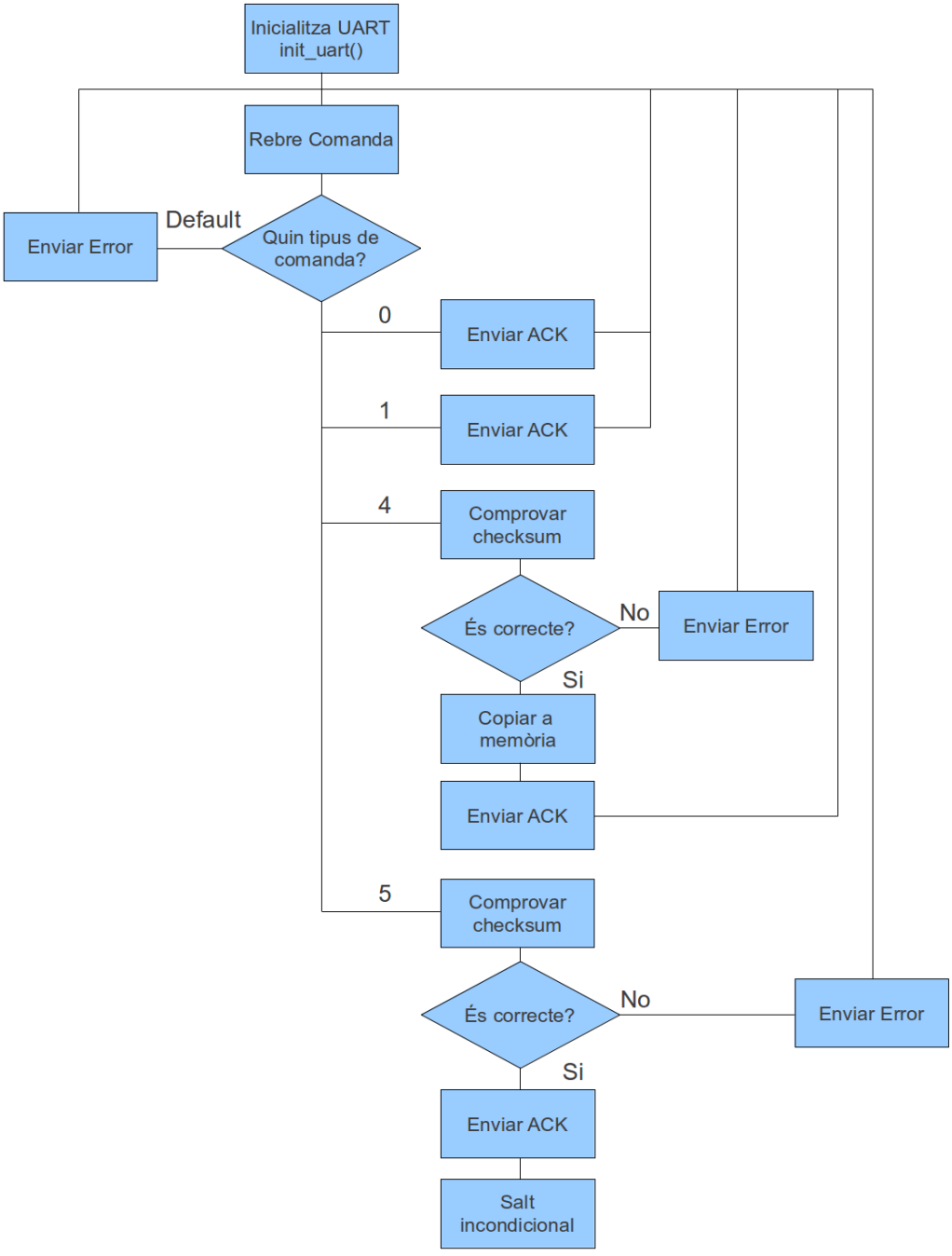


Figura 3.16: Diagrama de Flux del Firmware del NIOSII

# Capítol 4

# Tests i Resultats

## 4.1 Kits de Desenvolupament

Per les parts corresponents a la descàrrega del Firmware de l'FX2 i la configuració de la FPGA mitjançant el Passive Serial s'ha fet servir la placa de desenvolupament CY3681 de Cypress [figura 4.1]. Aquesta placa conté entre d'altres coses una entrada USB, un xip FX2 connectat a diversos ports de propòsit general, i connectat també a un CPLD que fa d'enllaç amb altres elements com un Display de 7 segments o un conjunt de polsadors.

Amb aquesta placa el que s'ha fet és descarregar un Firmware d'exemple facilitat per Cypress utilitzant les Vendor Requests exposades al capítol 3.

Per altra banda, per implementar el SoftCore NIOSII i carregar el codi a executar en última instància s'ha fet servir una placa de desenvolupament d'Altera anomenada DE2 [figura 4.2]. Conté una FPGA CycloneII connectada, entre d'altres coses, a un port d'interruptors que servirà per provar un programa amb interrupcions, un port de Leds per veure que realment està funcionant el codi que utilitza i un port sèrie RS232 per a transferir el codi.

Aquesta placa ha estat útil per a provar el Disseny del NIOSII, i implementar el Firmware que ha d'anar inclòs a la memòria RAM implementada a la FPGA que s'encarregarà de rebre, des del PC, el fitxer SREC utilitzant el port sèrie RS232, i desant-lo a les posicions de memòria que correspongui.

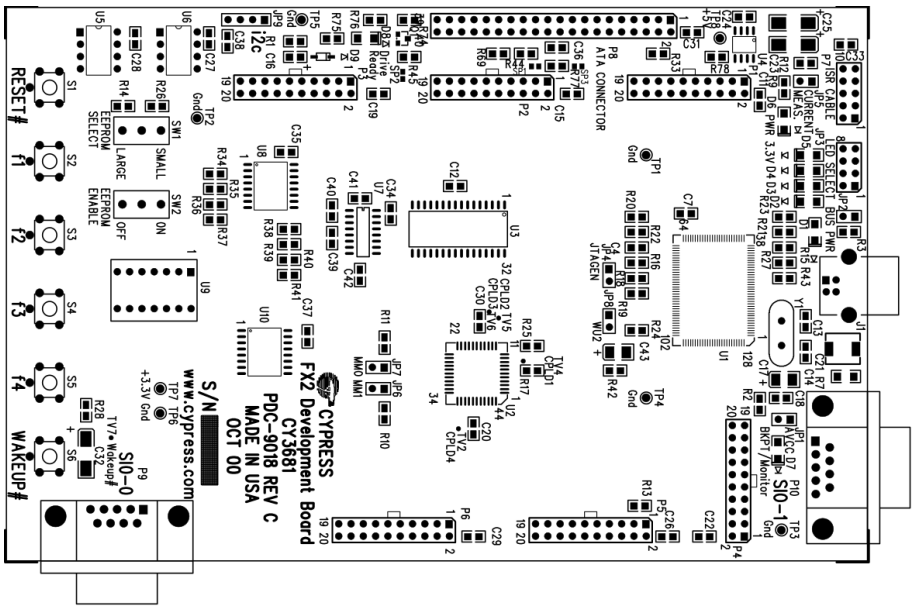


Figura 4.1: Kit de Desenvolupament de Cypress CY3681

Per últim, la placa de l'esquema que s'ha seguit durant tot el projecte (figura 1.1 del capítol 1) és la placa de prototip del producte definitiu, i ha servit per provar totes les parts implementades amb els kits de desenvolupament anteriors, i provar la part del Passive Serial, en la que el controlador FX2 s'encarrega de configurar el contingut de la FPGA fent servir el fitxer RBF que rep pel port USB. A més s'ha fet servir un port de propòsit general d'entrada/sortida de la placa per a connectar-hi un petit mòdul amb 3 leds i un polsador per veure que el Firmware final que executarà el NIOSII funciona correctament.

## 4.2 Entorns de Desenvolupament

Per implementar el Firmware final que ha d'executar el controlador FX2, i adaptar el codi facilitat per Cypress per a provar el seu Kit [figura 4.1], s'ha fet servir l'entorn Keil uVision2, que és el que proporciona per a aquest propòsit. A més per implementar aquests Firmwares s'han fet servir les Macros i llibreries facilitades per Cypress amb el seu Kit.

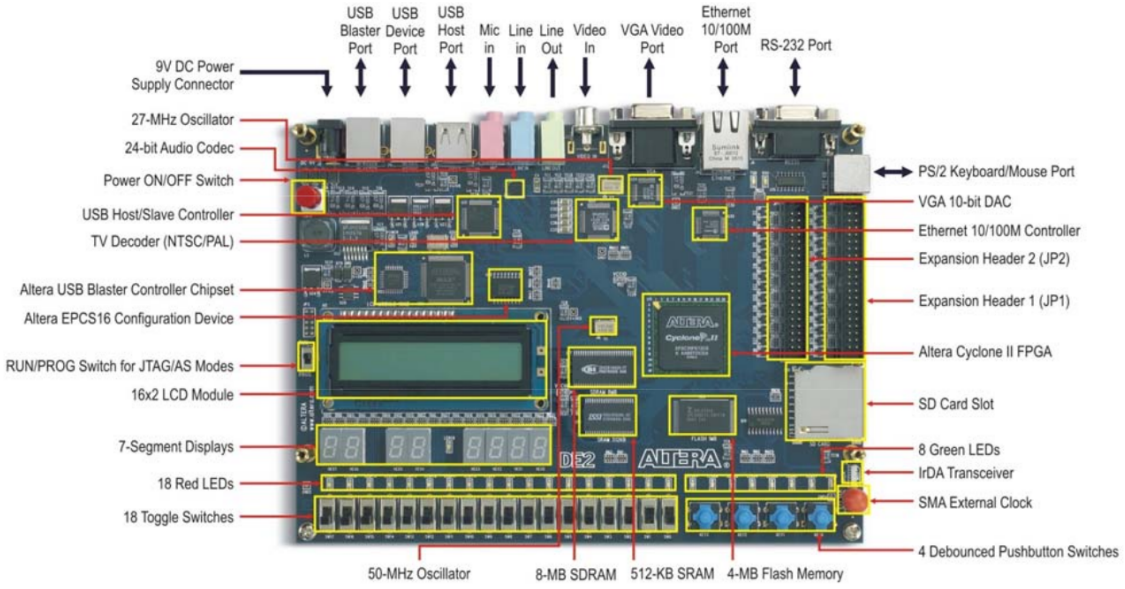


Figura 4.2: Kit de Desenvolupament DE2

El disseny del NIOSII per a fer les proves amb la placa DE2 d'Altera, s'ha fet mitjançant el seu Software QuartusII 9.0sp2 i la eina SOPC Builder.

La implementació del codi en C que executarà el NIOSII s'ha fet mitjançant el NIOSII 9.0 IDE, que és una eina per a l'Eclipse que permet compilar codi en C per a executar en aquest SoftCore utilitzant unes Macros i unes llibreries que et genera automàticament segons el sistema que s'implementi (system.h i altera_avalon_pio_regs.h). Aquest Software genera un fitxer executable d'extensió ELF que es transformarà en un SREC mitjançant una aplicació d'Altera anomenada nios2-elf-objcopy, que dona la possibilitat de manipular alguns paràmetres del format del fitxer, com ara la llargada de les línies o les adreces de destí on ha d'anar el codi.

Finalment per als programes que s'executen des del PC s'ha fet servir l'entorn DEV-C++ per la seva simplicitat. A més, per al port sèrie RS232 s'ha utilitzat la llibreria windows.h, i per a l'USB la llibreria genèrica usb.h.

| Eines | Utilització |
|---|---|
| Keil uVision2 | Implementació i compilació del codi que executa l'FX2. |
| QuartusII<br>- SOPC Builder<br>- Programmer | Sintetització del Sistema NIOSII.<br>Disseny del Sistema NIOSII.<br>Descàrrega del Sistema NIOSII fent servir el JTAG. |
| NIOSII 9.0 IDE | Implementació i compilació del codi a executar al NIOSII. |
| nios2-elf-objcopy | Conversió de l'executable ELF al fitxer de text SREC. |
| DEV-C++ | Implementació de les aplicacions a executar des del PC. |

Taula 4.1: Resum dels entorns utilitzats

## 4.3 Aplicacions de Test

### 4.3.1 Aplicacions de test sobre l'FX2

#### EZ-USB Control Panel

És una aplicació per al PC facilitada per Cypress, i es pot fer servir per a descarregar a memòria un fitxer HEX que contingui qualsevol codi compilat per a l'FX2. A més permet enviar comandes preestablertes com ara la comanda que demana la identificació del dispositiu o la configuració dels EndPoints.

#### IoDev

IoDev és un Firmware d'exemple per a executar al Kit de desenvolupament de Cypress, i el seu funcionament consisteix en incrementar, decrementar, posar a 0 i posar a F el Display de set segments de la placa.

Aquest Firmware s'ha fet servir per comprovar que l'enviament des del PC del fitxer HEX fent servir les Vendor Requests implementades per Cypress fos correcte.

#### BulkLoop

BulkLoop és un altre Firmware d'exemple per a l'FX2 que fa un Echo USB. Més concretament, torna pels EndPoints 6 i 8 el que li arriba pel 2 i el 4 respectivament. Però el que ha resultat interessant per a desenvolupar aquest projecte, és que una

part del seu codi conté el Switch que detecta quin tipus de Vendor Request arriba per l'EndPoint 0 i actuar en conseqüència.

Així que aquest Firmware s'ha fet servir com a plantilla per a implementar les 3 Vendor Request necessàries per a configurar la FPGA amb el Passive Serial, i la corresponent a l'enviament del fitxer SREC al NIOSII, i comprovar que el seu funcionament era correcte.

### 4.3.2 Aplicacions de test sobre la FPGA

L'objectiu de les aplicacions que s'exposen en aquest apartat és el d'avaluar la funcionalitat, i no pas les prestacions.

#### Disseny del Sistema NIOSII

Per a provar la FPGA s'ha fet el disseny d'un sistema NIOSII com el de la figura 4.3 que consta dels següents elements:

- El processador NIOSII: És el model més senzill, ocupa entre 600 i 700 Elements lògics.
- Timer: Un timer simple per a poder controlar interrupcions per temps.
- PLL: El PLL serveix per a gestionar diferents rellotges al sistema a partir d'un rellotge extern. En aquest cas li arriba un rellotge extern de 50MHz, i en genera un amb una fase de -3ns per al mòdul de SDRAM i un altre sense desplaçament per a la resta del sistema. Això es fa per a sincronitzar la memòria SDRAM amb la resta d'elements.
- Controlador SDRAM: El mòdul per gestionar una memòria SDRAM de 8MB que hi ha a la placa DE2. És on es guardarà el Firmware final.
- RAM1: Aquest mòdul de RAM conté implícitament el Boot Code que ha d'executar el NIOSII per començar a funcionar. Aquest Boot Code és el corresponent al programa de recepció explicat al capítol 3.

- RAM2: En aquest mòdul es guardaran els vectors d'interrupció del Firmware que es vol enviar al NIOSII.
- JTAG: És un mòdul que serveix per a connectar el sistema amb l'ordinador i poder veure el seu funcionament. És una eina de Debug.
- UART: És el mòdul corresponent al port sèrie RS232 per on es rebrà el fitxer.
- 2 I/O Parallel Ports: Són dos ports per a comunicar-se amb elements de la placa, en aquest cas per als interruptors i per als Leds. Simplement per a veure que el funcionament del programa final és el correcte.
- Bus AVALON: És el bus propi d'Altera per a interconnectar tots els mòduls anteriors.

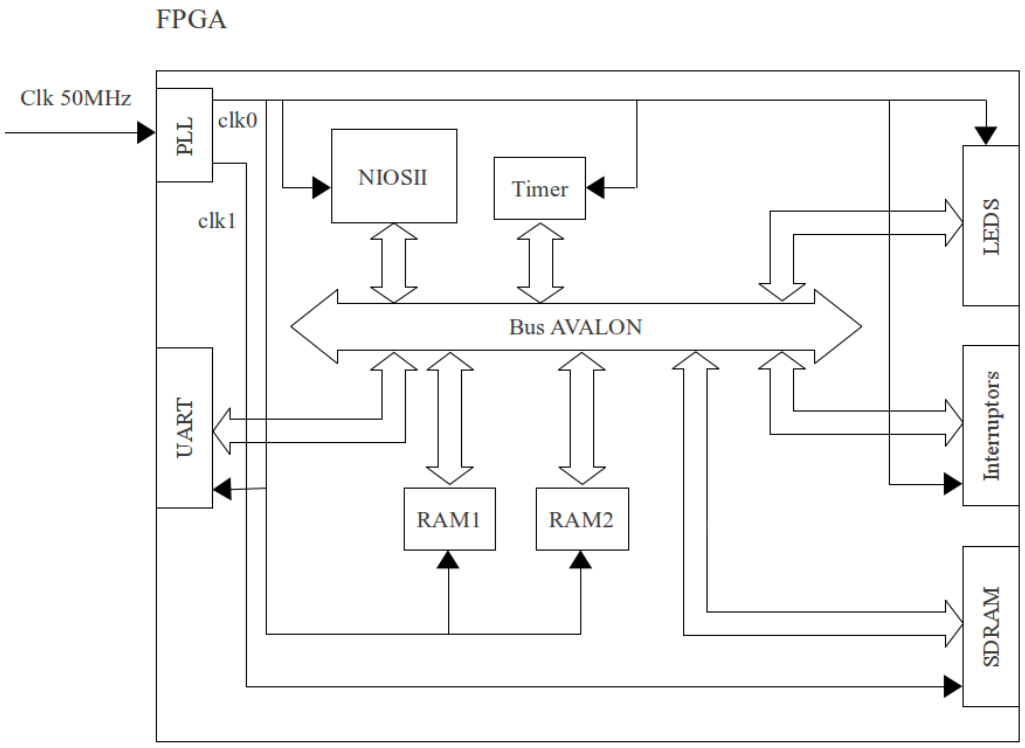


Figura 4.3: Disseny del Sistema a la FPGA

#### Firmwares de test del NIOSII

S'han implementat 3 tests per comprovar que el sistema NIOSII funcioni correctament. El primer simplement encén una combinació de Leds a la placa, el segon és un comptador que es va incrementant cada mig segon, i aquest comptador es treu pels Leds, d'aquesta manera es pot comprovar que el timer i les funcions d'espera funcionen correctament. I finalment un tercer test que cada vegada que li arriba la interrupció d'un interruptor canvia el sentit del compte, de manera que es prova que les interrupcions funcionen sense problemes.

Aquests 3 tests, en primer lloc s'han posat com a Firmware implícit a la RAM per a comprovar que un cop configurada la FPGA, el sistema arrenca de forma correcta executant cadascun d'aquests 2 tests, confirmant d'aquesta manera que el protocol Passive Serial funciona correctament. I finalment es van convertir en fitxers SREC per a enviar-lo des del PC a l'FX2, i des de l'FX2 al NIOSII a través del port sèrie per comprovar que la fase final acabés amb èxit.

## 4.4 Resultats

Tot i que l'objectiu és la optimització de l'arrencada del sistema des del punt de vista de l'ús de memòria, és interessant comprovar si aquesta optimització és factible des del punt de vista temporal. Per això s'ha mesurat el temps d'arrencada de cada component utilitzant les memòries no volàtils, i el temps amb el procediment desenvolupat durant aquest projecte. Per tant, aquesta secció del capítol serveix per avaluar les prestacions.

### 4.4.1 FX2

#### En mode C2

Tenint un Firmware de 4 KBytes allotjat a la memòria EEPROM, la descàrrega ha trigat al voltant de 320 ms.

La freqüència del rellotge que marca la lectura de les dades és de 100 KHz, per tant, 4 KBytes / 100 KHz = 328 ms.

#### En mode No EEPROM

En mode "No EEPROM", és a dir, sense cap memòria no volàtil connectada, i descarregant un Firmware de 9,7 KBytes des del PC fent servir la Vendor Request de Cypress, ha trigat 47 ms.

La mesura s'ha pres fent servir la llibreria time.h.

La transferència és així de ràpida perquè el port USB del controlador FX2 és capaç d'arribar a 480 Mbps, que és una freqüència molt més alta que no pas la que fa servir el rellotge per a llegir de la memòria EEPROM.

### 4.4.2 Configuració de la FPGA

Les figures a les que es fa referència en aquest apartat estan totes a l'Apèndix A.

#### JTAG

Utilitzant el mètode JTAG i descarregant un fitxer de configuració de 360 KBytes triga uns 650 ms. A la figura A.1 es veu el senyal de rellotge del procés complet.

A la figura A.2 es pot veure el principi del protocol, on el cicle de rellotge és més llarg i després unes quantes trames de Bytes, i es pot veure que des del començament d'un, fins al començament de l'altre triga uns 1,75 $\mu$s.

Finalment a la figura A.3 es pot veure un cicle de rellotge, i es veu que el període és de 800 ns.

Per tant, 360 KBytes x 1,75 $\mu$s per Byte dona 645,12 ms estrictament de transferència de dades.

#### Active Serial

Utilitzant l'Active Serial i descarregant un codi de 140 KBytes, des d'una memòria Flash sèrie triga al voltant de 130 ms. A la figura A.4 es veu el senyal de rellotge durant tot el procés. Com es pot veure a la figura, tot i que el temps total és de 130 ms, durant 100 ms el rellotge fixat a 1. Aquest és un temps d'espera que es pren la FPGA per a que el voltatge de tot el sistema s'estabilitzi abans de començar.

A la figura A.5 està ampliat només la part del procés en la que es transfereixen les dades, i com es pot veure triga uns 33 ms. En aquest cas no hi ha cap tipus d'espera entre Byte i Byte.

La figura A.6 és l'ampliació del senyal de rellotge que marca la tassa de transferència del procés, i té un període de 30 ns.

Si es fan els càlculs, són 140 KBytes x 30 ns per bit, dona un total de 34 ms a la fase de dades.

**Passive Serial**

Per acabar amb els temps de configuració de la FPGA falta mesurar els temps del Passive Serial. A la figura A.7 es veu la línia de rellotge durant tot el procés de descàrrega d'un fitxer RBF de 156 KBytes, i es pot veure com la configuració triga 4,5 s.

A la figura A.8 es veu el senyal de rellotge de color blau, i el senyal de dades de color groc, des de que comença un Byte fins al següent, i triga 23 $\mu$s.

Per últim a la figura a.9 es pot veure un cicle de rellotge. Es pot veure que el període és de 2,25 $\mu$s, però que els semicicles no són regulars, sinó que és manté el rellotge a 1 durant aproximadament 750 ns, i a 0 1,5 $\mu$s.

Fent els càlculs dona que hauria de trigar, 156 KBytes x 23 $\mu$s igual a 3,67 segons, però s'ha de tenir en compte que entre línia i línia del fitxer SREC hi ha un espai més gran, que és el que fa que hi hagi aquest segon de diferència.

### 4.4.3 NIOSII

Finalment, per a la última fase del projecte, s'ha mesurat el temps que triga en enviar-se un fitxer sencer SREC de 4'3 KBytes. Com el port sèrie no té una línia de rellotge, s'ha fet mesurant la línia de dades, i per això no s'ha pogut mesurar un Byte sol.

Com es veu a la figura A.10, el temps total és de 360 ms.

Com el port sèrie està configurat a 115200 bauds, serien 4'3 KBytes / 115200 bauds = 305 ms, al que se li hauria de sumar els càlculs que es fan després de

llegir cada línia del fitxer SREC.

### 4.4.4 Temps Total

A la taula 4.2 està detallat el temps d'arrencada de cada component, i del sistema final. S'ha de tenir en compte que fent servir les memòries no volàtils integrades a la placa, tots el elements comencen l'arrencada al mateix moment, mentre que fent servir el procés descrit en aquest projecte, l'arrencada es du a terme en mode de cascada, és a dir, es van inicialitzant els dispositiu de forma seqüencial, i cada dispositiu arrenca recolzant-se en l'anterior.

| **Procés d'arrencada** | **Tassa de Transferència** | **Mida del Fitxer** | **Temps** |
|---|---|---|---|
| Lectura del Firmware de l'FX2 de la EEPROM | 12'5 Bytes/ms | 9'7 KBytes | 794'6 ms |
| Configuració de la FPGA (Active Serial + Firmware NIOSII) | 1'08 KBytes/ms | 160'3 KBytes | 148'4 ms |
| Temps Total | | | 794'6 ms |
| Descàrrega del Firmware de l'FX2 (Vendor Request) | 211'3 Bytes/ms | 9'7 KBytes | 47 ms |
| Configuració de la FPGA (Passive Serial) | 34'6 KBytes/s | 156 KBytes | 4'5 s |
| Descàrrega del Firmware del NIOSII | 12'23 Bytes/s | 4'3 KBytes | 360 ms |
| Temps Total | | | 4'9 s |

Taula 4.2: Temps d'arrencada

# Capítol 5

# Conclusions

Com a conclusions, en aquest capítol es farà un repàs als objectius assolits, a les possibles millores en un treball futur i s'acabarà valorant si s'ha complert la planificació temporal plantejada inicialment.

## 5.1 Compliment dels Objectius

L'objectiu general del projecte ha estat el d'eliminar totes les memòries no volàtils d'una placa com la de l'esquema de la figura 1.1 del capítol 1. Més concretament, el projecte s'havia dividit en 3 objectius més acotats:

- Eliminar la memòria que conté el Firmware del controlador USB: Aquest objectiu s'ha assolit fent que el controlador s'identifiqui en mode "No EEPROM", i descarregant el Firmware a través del port USB mitjançant una Vendor Request implementada per Cypress. D'aquesta manera s'ha pogut prescindir de la memòria EEPROM de la placa.

- Configuració de la FPGA: Per a aquesta part s'ha implementat un Firmware per a l'FX2 que sigui capaç de dur a terme el protocol Passive Serial, rebent pel port USB les dades del fitxer de configuració RBF emmagatzemat al PC, a través d'unes Vendor Requests implementades específicament per a

aquest propòsit. D'aquesta manera s'ha eliminat la part de la memòria Flash que contenia la informació de configuració.

- Eliminar la memòria que conté el Firmware del SoftCore: Finalment aquest tercer objectiu s'ha assolit implementant un protocol de comunicació que permeti enviar unes comandes des del PC al NIOSII, i desar la informació continguda al fitxer SREC en la memòria SDRAM externa. Per a fer això, s'ha implementat també una Vendor Request per a l'FX2 que rebi les dades enviades des del PC, i les reenviï a través del port sèrie que el comunica amb el NIOSII. I d'aquesta manera es pot prescindir de tota la memòria Flash de la placa de l'exemple.

Finalment s'han agrupat els tres procediments per a fer arrencar una placa amb tots aquests elements en forma de cascada, és a dir, un darrere de l'altre, aconseguint així l'objectiu general del projecte, demostrant que és possible arrencar un sistema sencer descarregant tota la informació necessària des del PC al que estigui connectat.

Tot i que s'ha demostrat l'assoliment dels objectius, és important no perdre de vista la motivació del projecte, que bàsicament pretenia abaratir costos de fabricació prescindint d'aquests elements, per tant s'ha de fer un repàs de l'estalvi aconseguit per veure si és significatiu.

En una placa com la de l'exemple que s'ha seguit durant el projecte, els components més costosos són el controlador FX2, la EEPROM de 8 KBytes, la FPGA CycloneII, la memòria Flash de 4 Mbits i la memòria SDRAM de 256 MBytes.

Els preus de cadascun dels components són:

-FX2 (CY7C68013A): 7'23 €

-CycloneII (EP3C10F256C8N): 19'88 €

-SDRAM (48LC16M16A2): 8'29920 €

-Flash (M25P40): 1'0906 €

-EEPROM (24LC64): 0'54 €

Per tant, el preu total és de 37'0398 €, i si es descompten les dues memòries no volàtils costa 35'4092 €, que és un 4'40% més econòmic, i donat que aquest tipus de productes es fabriquen en llargues tirades, suposa un estalvi important.

## 5.2 Ampliacions o millores

La FPGA CycloneII que s'ha fet servir en aquest projecte només disposa dels 3 mètodes de configuració exposats al capítol 2, però el següent model, la CycloneI-II, té dos mètodes nous, l'Active Parallel i el Passive Parallel, que com indica el nom, serveixen per enviar els bits de configuració en paral·lel, de manera que la configuració seria més ràpida, i donat que aquesta part és la que ocupa la gran majoria del temps, seria una bona millora de prestacions.

A més, un problema que hi ha quan es munten les plaques de sistemes encastats de forma massiva, és que hi ha un percentatge de plaques que queden mal soldades, de manera que a simple vista és difícil de detectar on està el problema, i per tant, d'arreglar-lo. Adaptant una mica el sistema d'arrencada exposat en aquest projecte, és pot fer una aplicació de test que permetés anar arrencant els elements un per un per tal de detectar quin és el que falla, i un cop detectat tornar a fer les soldadures corresponents.

## 5.3 Compliment de la Planificació Temporal

Des de bon principi ja era conscient de la dificultat de mantenir aquesta planificació, degut a la càrrega de feina de les matèries del segon semestre i dels exàmens. I a més, quan va arribar l'hora de provar el Passive Serial hi va haver alguns problemes amb les plaques, de manera que va fer que fos impossible acabar-ho pel setembre. A partir d'aquell moment m'ho vaig pendre amb més calma i vaig acabar de planificar el que quedava fins al Gener.

A la taula 5.1 es pot veure la planificació a partir d'aquell moment.

| **Tasques** | **Duració** | **Inici** | **Fi** |
|---|---|---|---|
| *Informe previ* | | | |
| - Redacció | 5 dies | 03/01/11 | 07/01/11 |
| - Entrega | 1 dia | 10/01/11 | 10/01/11 |
| *Estudi de conceptes bàsics* | | | |
| - Funcionament bàsic d'un controlador d'USB | 14 dies | 21/02/11 | 6/03/11 |
| - Conceptes bàsics d'una FPGA | 7 dies | 06/03/11 | 13/03/11 |
| *Sistema NIOSII* | | | |
| - Disseny del Sistema | 14 dies | 14/03/11 | 27/03/11 |
| - Disseny del Protocol | 21 dies | 28/03/11 | 17/04/11 |
| - Elaboració de la memòria | 21 dies | 18/04/11 | 08/05/11 |
| *FX2* | | | |
| - Descarregar el Firmware | 21 dies | 16/05/11 | 05/06/11 |
| - Implementació de Vendor Requests | 14 dies | 11/07/11 | 24/07/11 |
| - Elaboració de la memòria | 14 dies | 25/07/11 | 07/08/11 |
| *Placa Final* | | | |
| - Vendor Request per al Passive Serial | 21 dies | 22/08/11 | 11/09/11 |
| - Adaptació del codi anterior | 21 dies | 12/09/11 | 02/10/11 |
| - Elaboració de la memòria | 21 dies | 03/10/11 | 23/10/11 |
| *Mesura de temps* | 4 dies | *09/01/12* | *12/01/12* |
| *Elaboració de la memòria* | 19 dies | *13/01/11* | *31/01/11* |

Taula 5.1: Planificació inicial del projecte.

# Apèndix

# Apèndix A

# Gràfiques de l'Oscil·loscopi

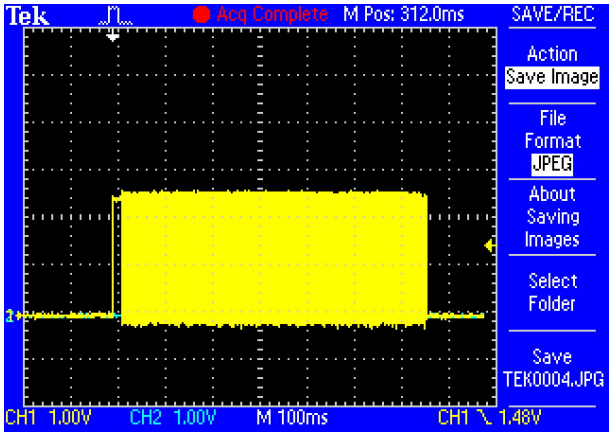


Figura A.1: Temps total del JTAG

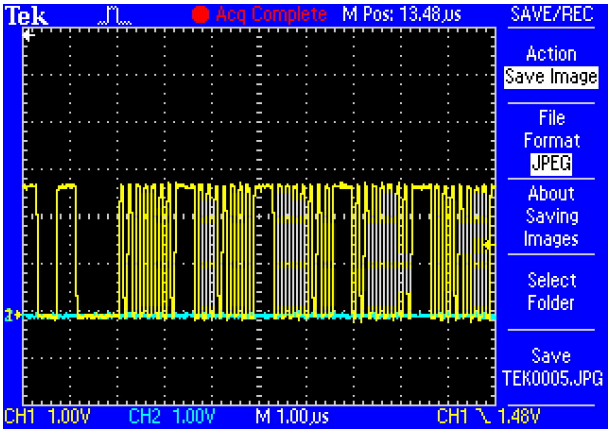


Figura A.2: Temps per Byte del JTAG

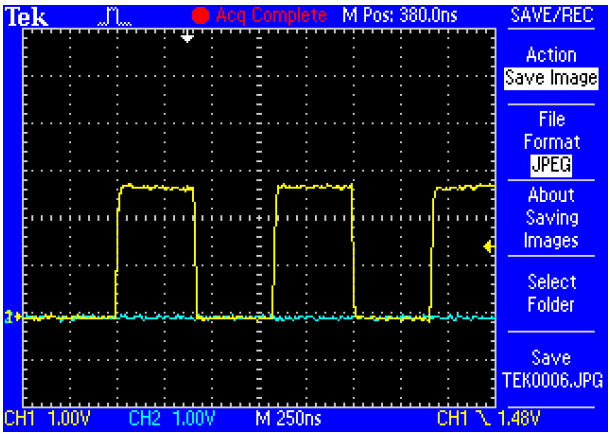


Figura A.3: Cicle de rellotge del JTAG

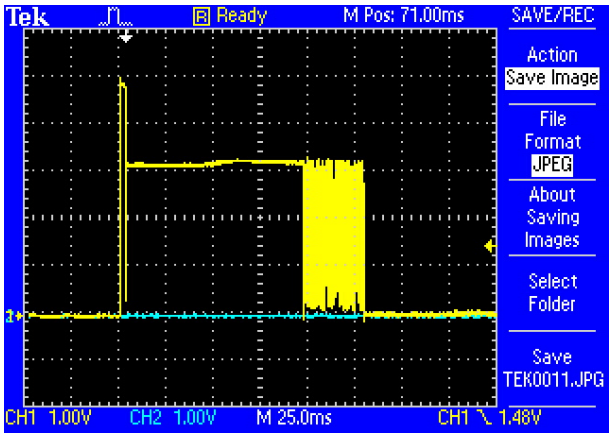


Figura A.4: Temps total de l'Active Serial

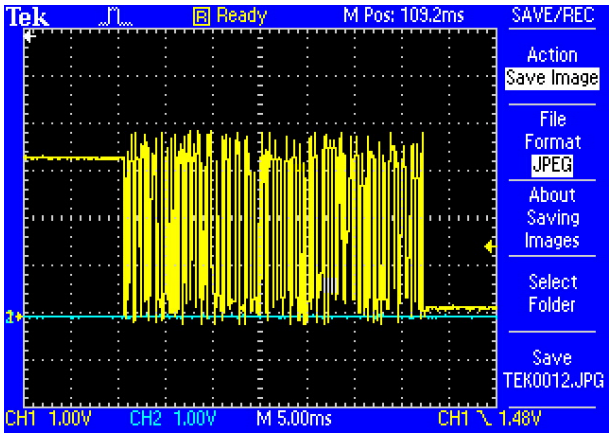


Figura A.5: Transferència de dades de l'Active Serial

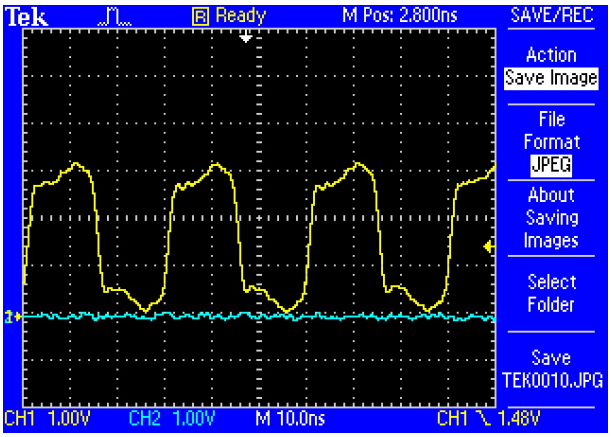


Figura A.6: Cicle de rellotge de l'Active Serial

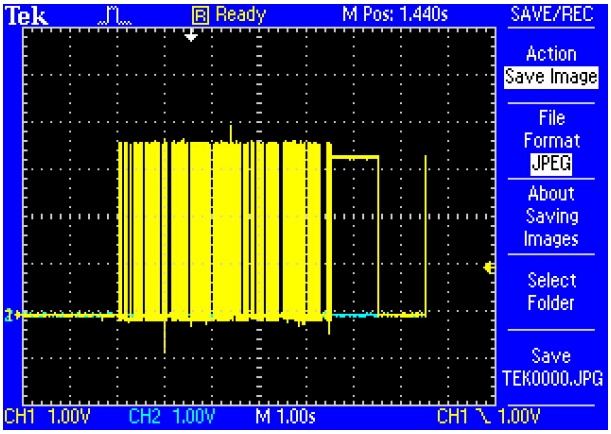


Figura A.7: Temps total del Passive Serial

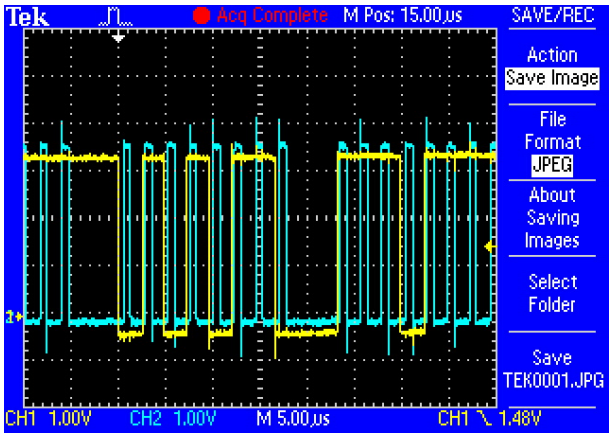


Figura A.8: Temps de transferència d'1 Byte del Passive Serial

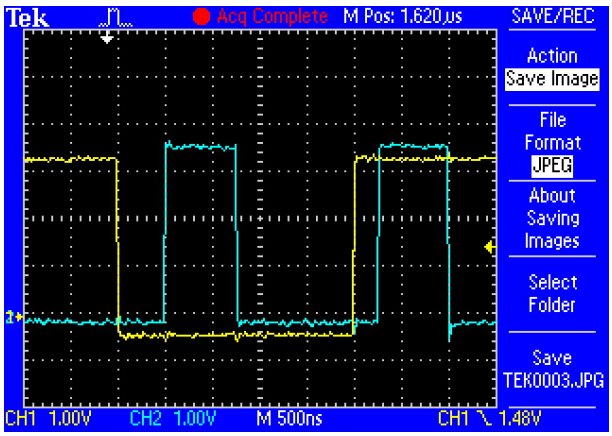


Figura A.9: Cicle de rellotge del Passive Serial

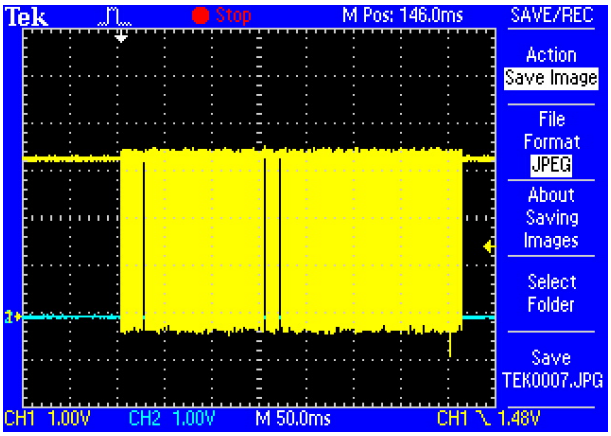


Figura A.10: Temps total de l'enviament de l'SREC

# Bibliografia


[DKUG11] EZ-USB Development Kit User Guide, Febrer 2011.

[TRM10] EZ-USB Technical Reference Manual, Setembre 2010.

[alciro] alciro Books Science Engineering & Information Technology, Setembre 2011.
<http://www.alciro.org/>

[LUSB] libusbwin32 Documentation, Febrer 2012.
<http://sourceforge.net/apps/trac/libusb-win32/wiki/libusbwin32_documentation>

[DE206] DE2 Development and Education Board, Novembre 2006.

[CDH08] CycloneII Device Handbook, Febrer 2008.

[NPR10] NIOSII Processor Reference, Decembre 2010.

[EPIP11] Embedded Peripherals IP, Febrer 2011.

[NFP10] NIOSII Flash Programmer, Gener 2010.

[CPP12] cplusplus.com, Gener 2012.
<http://www.cplusplus.com/>

Firmat: Hugo Lluelles Martínez

Bellaterra, Febrer de 2012

## Resum

L'objectiu d'aquest projecte és el d'optimitzar l'arrencada d'un sistema encastat amb diferents elements de còmput des del punt de vista de l'estalvi de memòria, per tal de reduir el cost de la fabricació de les plaques. El que s'ha fet és aprofitar una connexió USB per a connectar la placa a un host, en aquest cas un PC, i descarregar via USB tota la informació necessària per a arrencar cadascun dels components de la placa, prescindint d'aquesta manera de les memòries no volàtils de les que depenien originalment.

## Resumen

El objetivo de este proyecto es el de optimizar el arranque de un sistema empotrado con diferentes elementos de cómputo desde el punto de vista del ahorro de memoria, para reducir costes en la fabricación de las placas. Lo que se ha hecho es aprovechar una conexión USB para conectar la placa a un Host, en este caso un PC, y descargar via USB toda la información necesaria para arrancar cada uno de los componentes de la placa, prescindiendo de esta manera de las memorias no volátiles de las que dependían originalmente.

## Abstract

The aim of this project is to optimaze the boot process with different computing elements from the point of view of memory saving to reduce the manufacturing board's costs. What has been done is using a USB connection to connect the board to a host, in this case a PC via USB, and download all the information needed to start each of the components on the board, dispensing therefore the nonvolatile memories which originally depended on.