CREACIÓ D' *IP CORES* EN UNA PLATAFORMA NIOS: METODOLOGIA DE DISSENY

Memòria del Projecte Fi de Carrera
d' Enginyeria en Informàtica
realitzat per
Antoni Costa Sanfeliu
i dirigit per
Joan Oliver Malagelada
Bellaterra, 15 de Juny de 2007

El sotasignat, Joan Oliver Malagelada

Professor de l' Escola Tècnica Superior d' Enginyeria de la U.A.B,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Antoni Costa Sanfeliu.

I per tal que consti firma la present.

Signat:

Bellaterra, 15 de Juny de 2007

AGRAÏMENTS

En primer lloc, m'agradaria agrair a Joan Oliver Malagelada, el temps que ha dedicat en aclarir-me i resoldre'm dubtes, així com les reunions que ha hagut d'adaptar al meu més que complex horari laboral. La visió del projecte, la guia i el seguiment del mateix han estat claus en el desenvolupament del projecte. Així doncs, moltes gràcies.

D'altra banda, també m'agradaria fer extensiu aquest agraïment a tots els professors de la Titulació, ja que cadascun ha aportat el seu grà de sorra i d'una manera o altre han contribuït a què aquest projecte hagi estat possible.

Tampoc vull oblidar-me dels meus companys de carrera, amb els quals hem compartit molts moments agradables i també difícils en èpoques d'exàmens i d'entrega de pràctiques, dedicant molts esforços que avui es veuen recompensats.

Agrair també a tots els meus companys i amics de Manresa, així com la meva família que m'han recolzat des del primer dia, tan en la carrera com en la realització d'aquest projecte. Moltes gràcies a tots ells, per aguantar els mals humors i ser pacients amb mi quan feia falta.

Finalment, unes paraules pel Ricoh Manresa que de nou aquest any torna a ser equip ACB i ha ajudat a amenitzar les males estones.

ÍNDEX

1. INTRODUCCIÓ I OBJECTIUS

NIOS és el processador que Altera empra en els seus dissenys SOC (*System On Chip*). Per tal de facilitar-ne el seu ús, Altera també proporciona una plataforma de desenvolupament SOPC (*System On Programable Chip*) que agilitza enormement el disseny d'aquests sistemes. Així doncs, aquest projecte està centrat en la definició metodològica per a la creació d' *IP Cores* en una plataforma NIOS.

Aquest projecte analitzarà l'entorn, i un cop havent definit una metodologia de treball adequada, (amb l'objectiu de validar la metodologia de disseny), es crearan dos nous cores a partir d'aquesta, que es gestionaran a alt nivell. La interacció del hardware i el software, així com la utilització de l'entorn d'Altera seran els punts clau en el projecte.

1.1 Introducció

S'entén per *IP Core* (*Intellectual Property Core*) , un bloc lògic o de dades que és utilitzat en FPGA (F*ield Programmable Gate Array*) o en aplicacions específiques de circuits integrats (ASIC).

L'essència de la utilització d'aquests elements és la reutilització del seu disseny. Els *IP Cores* són part del creixent augment de l' automatització dins de la indústria, que tendeix cap al repetit ús dels components prèviament dissenyats.

Idealment, un *IP Core* ha de ser completament portable a qualsevol altra tecnologia, o si més no, fàcilment integrable a la tecnologia emprada per l'usuari o client. La metodologia de disseny també serà molt important, ja que com més genèrica sigui més portable serà.

Es pot considerar que un component és un *IP Core* si té com a mínim unes 6000 portes lògiques. Així doncs, es pot trobar amb components petits com: *Universal Asynchronous Reciver/Transmitter* (UARTs), *Ethernet* controllers, i PCI *interfaces* que són exemples de *IP Cores.* O bé, components com: C*entral Processing Units* (CPUs), *Digital Signal Processor* (DSP), i *drivers* en general, com a exemples d'*IP Cores* de més magnitud o importància.

Els *IP Cores* estan dividits en tres categories: els *hard cores*, els *firm cores* i els *soft cores.* Els *hard cores* són la manifestació física del disseny IP. A tall d'informació, cal destacar que són els millors per aplicacions plug-and-play, tot i que són menys portables i flexibles que les altres dues. Com les *hard cores*, els *firm* cores són utilitzats pel posicionament de dades i són configurables per a vàries aplicacions. Pel que fa als *soft cores*, cal mencionar-ne els següents aspectes. Primerament, cal dir que són els més flexibles de tots tres. I en segon lloc, que estan definits a través d'una descripció de llenguatge hardware (HDL) o bé, a través d'una llista de portes lògiques i associacions d' interconnexions que creen un circuit integrat. A més a més, aquest últim, serà el que s'utilitzi en la realització d'aquest projecte.

1.2 Objectius

L'objectiu principal d'aquest projecte és l'estudi i l'ús d'una metodologia per a la construcció d' *IP cores*. Aquest estudi tindrà un component pràctic, ja que un cop definida aquesta metodologia, es seguirà creant dos *IP cores*, que seran provats sobre la plataforma NIOS.

Molts dispositius d'avui en dia vénen programats amb la lògica que aporta el fabricant, mentre que en altres ocasions es permet al programador retocar-ne la lògica per a tenir-ne un major control. No obstant, hi ha situacions en les quals és necessari,

sigui per un motiu o per l'altre, la creació d'un propi *driver* per a ser capaços de gestionar un dispositiu aliè al sistema, i que per tant, no s'hi pot comunicar.

Així doncs, el què es pretén en aquest projecte és que davant la necessitat de creació d'un *driver* que no es té, establir una metodologia a seguir per a la realització d'aquest *driver* o *IP Core*, i més concretament, per a plataformes NIOS.

Una vegada definida aquesta metodologia s'haurà de corroborar que és útil. La plataforma escollida per a determinar-ho serà la plataforma NIOSII d'Altera. D'aquesta manera, tenint aquesta plataforma s'implementaran dos *IP Cores*: PWM (*Pulse Width Modulation*) i I2C (*Inter-Integrated Circuit*). Aquests dos *drivers* seran els quals es construiran i se'n provarà el seu funcionament.

El funcionament dels dos *IP Cores*, es provarà a través de la placa UP3. S'utilitzarà un servo motor o actuador i un sensor d' I2C. Tots dos elements es gestionaran des de l'alt nivell a través d'una lògica de control adequada. D'aquesta manera, el projecte necessita de la utilització de sensors i actuadors per a comprovar el funcionament dels *drivers*.

Un sensor és un tipus de transductor que transforma les magnituds que vol mesurar, en altres que faciliten la seva mesura. Poden ser d'indicació directa o poden estar connectats a un convertidor analògic o digital, el qual facilitarà que els valors que es llegeixin siguin més comprensibles per un ésser humà. Així doncs, i com el seu nom indica, són dispositius que detecten i adapten el senyal que generen perquè un altre element la pugui llegir.

Quant als actuadors, cal dir que són uns mecanismes a través dels quals un agent pot influir en el seu entorn. Aquest agent pot tractar-se d'un element artificial o de tipus autònom. D'aquesta manera, un mecanisme que posa quelcom en acció autònoma és

denominat un actuador. També s'acostuma a definir com aquell element d'un sistema de control que converteix els senyals en accions físiques.

D'aquesta manera, els objectius concrets del projecte són:

- Anàlisi de l'entorn presentat per Altera en el desenvolupament d' *IP Cores.*
- Presentació d'una metodologia de disseny per a la construcció d' *IP Cores*.
- Seguint aquesta metodologia, creació de dos *IP Cores*: PWM i I2C.
- Comprovació pràctica de la creació d'aquests *IP Cores*.

El projecte pretén analitzar a fons la viabilitat en la realització d' *IP Cores* emprant l'entorn proporcionat per Altera. Així doncs, es tractarà amb la interacció de hardware i software, dos camps que en aquesta ciència estan condemnats a viure plegats. Es programaran uns elements hardware i es gestionaran a través d'un software. D'aquesta manera, s'observa que més aviat és un projecte on es parla i es toca l'essència de la Informàtica.

2. PLANIFICACIÓ, VIABILITAT I CRONOLOGIA

2.1 Especificació de requisits. Anàlisi funcional detallat

Les etapes que es seguiran en el desenvolupament del projecte són:

1. Estudi de tots els elements necessaris per al funcionament de la placa UP3. El cap de projecte, estudiarà i comprendrà els elements d'aquesta placa que seran claus a l'hora de la seva utilització pràctica. Aquest estudi també requerirà saber llenguatges HDL i C++.
2. El coneixement detallat de la placa implica fer ús d'aquesta. Fent això, es comprendrà el funcionament de tots i cada un dels elements de la placa.
3. Personalització d'aquesta metodologia per a la placa UP3 d'Altera per a la creació d'un nou *IP Core*. Es fixarà un procediment estàndard a seguir en el cas que qualsevol persona vulgui crear un nou *IP Core*. Per a seguir aquest model s'hauran de complir una sèrie de requisits hardware i software i s'haurà de fer un anàlisi funcional.
4. Els llenguatges que s'utilitzaran per a la realització d'aquests cores seran HDL (*Hardware Description Language)* que com el seu nom indica, són llenguatges de descripció del Hardware i C++ pel que fa a la lògica i control d'aquest hardware a alt nivell.
5. Seguir aquesta metodologia per a la creació del *driver* PWM (*Pulse Width Modulation*). Aquest *driver* serà creat i provat sobre un element mòbil amb 2 servo motors que giraran en diverses direccions segons els polsos enviats.
6. Seguir la metodologia per a la creació del *driver* I2C (*Inter-Integrated Circuit)*. Aquest *driver* serà creat i provat a través de la instal·lació d'un sensor a la placa, el qual farà canviar el sentit de gir dels servo-motors instal·lats.

7. Proves. A fi d'analitzar el correcte funcionament dels nous *IP Cores.*

2.2 Planificació

Aquest projecte final de carrera ha estat planificat amb l'objectiu que la seva durada correspongui a un semestre. Originalment, estava plantejat per ser realitzat durant el primer semestre corresponent al període setembre – febrer, però per problemes amb altres feines s'ha hagut d'endarrerir fins al període febrer – juny.

S'ha de dir que aquesta planificació s'ha seguit escrupolosament, i pràcticament no s'ha vist afectada durant el transcurs del projecte, encara que hi ha hagut tasques que s'han vist endarrerides, per problemes que es comentaran més endavant.

2.2.1 Identificació dels recursos

L'equip de desenvolupament del projecte estarà format per una sola persona que exercirà les tasques de Cap de Projecte, tècnic i programador. L'enginyer en qüestió, s'encarregarà de mantenir un control tècnic i econòmic a fi de fer-lo rentable. Serà el responsable i encarregat de planificar i distribuir els recursos i de participar en l'elaboració de les especificacions funcionals detallades.

També serà l'encarregat de la creació i especificació de l'arquitectura a utilitzar, així com també, de la programació dels elements necessaris per al funcionament del projecte.

2.2.2 Terminis de lliurament

Els terminis de lliurament seran els fixats per l'Escola Tècnica Superior d'Enginyeria (ETSE). En el cas de la convocatòria del mes de juny del curs 2006 - 2007 serà el dia 1 de juny la data màxima per a decidir si es presenta el projecte. Mentre que el document de memòria es podrà entregar com a molt tard el dia 15 de juny de 2007.

2.2.3 Identificació de les tasques necessàries per a realitzar altres tasques

A continuació es detallen una sèrie de tasques que són vinculants per a la realització d'aquest projecte, i sense les quals resultaria impossible avançar en el desenvolupament del mateix.

Les taques es detallen a continuació:

1. Aproximació a la placa UP3. Entendre i comprendre el funcionament d'aquesta placa, així com la interacció de tots els seus components.
2. Aproximació a llenguatges HDL i C++ en la plataforma d'Altera.
3. Bugs i aspectes tècnics a considerar en l'entorn de treball. Serà necessari que tots els elements compilin i la placa respongui al requeriments fixats.
4. Creació de la metodologia necessària a seguir per a la creació d'un IP Core. S'ha de marcar la pauta de com han de funcionar aquests elements.
5. Realització del *driver* PWM, seguint aquesta metodologia.
6. Realització del *driver* I2C, seguint aquesta metodologia
7. Provar que funcionen aquests dos *drivers* a la placa.

2.2.4 Càlcul del termini necessari per a la realització de cada tasca

El projecte de final de carrera està fixat a una durada de 15 crèdits. Un crèdit ECTS que serà el considerat per a aquest projecte són 25 hores. D'aquesta forma el valor en hores del projecte equival a 375h.

Aquest valor és un temps raonable encara que es comptarà amb un 10% de marge d'error que s'aplicarà abans de començar el projecte, i del qual el Cap de Projecte n'assumeix tota la responsabilitat. S' aplicarà per tenir un percentatge de marge en cas que es produeixi algun imprevist. D'aquesta manera es contemplarà que el projecte tindrà una durada de 412,5 hores.

Conseqüentment, s'establiran aquests valors com a base per a establir un model a seguir per a la realització d'aquest projecte, així com a base per determinar la viabilitat econòmica. Aquestes hores de més estan pressupostades i no afectaran a la viabilitat del projecte.

Tot seguit es detallaran totes i cadascuna de les tasques a realitzar juntament amb la seva durada:

a) Etapa Preliminar (22,5 h)

1. Objectius i abast del projecte: 7,5h
2. Anàlisi funcional i requeriments: 10h
3. Planificació inicial: 5h

b) Desenvolupament del projecte (390 h)

1. Memòria (100 h)

1.1 Especificació de requisits. Anàlisi funcional: 10 h

1.2 Planificació: 5 h

1.3 Estudi de viabilitat: 3 h

1.4 Cronologia: 2 h

1.5 Creació d' IP Cores (20 h)

1.5.1 Disseny Hardware: 10 h

1.5.2 Disseny Software: 5 h

1.5.3 Verificació: 5 h

1.6 Prototipatge sobre UP3 – NIOS (30 h)

1.6.1 La plataforma Altera. Procediment: 10 h

1.6.2 Explicació del disseny dels IPs sobre la plataforma NIOSII (20 h)

1.6.2.1 *Driver* PWM: 10 h

1.6.2.2 *Driver* I2C: 10 h

1.7 Conclusions i millores: 9 h

1.8 Bibliografia: 1h

1.9 Revisió (20 h)

1.9.1 Aplicació dels canvis introduïts: 20 h

2. Desenvolupament (290 h)

2.1 Aproximació als HDL's: 10h

2.2 Aproximació a l'entorn de programació d' Altera: 10 h

2.3 Bugs de l'entorn: 60 h

2.4 Implementació del *driver* PWM (50 h)

2.4.1 Descripció Hardware: 40 h

2.4.2 Descripció Software. 10 h

2.5 Implementació del *driver* I2C (80 h)

2.5.1 Descripció Hardware: 60 h

2.5.2 Descripció Software: 20 h

2.6 Implementació de la lògica de control (30 h)

2.6.1 Estudi de la lògica: 10 h

2.6.1 Implementació de la lògica d'alt nivell: 20 h

2.7 Test i proves: 25 h

2.8 Revisió i modificacions: 20 h

2.9 Possibles millores: 5 h

2.2.5. Diagrama de Gantt

El diagrama de Gantt que es mostra a continuació, (veure **Figura 2.1**) mostra la distribució de tasques en què es va dividir el projecte, i sobre les quals es va realitzar una planificació temporal per dur-les a terme seguint les mostrades en l'apartat anterior.

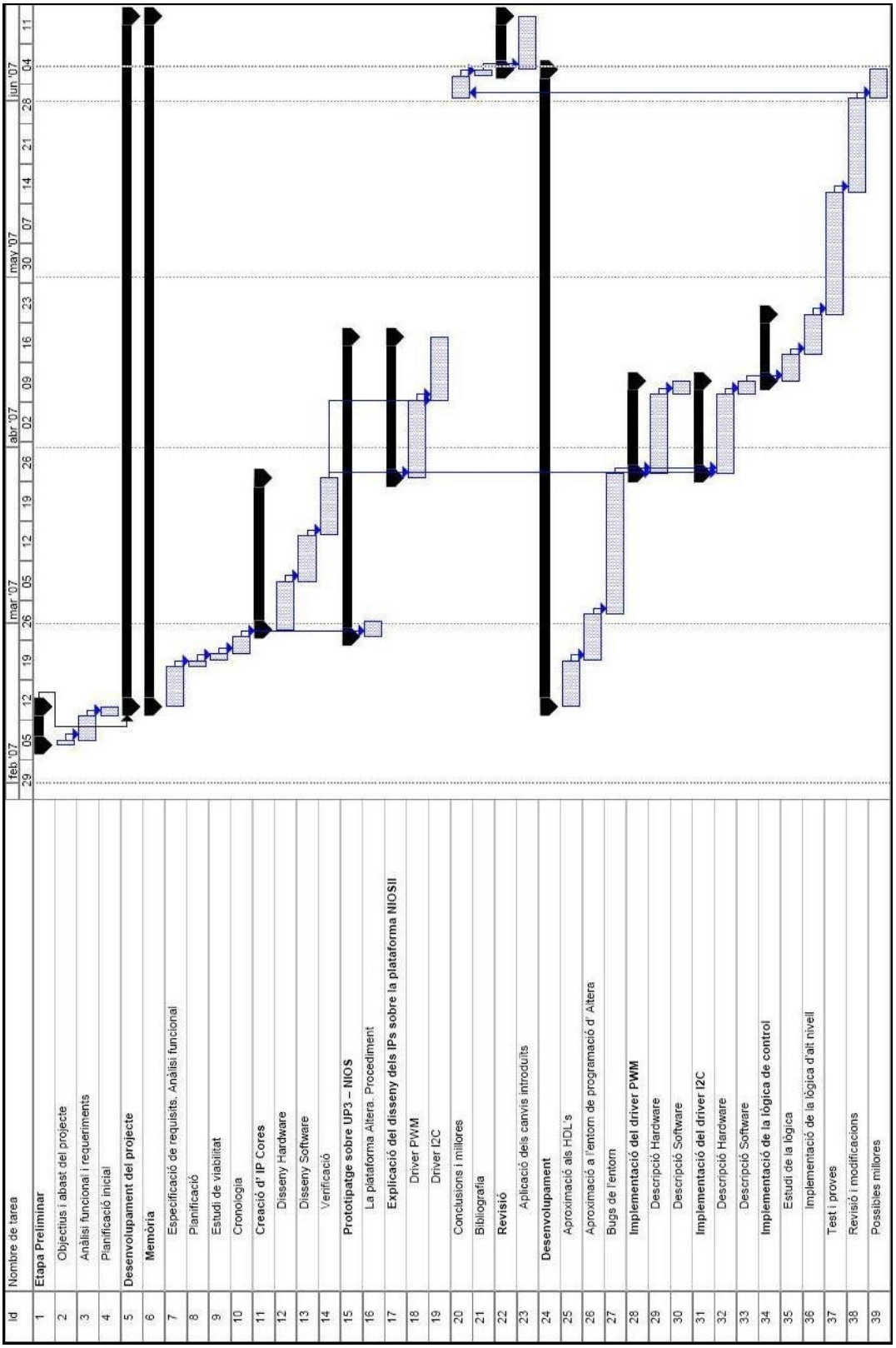

Figura 2.1 Diagrama de Gantt. Planificació

2.3 Viabilitat

La viabilitat d'aquest projecte vindrà determinada per l'estudi de viabilitat que es realitzarà a continuació, juntament amb els riscs que la realització d'aquest projecte comporta.

2.3.1 Estudi de viabilitat

Tal i com s'ha exposat anteriorment, el projecte serà desenvolupat per a un únic recurs, el qual estarà dedicat a temps total en el projecte. El fet que tan sols hi hagi un recurs disponible, no suposarà un problema perquè el projecte pugui estar completat en les dates assenyalades en els apartats anteriors.

Els temps assignats a cada tasca són temps raonables i cadascun ha estat pensat perquè pugui ser realitzat en aquest espai de temps. A part d'això, ja es contempla el percentatge d'error al qual pot estar sotmesa cada tasca, donant més força a una planificació estudiada.

Quant als elements necessaris per al desenvolupament del projecte serà necessari disposar de l'aplicació QuartusII v.5.1, el SOPC Builder i el NIOS IDE que es poden descarregar des del web http://www.altera.com i obtenir-ne una llicència gratuïta.

D'altra banda també serà necessari disposar d'una placa UP3 que serà subministrada per l' Escola Tècnica Superior d' Enginyeria en forma de préstec, juntament amb elements com sensors i servo motors.

Així doncs, tenint en compte aquestes consideracions i veient que la inversió econòmica és nul·la, tan sols es comptabilitzarà la inversió en hores del projecte per determinar-ne la seva viabilitat que en un principi, després de l'anàlisi efectuat prèviament, sembla factible.

Pel que fa a la legalitat del producte, i tal i com ja s'ha avançat en apartats anteriors, tan sols s'utilitzaran aplicacions amb llicències subministrades per Altera i que per tant, no suposen cap problema legal.

2.3.2 Riscs

Un projecte que tracta amb hardware se l'ha de tractar amb molta cura, ja que per efectes d'aquest hardware sobre el software es poden observar comportaments imprevistos, que poden afectar directament a la planificació dissenyada. Paral·lelament, cal tenir en compte que les planificacions sempre parteixen de la base ideal més un percentatge de marge que acostuma a ser insuficient.

Tot seguit doncs, s'exposaran els possibles riscos als quals pot estar sotmès el projecte, ja que poden provocar endarreriments en la realització de diverses tasques. Aquestes demores poden provocar un encariment en hores pel que fa la planificació del projecte, i temporal que pot fer que no s'ajusti a les dates estimades.

Així doncs, a continuació es detallen els possibles riscs que pot comportar el projecte:

1. Que el recurs disponible, en aquest cas únic per a la realització de les tasques es posi malalt, o no pugui estar disponible per motius aliens al projecte.

2. L'entorn de programació i aplicació no sigui el suficientment potent i tingui problemes.
3. La placa UP3 tingui elements deteriorats que distorsionin els resultats esperats o no deixin executar cap aplicació.
4. Possibles sorolls produïts pels elements afegits a la placa com servo motors i sensors alterin el comportament definit a través del software.
5. No es disposi del material necessari en els períodes indicats.
6. Canvi d'objectius del projecte mentre es duu a terme aquest.
7. La no utilització d'una metodologia adequada quant a la creació dels IP Cores.

2.4 Cronologia

En aquest apartat s'explica el què ha passat durant el desenvolupament del projecte.

2.4.1 Estudi cronològic

Durant el desenvolupament d'aquest projecte han sorgit diversos problemes, molts dels quals estaven contemplats en l'apartat de riscs, i que han provocat l'endarreriment del projecte. Tasques bloquejants com la creació de petits sistemes, per a iniciar la comprensió de la placa UP3 han fet que s'endarrerís més del compte l' inici de les altres tasques, cosa que ha provocat un increment d'hores de dedicació.

Els *bugs* de l'entorn d'Altera, així com aquests primers sistemes creats, han estat principal problema que s'ha trobat durant el desenvolupament del projecte. En

l'entorn SOPC quan algun element no funciona, provoca un endarreriment en cascada que afecta a la resta de tasques. Aquest endarreriment, a més a més acostuma a ser gran ja que la localització d'errors és fa feixuga, i en ocasions, com és el cas d'aquest projecte, els problemes no provenien d'un sistema mal construït, sinó d'un software corrupte que afectava a tot sistema que s'intentava provar.

Com és fàcil d'imaginar, problemes d'aquest estil tenen un cost molt elevat, ja que es perd molt de temps per intentar trobar un problema que en realitat no existeix.

Per altra banda, tasques inicialment pressupostades en excés, com la realització del *IP Core* PWM, han compensat mínimament l'excés de temps dedicat en l' anterior. Tot i això, s'ha modificat la planificació en alguns punts més, per tal d'adaptar-la a les noves situacions que han anat sorgint.

El *IP Core* de I2C, s'ha tingut molt poc temps per a provar-lo ja que es va rebre tard, prop de l'entrega d'aquest projecte.

També es té en compte que el projecte va iniciar-se abans del què estava planificat, ja que durant el mes de setembre i d'octubre del 2006 ja s'havia començat a parlar del mateix.

No obstant, un altre motiu dels endarreriments ha estat el fet de compaginar activitats alienes al projecte com són les obligacions laborals que han interferit directament amb la planificació prevista. Aquest fet que podria servir per dir que el projecte sí que és viable en termes relatius, tot i la realització d'hores extra.

3. CREACIÓ D' IP CORES

En aquest apartat del projecte es pretén descriure el seguit d'accions que s'han de seguir per a la creació d'un *IP Core*. La definició pretén ser tan **genèrica** com sigui possible, a fi de poder tenir-se en compte com si d'un estàndard es tractés.

La creació d'aquest document és possible gràcies a l'estudi de l'entorn utilitzat en Altera per a la creació de components (*SOPC Builder*), i a un treball de camp sobre les necessitats de la plataforma NIOS.

Tot i l' intent de ser el màxim genèrica possible, la definició tan sols estarà provada a l'entorn esmentat anteriorment, i la majoria de consideracions es faran partint de la base plantejada per Altera. Cal tenir en compte aquest factor, ja que de forma constant s'hi faran referències en motiu de la seva àmplia utilització per al desenvolupament de components. Així doncs, tot dissenyador de components que llegeixi aquest manual, entendrà que és genèric per a una plataforma NIOS, en l'entorn de programació proporcionat per Altera.

D'aquesta manera, els manuals d' Altera proposen una metodologia SOC (*System On Chip*) de disseny que cal ser completada amb treball propi en el moment en que es dissenya una aplicació. És molt recomanable llegir els manuals, sobre l'entorn d' Altera, que es destaquen a la bibliografia d'aquest document, ja que són la base per entendre què s'exposa a continuació.

3.1 Definició metodològica per a la creació d'un *IP Core*

En moltes ocasions, tal i com passa en aquest projecte, un es troba davant la necessitat de fer funcionar un cert component en un entorn que no està preparat per

acceptar-lo. Així doncs, la qüestió resideix en integrar aquest component a l'entorn indicat.

Hi ha ocasions, com la de l'entorn utilitzat en aquest projecte, que permet buscar i utilitzar *drivers* ja fets per altres dissenyadors de components, i tenir la possibilitat de crear-ne de nous. La gràcia de compartir els dissenys per a la gestió d'un component és alleugerir el treball de cada dissenyador, ja que no cal reinventar la roda cada vegada. El problema és si el component en qüestió no està inventat, i s'ha de crear de zero.

És en aquestes situacions que és necessari tenir una base a seguir per tal d'avançar de forma ordenada i clara cap a l'objectiu final, o sigui tenir un *driver* que gestioni el nou component hardware afegit.

3.1.1 Anàlisi de Requeriments

Abans de començar, s'han de tenir en compte els següents requeriments software i hardware, a fi d'obtenir una correcta creació del component.

Tal i com ja s'ha indicat anteriorment, serà necessari disposar de l'entorn de programació d' Altera (SOPC Builder, QuartusII i NIOSII IDE) i conèixer-lo. Una placa de desenvolupament NIOS. També serà important tenir coneixements bàsics sobre la interfície AVALON, que és el bus on anirà connectat el nou component perquè la CPU el gestioni.

Evidentment, si no es disposa de placa es pot desenvolupar un component, el problema resideix en què l'usuari serà incapaç de provar-ho d'una forma pràctica.

Partint d'aquests requisits, ara ja només falta saber que s'ha d'obtenir de l'entorn de desenvolupament. Així doncs, un component es pot desglossar en les següents parts o fitxers:

- Arxius hardware. Mòduls en llenguatge HDL que descriuen el hardware del component.
- Arxius software. Capçaleres en llenguatge C, on es mapegen els registres del component, i un *driver* software que permet al programa controlar el component.
- La descripció del component. Aquest fitxer defineix l' estructura del component i subministra la informació necessària per a la integració d'aquest al sistema (class.ptf).

3.1.2 Passos per al disseny d'un component

Aquesta secció es centra en els passos que s'ha de seguir per a la creació d'un component. La seqüència de passos està pensada per a un component que actuarà com a element esclau, encara que tal i com es pot veure són passos fàcilment extrapolables cap a components *master*. Els passos, no requereixen ser seguits en un escrupolós ordre, encara que sí exigeixen un cert grau de rigor. En la primera part d'aquest punt, es veuran els passos genèrics a realitzar mentre que tot seguit, es podran veure els passos complerts per al disseny d'un component, incorporant *IP Cores* propis.

Passos Genèrics:

1. Especificació de les necessitats hardware.
2. Si el microprocessador s'utilitzarà per a controlar el component, s'haurà d'especificar a l' interfície de programació de l'aplicació (API) que accedeixi i controli el hardware.
3. Basat en els requeriments hardware i software, definir una interfície pel bus AVALON que disposi de mecanismes de control adequats, així com un bon funcionament.
4. Escriure el HDL que descrigui el hardware en Verilog o VHDL.
5. Testejar els components hardware sols per a verificar-ne un correcte funcionament.
6. Escriure les capçaleres en C que defineixin el mapa de registres del hardware pel software.
7. Utilitzar l' editor de components per ajuntar els arxius hardware i software per a la creació d'un component.
8. Instanciar el component en mòdul del SOPC Builder.
9. Testejar els accessos als registres utilitzant un microprocessador com el processador NIOSII. La verificació pot ser feta al hardware o bé en un simulador com el ModelSim.
10. Si un microprocessador serà utilitzat per a controlar el component, s'ha d'escriure un *driver* en software.
11. De forma reiterada, realitzar les següents accions:
    - Fer els ajustaments hardware necessaris.
    - Fer els ajustament software necessaris.
    - Incorporar els canvis software i hardware al component utilitzant l'editor de components.
12. Construir un sistema complet a través del SOPC Builder incorporant una o més instancies del component creat.

13. Fer les verificacions necessàries per tal que els components s'ajustin als requeriments demandats.
14. Finalitzar el component i distribuir-lo perquè pugui ser reutilitzat.

El disseny per a un component que hagi d'actuar com a *master*, serà similar exceptuant la part de desenvolupament software.

Passos Complerts:

Pel que fa aquesta part, cal dir que es centra en seguir els passos explicats anteriorment, ampliant el procés de creació de nous *IP Cores*, que és el tema sobre el qual gira aquest projecte. Paral·lelament, abans d'enumerar la sèrie de passos a seguir cal tenir clars una sèrie de conceptes que s'exposaran a continuació, i en l' **Apartat 4** d'aquest projecte s'ampliaran, ja que és en aquell moment quan es parla d'arquitectura.

La complexitat o diferència de la creació d'un component en aquest entorn, vers els entorns tradicionals, recau sobre les particularitat del bus Avalon. Aquest bus està dissenyat per a connectar processadors *On-Chip* i perifèrics en un sistema SOPC. La interfície del bus especifica les connexions entre els components *master* i *slave*. Aquest mòdul es genera automàticament, així que el dissenyador de components tan sols s'ha d'ocupar d'unir els perifèrics al bus.

El problema és que el bus no pot comunicar-se directament amb els perifèrics, i necessita d'un traductor que transmeti la informació entre ell i el dispositiu. Per aquest motiu, el component creat ha d'anar recobert amb un *wrapper*, que tradueixi els senyals del bus amb les del perifèric i a l' inrevés. Per això, es requereix de l' interfície del bus, com a part imprescindible en la creació d'un nou *IP Core.* A través d'aquesta interfície, el bus podrà llegir o escriure en els registres del component, i actuar en conseqüència, executant la seva lògica a través de les ordres que li dóna el

dispositiu *master*. Altrament, també es pot enviar la informació pertinent als registres per tal que el processador actuï d'una forma determinada.

Així doncs, els passos complerts per a la creació d'un *IP Core* són:

1. Definir l'especificació funcional del nou component. La definició haurà de concretar el tipus de processador que s'utilitzarà per a controlar el component, el tipus de rellotge, tipus de component (*master* o *slave*), etc.
2. Un cop especificada la funcionalitat del component, s'ha de definir l'estructura lògica d'aquest. Cal crear un esquema amb tots els mòduls o eines que es necessitaran, utilitzant les funcionalitats definides, per a la definició de l'estructura o tasca lògica del component.
3. Establir els senyals d'entrada i sortida que tindrà l'estructura lògica. Els senyals d'entrada o sortida podran ser, per exemple, senyals d' *enable*, rellotge o *reset*, entre d'altres.
4. Programar aquesta funcionalitat en algun llenguatge de descripció de hardware (HDL), com ara Verilog o VHDL. D'aquesta manera, obtindrem un fitxer en aquest llenguatge que s'encarregarà de gestionar les tasques lògiques o hardware del component.
5. Un cop establerta la lògica del component, s'ha d'habilitar l'accés als registres per tal que el dispositiu pugui llegir i escriure. S'ha de crear un fitxer de registres on es mapejaran totes les entrades i sortides de la lògica de control al component cap al registre corresponent. Aquest fitxer és molt important, ja que cada registre anirà redirigit cap a l'espai d'adreces del bus Avalon, perquè el component es pugui comunicar amb el sistema.
6. Cada registre haurà de ser de lectura i/o escriptura depenent de les necessitats funcionals que requereixi el component, el què significa que el software que més endavant es crearà podrà llegir i establir valors.
7. En aquest fitxer, establir tots els senyals d'entrada i sortida del bus Avalon, així com els senyals cap al mòdul creat, tant d'entrada com de sortida.

8. Finalment, a fi de tenir el component instanciat, es necessària la inclusió d'un fitxer que gestioni les comunicacions entre el component i l'exterior. És aquí on s'establiran els senyals del bus Avalon que es mapejaran sobre els senyals del component creat. Aquesta assignació és imprescindible per a una correcta comunicació i entesa entre component i sistema.
9. Revisar cada un dels fitxers creats, verificant el seu correcte funcionament a través d'eines com el ModelSim, que permeten simular l'excussió del component.
10. Arribat aquest punt, ja es té a la disposició de l'usuari el component hardware, seguint l'estructura que demana l'arquitectura del bus Avalon, tal i com s'explica a l' inici d'aquest apartat, i s'amplia en el següent.
11. Si el component creat és de tipus *slave* s'haurà d'especificar a l' interfície de programació de l'aplicació (API) que accedeixi i controli el hardware.
12. Escriure les capçaleres en llenguatge C que defineixin el mapa de registres, prèviament definit al fitxer de registres hardware, perquè el software els pugui gestionar. També defineix el software que necessitarà el processador NIOS per a gestionar el *driver*.
13. Obrir el QuartusII i crear un nou projecte.
14. A través d'aquest obrir el SOPC Builder i executar l'editor de components.
15. Utilitzar l'editor de components del SOPC Builder per a adjuntar els fitxers hardware i software per a la creació d'un component. Mitjançant aquesta eina, es compactaran els diferents mòduls que s'han creat per obtenir el component.
16. Triar l'opció de menú del SOPC Builder que permet afegir un nou component i afegir els fitxers HDL que s'han creat, en la pestanya indicada. L'editor de components analitzarà aquests fitxers i els validarà. L'ordre dels fitxers afegits és important, ja que s'han d'ordenar des del nivell més extern com és l' interfície Avalon fins a la tasca lògica.
17. Un cop afegits els fitxers, cada senyal que hi ha en els fitxer HDL de més alt nivell es mostrarà a la pestanya dels senyals. Per defecte, l'editor assigna els senyals que troba en aquest fitxer amb un tipus, que en la majoria d'ocasions

és vàlid. Si un senyal no sap de quin tipus és, la qualifica com a senyal *export*. Conseqüentment, és important modificar el tipus de cada senyal segons les necessitats del sistema. Si, tal i com s'ha comentat en apartats anteriors, es té fet un mapeig de senyals, determinar el tipus de cada un serà una feina trivial.

18. Un cop assignats els senyals és passa a la pestanya d' interfície. En aquesta pestanya es permet configurar les propietats d' interfície del bus Avalon del component. És aquí on s'ha d'indicar el tipus de component, se li assigna un nom. No obstant, la clau d'aquest punt és que s'indica que l'adreçament es farà a través dels registres.
19. La següent pestanya fa referència als fitxers software creats. Aquesta deixa associar fitxers software al component i especificar-ne quin tipus d'ús se'n farà. El fitxer software creat per a treballar amb els registres s'afegirà com a *include* dels registres. D'altra banda, si s'han creat rutines d'alt nivell en C per a gestionar el *driver*, també s'afegiran, però en aquest cas, optant per l'opció *HAL*/*inc* o *HAL/src* segons si els fitxers que s'afegeixen són capçaleres o les funcions creades.
20. Arribats a aquest punt, es pot salvar el nou component creat.
21. Un cop creat, es trobarà el nou component en el grup que li hagi definit i es podra afegir al sistema.
22. A l' afegir el nou component al sistema, sortirà un assistent que permetrà o no, modificar alguns paràmetres de configuració d'aquest. Un cop decidides les opcions que es prefereixen, s'inclourà el component al polsar en finalitzar.
23. Comprovar que les adreces d'excepció i *reset* del processador del sistema apunten a un mòdul de memòria per evitar que el sistema falla.
24. Un cop afegit i configurat es pot generar el sistema, polsant sobre l'opció *Generate* del SOPC Builder. A continuació, es tindrà el sistema correctament creat. Ara cal dirigir-se a l'aplicació QuartusII.
25. Modificar l'esquema del Quartus del sistema perquè les entrades i sortides al sistema quedin cobertes amb els elements que necessiti. Tot seguit, unir els components del sistema que requereixin estar connectats.

26. Assignar al sistema la placa que s'utilitzi, tenint en compte totes les particularitats d'aquesta, així com les indicacions del fabricant. Aquesta opció es troba en el menú *Assignments, devices*. Després d'assignar-la, s'informarà en aquest mateix menú com la placa ha d'interpretar els pins no utilitzats. Per defecte, la placa els haurà d'interpretar com a entrades, encara que depenent del sistema les podrà entendre com a sortides.
27. Realitzar l'assignació correcta de pins, a través del *Pin Planer*.
28. Compilar tot el sistema.
29. Un cop compilat, s'ha de descarregar el sistema a la placa. Per això, s'obrirà el *Programmer*. Aquesta opció permetrà que s'estableixi una comunicació entre la placa, que s'haurà connectat al port corresponent de l'ordinador, i l'aplicació.
30. Descarregada la informació, s'ha d'executar el software del processador NIOS per utilitzar el component creat. Tant és així, que requereix que obrim la tercera aplicació, que s'obre des del QuartusII. Aquesta aplicació és el NIOSII IDE.
31. A través d'aquesta aplicació, es crearà un projecte software que estarà basat en el sistema creat. Afegir les capçaleres i fitxers en C necessaris per a provar el component, i generar un projecte que sigui capaç de gestionar el component.
32. Compilar i descarregar aquesta lògica cap a la placa, tot observar-ne el comportament per tal de validar que la lògica programada és correcta.
33. Validada la lògica i el component, aquest ja està llest per a ser compartit amb la resta de la comunitat, perquè pugui ser reutilitzat.

Per a més informació sobre els elements o paràmetres de configuració modificables en les aplicacions, és imprescindible llegir els manuals que es poden trobar en la bibliografia d'aquest document, ja que s'ha donat més importància als passos a seguir, i no entra en punts de configuració que, tanmateix, es poden trobar en la documentació d' Altera.

3.2 Disseny Hardware

El disseny del hardware és com tot disseny lògic, el procés que ve després de la fase d'especificació de requisits. Un cop es tenen clars els objectius dels component i la seva funcionalitat, codificar el codi en HDL que el farà funcionar pot tractar-se d'una qüestió pràcticament immediata.

L'arquitectura d'un component típic habitualment segueix els següents blocs representats en la figura següent:

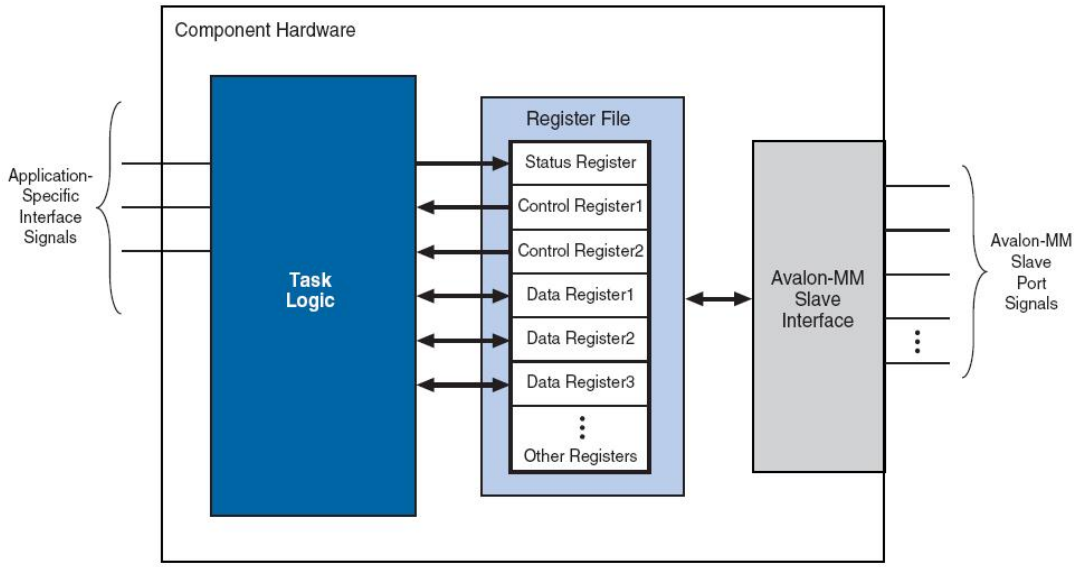

Figura 3.1 Arquitectura d'un component

Aquesta figura mostra el diagrama típic de blocs del component amb un port Avalon esclau. Les parts que el composen es comenten a continuació:

- *Task logic*: La tasca lògica implementa la funcionalitat principal del component. Aquesta tasca és depenent del disseny.

- *Register file*: Els registres proporcionen el camí de comunicació per als senyals des de la lògica de control fins al món exterior, i al inrevés. La interfície Avalon pot llegir i escriure en els registres, gràcies als nodes interns que estan mapejats en adreces a les quals pot accedir.
- *Avalon interface*: La interfície Avalon conté un registre de tipus *front-end*. Aquesta utilitza qualsevol senyal Avalon, per a accedir al registre i realitzar la transferència requerida a través de la tasca lògica. Hi ha factors com; l'amplada de dades que s'han de transmetre, els requeriments de transferència, el tipus de transferència i la rapidesa entre els dispositius hardware que interactuaran, que afecten directament a aquesta interfície.

3.3 Disseny Software

El disseny del software és el següent pas i serà l'encarregat de controlar i gestionar el dispositiu creat. Si es pretén tenir un control sobre un component, s'han de subministrar uns arxius que defineixin el comportament o la visió del component vers el hardware que es té.

Així doncs, com a mínim serà necessari definir un mapa de registres per a cada un dels ports esclaus que siguin accessibles per al processador. L'editor de components dóna un paquet de capçaleres en C per definir la visió software vers el hardware.

Habitualment, un fitxer capçalera s'utilitza per a la declaració de macros per a poder escriure i llegir cada registre en un component, on cada component té assignades unes adreces.

Els *drivers* escrits en llenguatges d'alt nivell, abstreuen els detalls d'un component hardware. L'abstracció, permet accedir al component des del mencionat alt nivell, cosa que facilita la vida a l'hora de programar. Els requeriments software varien

segons les necessitats del component. Les rutines més habituals utilitzades són les relacionades amb la inicialització de hardware, llegir dades i escriure-les.

El software depèn també del tipus de processador amb el que es tracta, i del nivell d'especificació que se li dóna al *driver*. L'editor de components permet empaquetar fàcilment els *drivers* en una capa d'abstracció HAL (*Hardware Abstraction Layer*) que NIOSII utilitza per al desenvolupament dels components. Aquesta capa és un subdirectori, del qual en pengen dos més; el primer és anomenat *inc* i és on van les capçaleres en C, mentre que l'altre és *src* i és on hi ha el cos de les funcions. D'altra banda, els fitxers que especifiquen el hardware estaran situats en el subdirectori *hdl*, mentre que les capçaleres accessibles des del software programat, per a controlar el component, estaran en el subdirectori *inc*.

Així doncs, per tal de facilitar els *drivers* per a altres processadors, s'han de modificar de tal manera perquè s'adaptin a les necessitats de desenvolupament d'aquests.

3.4 Verificació del component

El component dissenyat es pot verificar a través d'estats incrementals a mesura que es va avançant en el disseny del component. Habitualment, primer s'acostuma a verificar la lògica del hardware com a una unitat, i més endavant, ja es verifica el component com a part del sistema.

Per a provar la unitat lògica hardware, s'han d'utilitzar els mètodes que a l'usuari més li vingui de gust o li sigui més còmode. Eines que permetin la simulació del hardware com el ModelSim poden ser de gran ajuda per a comprendre la funcionalitat programada i si hi ha algun tipus de problema poder-lo corregir. De la mateixa manera, també es pot verificar tot el conjunt del component incloent el fitxer de

registres de la interfície Avalon i realitzant les proves que es desitgi un cop ja inclòs el component en el projecte.

Un cop empaquetats els fitxers en HDL en un component, utilitzant l'editor de components, el *kit* de desenvolupament del NIOSII ofereix un mètode senzill per a simular transaccions de lectura i escriptura en un component. L'usuari pot escriure un codi en C perquè el processador iniciï transferències de lectura i escriptura cap al component. Els resultats poden ser verificats amb un simulador com el ModelSim o sobre el hardware, o sigui sobre la placa UP3, ja siguin escriptures a memòria, com la lectura de dades en la pantalla LCD.

D'altra banda, un cop es té el component, es pot instanciar fàcilment el component en el sistema a través de l'editor de components, i d'aquesta forma, verificar la funcionalitat de tot el mòdul del sistema. El SOPC Builder pot produir tests de prova que poden verificar cada nivell del sistema. La capacitat de simulació dependrà de l'entorn i dels components utilitzats en el sistema.

Durant l' etapa de verificació, incloure un processador NIOS al sistema pot ser de gran ajuda, ja que un es pot beneficiar de l'entorn de simulació que té. Si el component creat no té cap mena de relació amb aquest processador, la simulació autogenerada del ModelSim serà una bona base per tal que es pugui construir una lògica de control per al sistema en qüestió.

4. PROTOTIPATGE SOBRE UP3 - NIOS

Aquest apartat conté el procediment que s'ha anat seguint a partir de l'apartat anterior, personalitzat segons els avantatges i desavantatges que proporciona la plataforma Altera sobre els processadors NIOS. Aquests problemes seran comentats al final d'aquest apartat.

4.1 La plataforma Altera. Procediment

La plataforma i l'entorn que proporciona Altera està composat per 3 aplicacions: SOPC Builder, Quartus II i NIOS II IDE.

El Quartus II, és el programa principal dels tres. És el que permetrà realitzar el disseny de l'arquitectura del sistema, així com els esquemàtics, l'assignació dels components a les sortides dels pins de la FPGA, i tot a un cost molt petit. A més a més, habilitarà la connexió a la placa per tal d'enviar-hi tot aquest disseny.

El SOPC Builder és una potent eina per al desenvolupament i creació de sistemes basats en processadors, perifèrics i memòries. És una aplicació capaç de definir i generar un complet *System On a Programmable Chip* (SOPC) en molt menys temps que utilitzant mètodes manuals més tradicionals. És una eina de propòsit general per a la creació de varis dissenys que poden o no, contenir processador i per descomptat, no són d' obligat ús d'un processador NIOSII. Així doncs, serà la peça encarregada d'integrar els components hardware en un sistema.

El NIOS IDE és la plataforma de desenvolupament del software. El llenguatge de programació que s'utilitzarà en aquest entorn és C++. Aquest entorn permet compilar, editar i depurar els programes escrits per a controlar els components afegits a la placa.

Així doncs, un cop conegut l'entorn en que es treballarà s'ha de veure el procediment a seguir.

4.1.1 Arquitectura del sistema

Aquesta secció parlarà de l'arquitectura fonamental d'un sistema SOPC Builder.

Un component del SOPC Builder és un mòdul que el programa pot reconèixer i automàticament integrar al sistema. L'aplicació connecta múltiples components junts per a crear un sistema que gestioni tota la connectivitat que hi ha entre ells. La **Figura 4.1** mostra un exemple d'un sistema *multi-master,* i components *slave*.

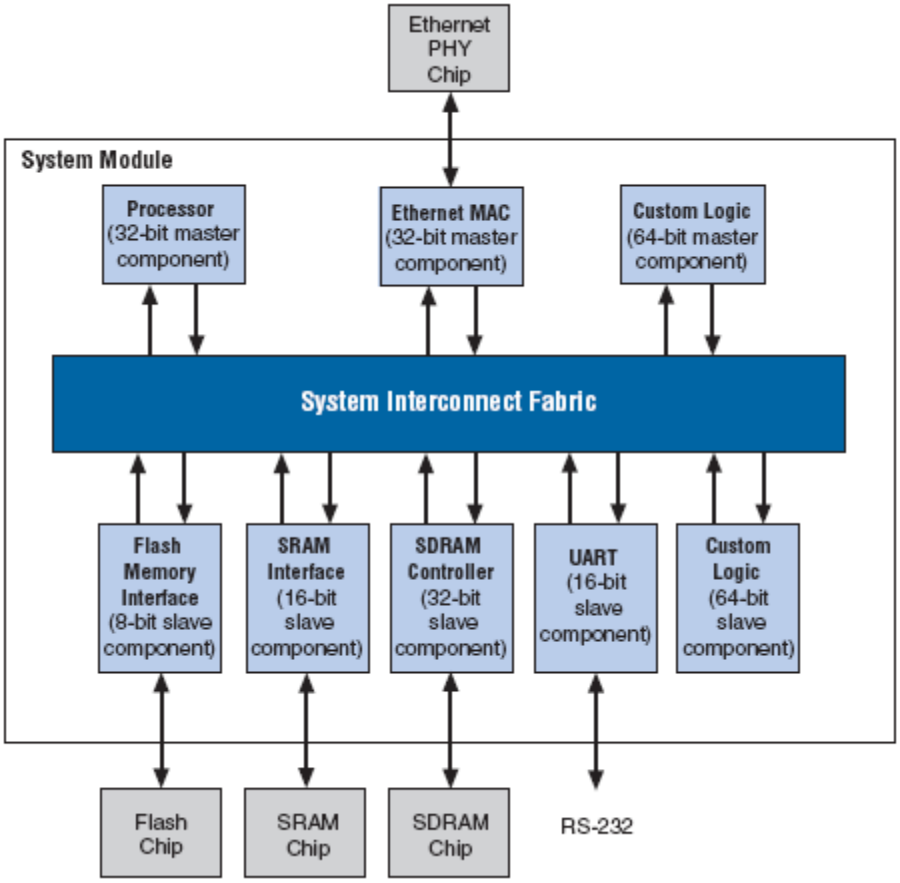

Figura 4.1 Exemple de mòdul del sistema generat amb el SOPC Builder

Tal i com es pot observar, l'arquitectura pot estar composada per una sèrie de components que cada usuari podrà personalitzar, però en tots els seus casos s'haurà de tenir en compte l' especial gestió que es fa de la memòria en aquest tipus de sistemes, així com les particularitats del bus Avalon i de la connectivitat d'aquest vers el processador. El bus Avalon és la particularitat més destacable d'aquest sistema.

Segurament l'element més destacable d'aquest sistema és el bus Avalon. El bus Avalon és un element d'arquitectura simple dissenyat per a connectar processadors *On-Chip* i perifèrics junts en un sistema SOPC. La interfície Avalon especifica les

connexions entre els components *master* i *esclaus*, així com també, el temps amb el que aquests components es comuniquen. Els avantatges que ofereix aquest bus en la present arquitectura són entre d'altres; la simplicitat d'entendre un protocol amb poc temps d'aprenentatge, l'optimització de recursos d'utilització d'un bus lògic i el sincronisme de les operacions, ja que integra la lògica de tots els elements que coexisteixen en el mateix PLD (*Programmable Logic Device*), mentre evita l'anàlisi dels complexos *timings* que existeien entre els components. La **Figura 4.2** mostra l'arquitectura del bus Avalon en forma de diagrama de blocs.

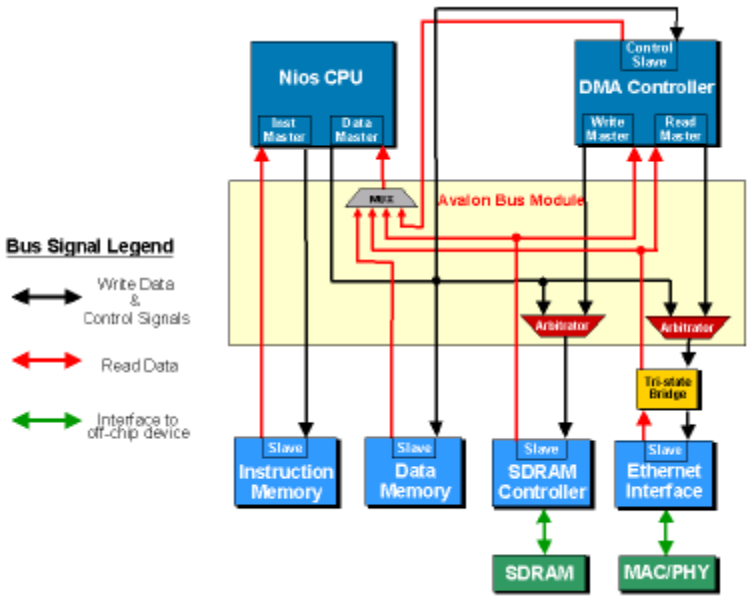

Figura 4.2 Diagrama de blocs del bus Avalon

Aquest mòdul del bus es genera automàticament a través del SOPC Builder, així que el dissenyador tan sols s'ha d'ocupar d'unir els perifèrics al bus. Aquest bus sempre serà utilitzat de forma automàtica pel processador i s'integrarà en el mòdul del sistema, tal i com altres perifèrics. D'aquesta manera com a mòdul és el conducte principal de comunicacions entre els components. Així doncs, es pot dir que aquest

bus és la suma de tot el control, de l'arbitratge de les dades i dels senyals d'adreça que es connecta junt amb els components creant un sistema.

L'especificació del bus defineix senyals i temps per a l'enviament de dades entre un component *master* i un component *slave*. Els senyals que són diferents depenent del tipus de transferència entre el perifèric i el mòdul del bus, així doncs, per a transmissions de tipus *slave*, cal saber que l' activitat dels senyals començarà sempre en la part *master* que serà l'encarregada d'inicialitzar la transferència., però el port *slave* no rebrà els senyals d'entrada directament del port *master*. Per aquesta raó, les transferències es diferencien entre transferències de tipus *master* i de tipus *slave*.

La interfície de sincronització tal i com s'ha esmentat anteriorment és síncrona i està controlada per a un rellotge de mateix bus Avalon que actua com a *master*. Totes les transferències del bus s'inicialitzen quan el rellotge està en fase ascendent i acaba en el moment en què es capturen les últimes dades vàlides, o bé quan el rellotge baixa.

La configuració del bus Avalon i dels seus perifèrics pot ser especificada utilitzant la interfície gràfica que ofereix el SOPC Builder. A través d'aquesta interfície, es poden especificar diversos paràmetres i elements que generaran un fitxer de sistema en format PTF. Aquest fitxer és de text, el qual conté els paràmetres que contenen la funcionalitat del mòdul, els paràmetres de cada perifèric amb la seva funcionalitat, els rols de *master/slave* de cada perifèric, els ports de lectura i escriptura, i el mecanisme d'arbitratge per a cada port esclau, així com el seu accés. Aquest fitxer és passat al generador de HDL el qual crearà un RTL (*Register Transfer Level*), que contindrà la descripció del mòdul del sistema.

Del bus Avalon també es pot comentar que té una interfície de mapeig en memòria. La relació entre aquests dos components va molt lligada ja que en moltes ocasions, es vol mantenir una comunicació entre els components i el processador, i aquesta no pot ser directa. El sistema proporciona una descripció d'adreces on es pot llegir i escriure,

i que tan el component *master* com el *slave* coneixen. D'aquesta manera l'esclau es comunica amb el *master* a través d'aquestes direccions de memòria.

Així doncs, les diferències entre l'arquitectura d'un bus Avalon són ben diferents respecte a les arquitectures tradicionals, en les quals un o més busos *master* o *slave*, es connecten a un bus compartit. Un sol àrbitre té el control sobre el ús, per evitar que es produeixin múltiples accessos sobre ell, ja que en els busos de tipus *master* és l'àrbitre qui s'encarrega de garantir un sol accés simultani. Una vegada aquest té el control, envia la transferència al component esclau. Si hi ha múltiples peticions d'accés al bus, l'àrbitre s'encarrega de fer-les esperar.

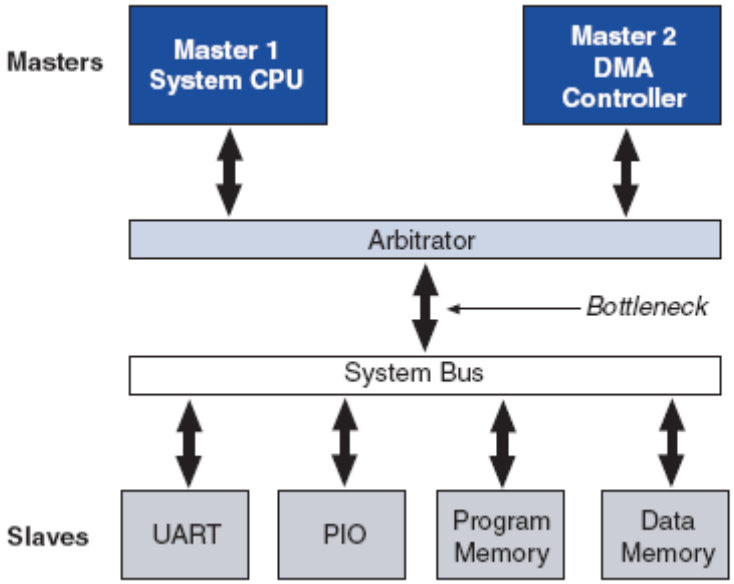

Figura 4.3 Diagrama de l'arquitectura d'un bus tradicional

Altres elements com els diversos tipus de memòries que es poden afegir al sistema, tenen molt a veure amb el bus Avalon ja que la interacció d'aquesta amb el bus, és un altre tret destacable d'aquest tipus d'arquitectura. La **Figura** que es mostra a continuació, és un exemple de com han d'estar afegits elements en aquesta arquitectura.

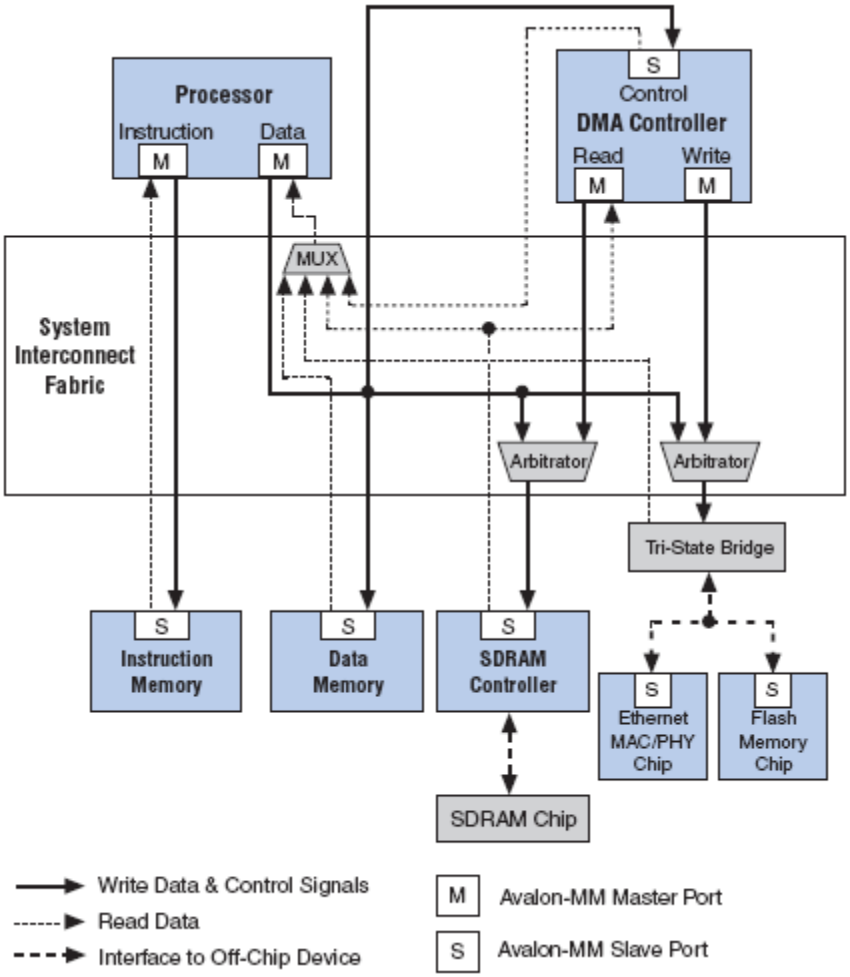

Figura 4.4 Diagrama d'interconnexió entre els blocs d'un sistema

Tal i com es pot observar els elements de tipus *master* tenen el bus de dades multiplexades per a cada dispositiu connectat a ell, així doncs la comunicació serà un a un, i cada una tindrà el seu bus mitjançant transferir la informació Altres elements com memòries de tipus FLASH, o bé components com pantalles LCD hauran d'anar connectades a l'àrbitre del bus Avalon mitjançant un Tri-State que bypassarà la memòria comentada o bé el component que ho requereixi.

Els sistemes SOPC poden accedir directament a molts tipus de memòries RAM i ROM, sense la necessitat de d'utilització de controladores de memòries *off-chip*. Les memòries *off-chip* tenen una velocitat d'accés més lenta al dispositiu de memòria, així com més barata i limitada a 32-bit d'espai d'adreça del bus Avalon. Aquests components faciliten la creació de sistemes de memòria per al desenvolupament en plaques d'Altera. El problema o la qüestió, és que en tots ells és necessari instanciar un component *TriState Bridge*.

El component *TriState Bridge*, és l'encarregat de connectar dispositius al sistema d'interconnexió *on-chip*. Crea una sèrie de senyals d'entrada/sortida al mòdul del SOPC Builder on s'hauran de connectar els pins de la FPGA a través del QuartusII. Aquests pins representen el sistema d'interconnexió cap als dispositius *off-chip*. A més a més crea unes adreces i pins de dades els quals estan compartits per a múltiples dispositius *off-chip.* Així doncs, pel què fa a l'accés s'assegura l'exclusivitat.

4.1.2 Procediment

El procediment que es seguirà per al treball sobre aquesta plataforma és el que es detalla a continuació mitjançant un diagrama:

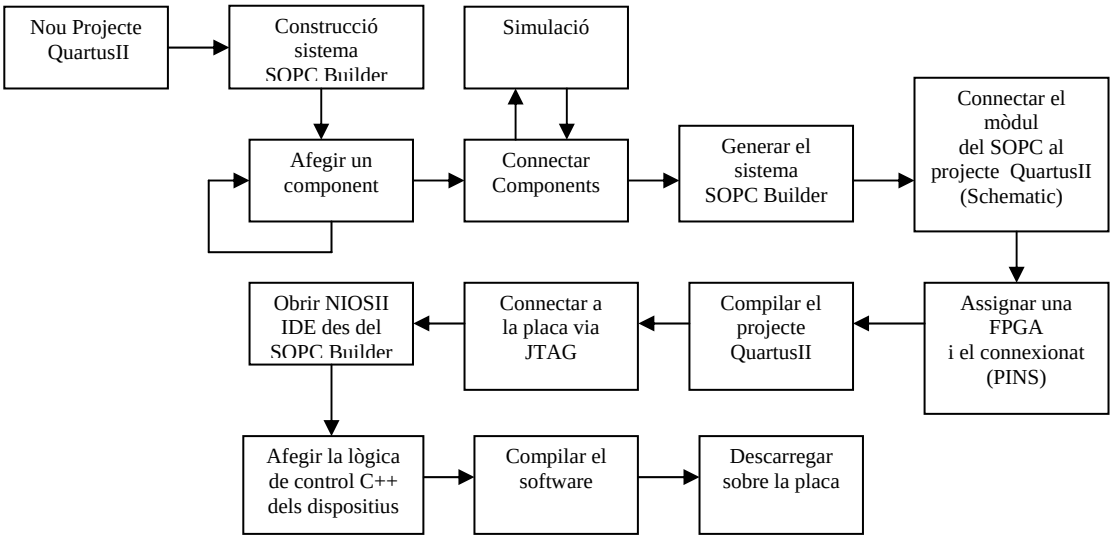

Figura 4.5 Diagrama del procediment emprat

Com es pot comprovar en el següent diagrama, hi ha una vinculació directa amb els passos proposats en l'apartat 3 per a una definició metodològica per a la creació d' *IP Cores* i a la utilització dels mateixos.

Primer de tot, obrim un nou projecte amb l'eina QuartusII.

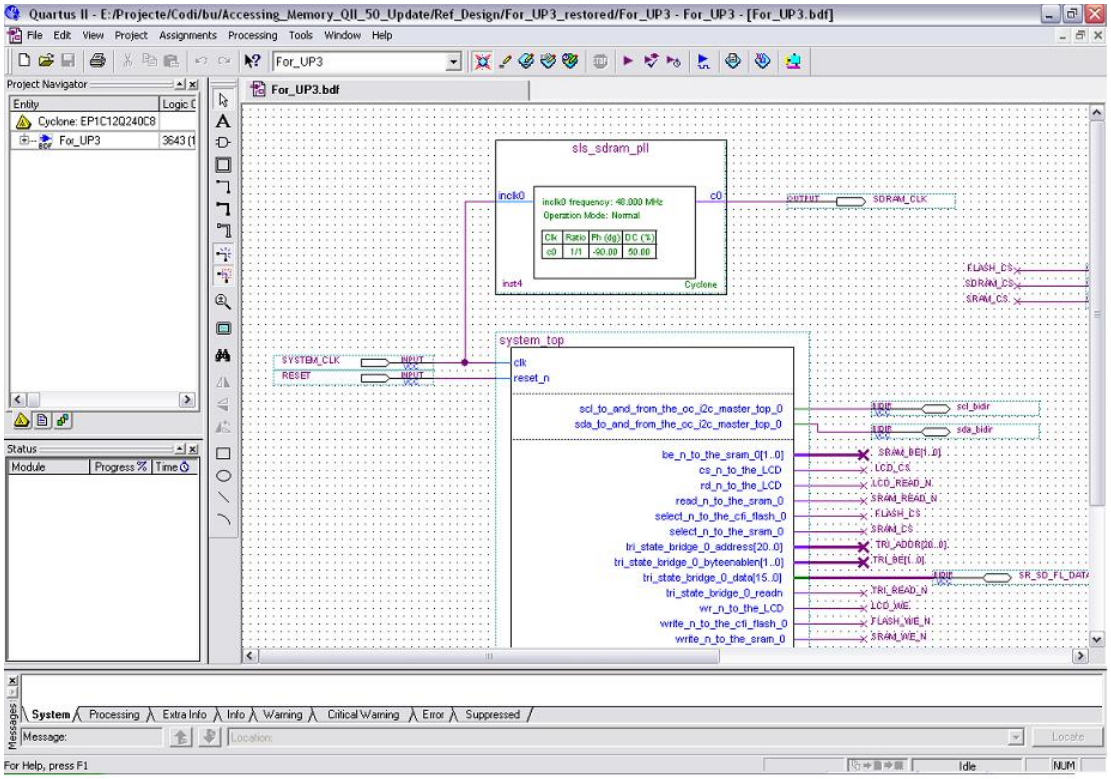

Figura 4.6 QuartusII v5.1 Web Edition Full

Amb aquesta eina que es pot observar en la figura anterior s'obre i es crea o carrega el projecte en qüestió. A la figura també es pot veure els mòduls afegits al projecte (en forma d'esquemàtic) , amb les seves respectives connexions. Els mòduls han estat creats pel SOPC Builder que s'obre amb botó situat a la part superior dreta de la barra d'eines del Quartus.

Si observem el diagrama de la **Figura 4.1**, es pot comprovar com el QuartusII és l'encarregat de la realització dels passos centrals del diagrama, tal i com s'explica en el paràgraf anterior.

Tot seguit s'obre el SOPC Builder.

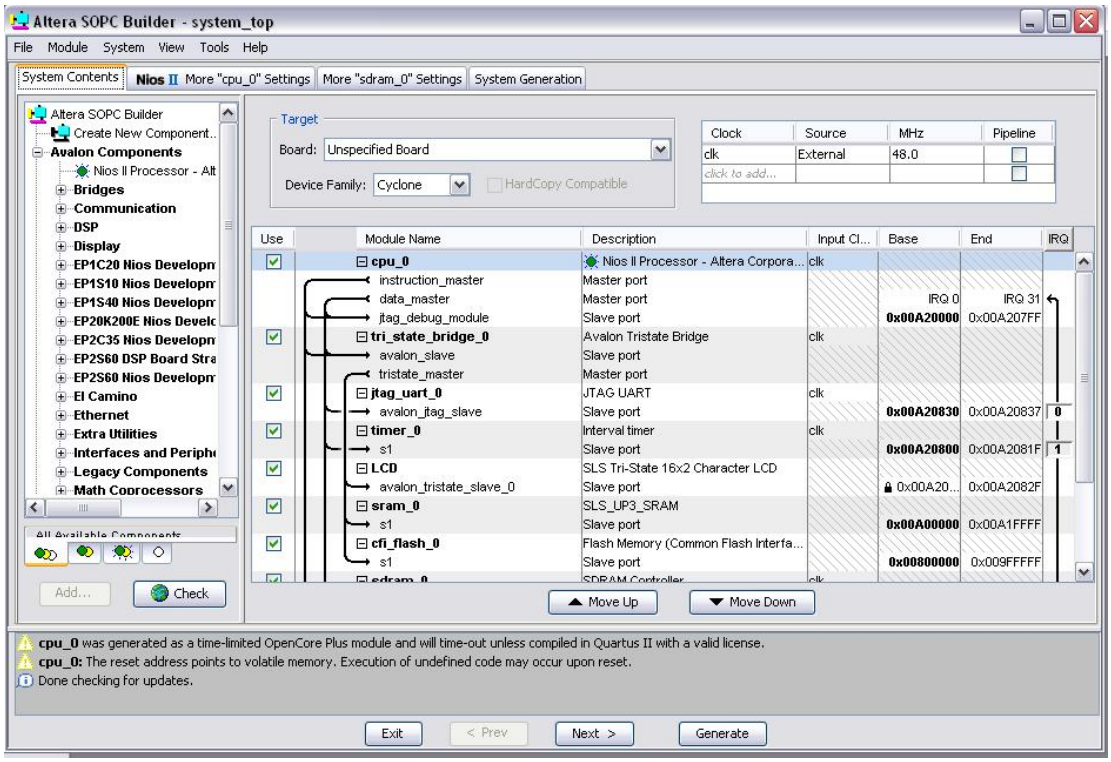

Figura 4.7 SOPC Builder

Observant aquesta figura, es pot veure el SOPC Builder el qual s'encarregarà de realitzar els primers passos per a la creació d'un sistema complet. És aquí on s'afegiran components nous pel sistema, així com la possibilitat de crear-ne i afegir-ne de nous. En aquesta part doncs, es definirà el sistema, i es creara un mòdul que es gestionarà amb el QuartusII.

Aquest component serà el més important del projecte, tenint en compte l' arquitectura comentada en l'**Apartat 4.1.1**. A part d'això és on definirem i afegirem els components creats al sistema.

Des d'aquesta mateixa aplicació podem obrir el NIOS II IDE.

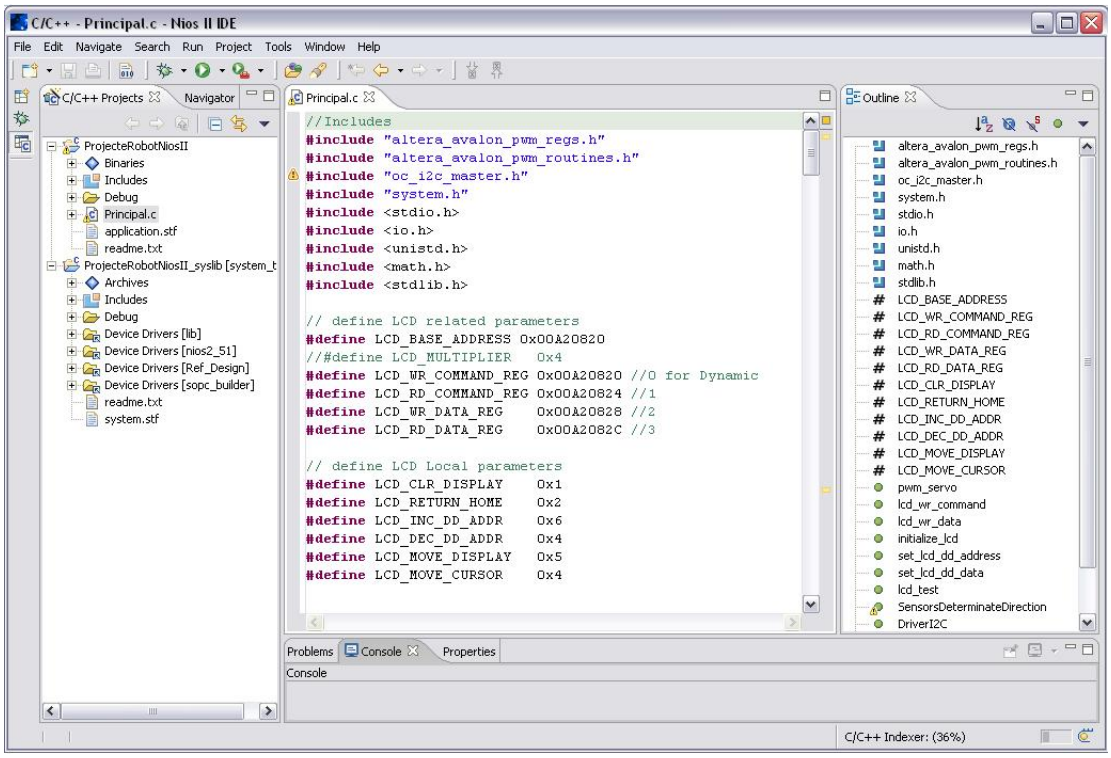

Figura 4.8 NIOS II IDE

Finalment, en aquesta aplicació, es crearà el projecte software a partir del sistema generat. Aquest software es compilarà i es descarregarà sobre la placa perquè pugui actuar sobre els components que haguem afegit al sistema i gestionar-los.

Val a dir que per a una millor comprensió del procediment seguit, es necessari llegir els manuals, els quals es fa referència en la bibliografia d'aquest projecte, en l'apartat d' Altera i el seu entorn.

4.2 Disseny dels *IP Cores* en la plataforma NIOSII

En aquesta secció s'explicaran els passos seguits en el disseny dels dos *IP Cores* implementats. Els passos que s'han seguit són els explicats a l' **Apartat 3** d'aquest projecte, utilitzant les eines facilitades per Altera explicades en la secció anterior.

Cada *IP Core* implementat té una sèrie d'especificacions i necessita d'una sèrie d'accions per a convertir-se en un component, tal i com s'ha explicat en apartats anteriors. Primer de tot, es necessita l'element encarregat de efectuar les tasques lògiques i que descriurà el hardware. Un cop fet això, es necessari definir la funcionalitat del *core* a nivell dels registres, i finalment un cop instanciades les accions anteriors es necessitarà un component d'alt nivell que els recobreixi i que proporcioni la interfície necessària amb el bus Avalon.

4.2.1 *IP Core*: PWM (*Pulse Width Modulation*)

Pulse Width Modulation (PWM) és una tècnica de gran abast per a controlar els circuits analògics a través de les sortides digitals d'un microprocessador. El PWM s'utilitza en molts camps per a una varietat d'accions diferents, estenent-se cap a camps com les telecomunicacions, la regulació de voltatge, la conversió d' energia i els efectes i l'amplificació de l' àudio.

En el camp de les telecomunicacions, l'amplada del pols correspon a les dades específiques codificades a una banda i descodificades a l'altre. Els polsos tenen una llargada diferent però que seran enviats en intervals regulars. D'altra banda, en el camp de la regulació del voltatge el PWM és un eficient regulador, ja que

seleccionant un voltatge d'entrada determinat, juntament amb un cicle de treball adequat, la sortida serà un voltatge amb el nivell desitjat.

No obstant, en camps com els de la conversió d'energia es poden utilitzar per reduir la quantitat total d'energia entregada a una carrega sense tenir en compte les pèrdues que es tenen quan una font d'energia és limitada per temes resistius. Això és així, perquè l'energia mitjana entregada és proporcional al cicle de treball de la modulació. D'aquesta manera amb una modulació suficientment alta, els filtres electrònics passius es poden utilitzar per a aïllar un tren de pols i recuperar una forma d'ona mitja.

Per altra banda, i com a últim exemple de camp on s'utilitzen cal esmentar que a vegades s'utilitza en la síntesis del so. Dóna un efecte de so semblant al de una tornada en una cançó, o als oscil·ladors lleugerament desintonitzats sonant a la vegada. Un exemple de la utilitat en aquest camp són els amplificadors de *classe-d*

Així doncs, aquest *driver* consisteix en què a partir de la modulació d'uns senyals analògics som capaços de moure components *hardware* com ara motors, que evidentment funcionen amb aquest tipus de senyals.

Per fer variar el seu comportament, s'ha de variar el senyal del seu cicle de treball, així que depenent de la carrega que se li indiqui treballarà d'una forma o d'un altre.

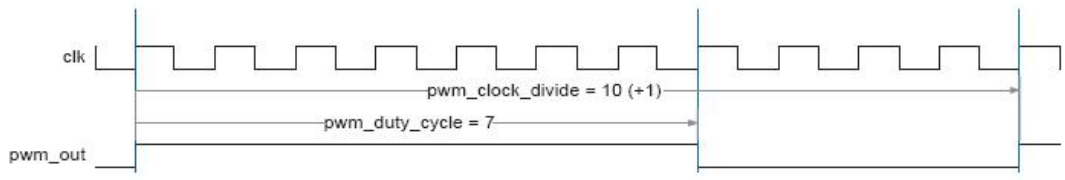

Figura 4.9 El senyal bàsica del *Pulse Width Modulation*

En aquesta figura es pot veure la forma del senyal PWM. Un component d'aquest tipus genera com a sortida una ona quadrada amb la modulació indicada, depenent del cicle de treball que se li hagi aplicat.

El component que s'ha especificat i creat per a aquest projecte segueix les següents especificacions:

- La tasca lògica opera de forma síncrona amb l'únic rellotge existent.
- La tasca lògica utilitza comptadors de 32 bits per a oferir un rang prou ampli per a configurar diversos períodes PWM i cicles de treball.
- El processador serà l'encarregat d'assignar el valor del període PWM i del cicle de treball. Aquests requeriments impliquen la necessitat de llegir i escriure en la interfície per a controlar la lògica.
- Registrar els elements que estan definits per a mantenir el valor de període i el valor del cicle de treball del PWM.
- El processador pot parar la sortida PWM utilitzant el bit de control *enable*.

A continuació, es pot observar l'esquema del component implementat, seguint l'estructuració anterior.

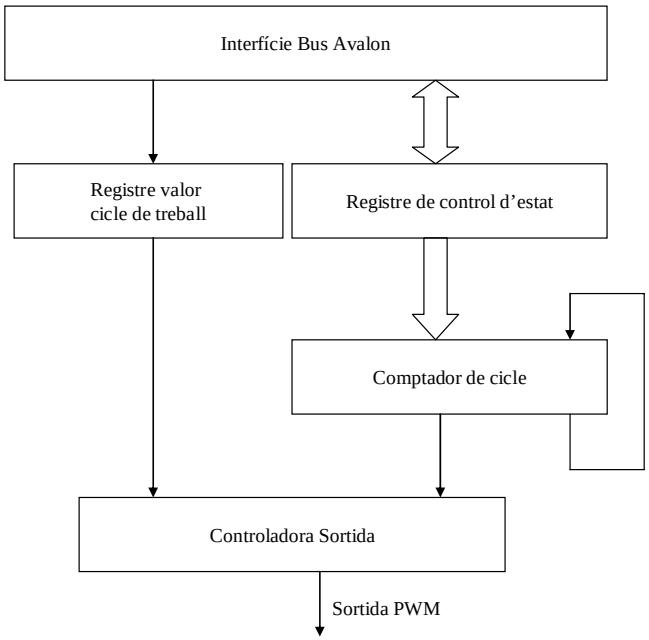

Figura 4.10 Arquitectura del *IP Core: Pulse Width Modulation*

Tal i com es pot observar en la figura anterior, el *IP Core* en qüestió, està dotat de les tres seccions esmentades. La tasca lògica la composen els comptadors de cicles i la controladora de sortida, la qual s'encarregarà de generar el senyal de sortida de pols. Per altra banda, els registres contindran la informació de del valor del cicle de treball amb el qual es treballarà i si està o no actiu. Finalment, la interfície amb el bus Avalon serà l'encarregada de traduir els senyals que li arriben per uns altres els quals el *driver* podrà treballar-hi.

La tasca lògica d'aquest component té les següents característiques:

- La tasca lògica consisteix en un rellotge d'entrada (clk), un senyal de sortida (pwm_out), un bit d' *enable*, un comptador de 32-bits i un circuit comparador també de 32-bits.
- Els *drivers* del rellotge, estableixen el període d'un senyal pwm_out.
- El comparador compara el valor actual amb el cicle de treball i determina la sortida del pwm_out.
- Quan el valor actual és inferior o igual que el cicle de treball, la lògica del pwm_out és 0, per altra banda, el valor lògic del *driver* serà 1, en altres casos.

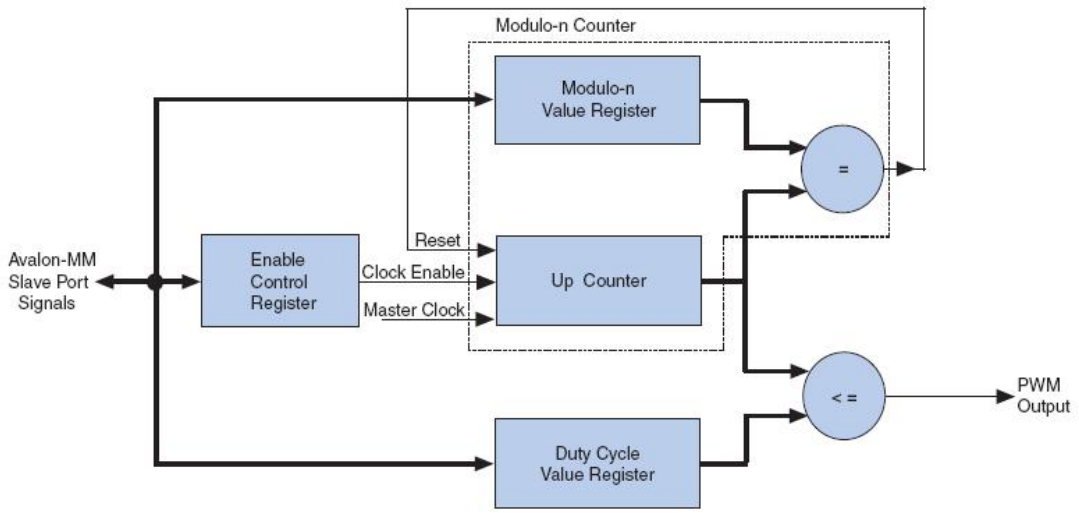

Figura 4.11 Estructura de la tasca lògica del PWM

Pel què fa als registres, són els encarregats de facilitar l'accés al bit d'*enable*, el mòdul del registre que conté el valor i el cicle de treball, tal i com es pot veure en la figura anterior. El mapeig de cada registre cap a una únic *offset* en l' espai de memòria del bus Avalon és imprescindible. Cada registre ha de poder tenir permisos de lectura i escriptura, el què significa que el *software* pot llegir valors antics prèviament escrits als registres. D'aquesta manera podran interactuar els dispositius amb el *software*, ja que el processador podrà llegir en memòria el què li diu cada

component esclau. Aquesta és una consideració que s'ha tingut en compte aquí, tot i que també es podria plantejar que els registres fossin només d'escriptura, cosa que faria que es conservaria la lògica dels dissenys *on chip*, però d'altra banda faria impossible pel *software* llegir els valors dels registres, impossibilitant la interacció.

El fitxer dels registres conté bàsicament els registres que es mostren en la taula següent, on es pot comprovar que es necessitaran dos bits de codificació per a un correcte adreçament.

Nom del registre	Offset	Accés	Descripció
clock_divide	00	Lectura/Escriptura	El numero de cicles de rellotge comptats durant un cicle de sortida del PWM
duty_cycle	01	Lectura/Escriptura	El numero de cicles de rellotge en els quals la sortida PWM serà baixa
enable	10	Lectura/Escriptura	Habilita/Deshabilita la sortida PWM. Posant el bit de 0 a 1 s'habilita el PWM
reservat	11	-	

Figura 4.12 Fitxer de registres i mapeig de les adreces

Per a poder llegir o escriure els registres tan sols es requereix un cicle de rellotge, que afectarà als estats d'espera de la interfície Avalon.

La interfície Avalon per al component PWM requereix un port esclau, utilitzat per a efectuar les lectures i les escriptures en les transferències als registres. El port esclau del component té les següents característiques:

- És síncron amb el rellotge del port esclau.
- Es pot llegir i escriure en ell.
- No té estats d'espera per a la lectura i l'escriptura, perquè els registres estan capacitats de respondre a les transferències en un cicle de rellotge.
- No hi ha restriccions per realitzar una lectura i/o escriptura.
- Latència en la lectura no és necessària, perquè totes les transferències poden realitzar-se en un cicle de rellotge.

Així doncs, la llista de senyals requerits per a la implementació d'aquestes propietats de les transferències són les que es poden veure a continuació, i que també són el nom dels senyals que estan definits en el disseny HDL.

Nom del senyal en HDL	**Tipus de senyal Avalon**	**Amplada de bits**	**Direcció**	**Descripció**
Clk	clk	1	Entrada	Rellotge que sincronitza les dades de les transferències amb la tasca lògica
resetn	reset_n	1	Entrada	Senyal de reset; activa a baixa
avalon_chip_select	chipselect	1	Entrada	Senyal de chip-select
address	address	2	Entrada	2-bit d'adreça; només 3 codificacions són possibles
write	write	1	Entrada	Senyal d'enable d'escriptura
write_data	writedata	32	Entrada	32-bit valor de les dades a escriure
Read	read	1	Entrada	Senyal d'enable de lectura

read_data	readdata	32	Sortida	32-bit valor de les dades a llegir

Figura 4.13 Nom dels senyals del PWM i tipus de senyals del bus Avalon

I Finalment, i per a completar el component és necessari disposar dels fitxers de capçalera que defineixen el mapeig dels registres, i un *driver software* pel processador NIOSII. Les funcions del *driver* s'exposen a continuació:

Funció	Descripció del tipus
Altera_avalon_pwm_init();	Inicialitza el hardware del PWM
Altera_avalon_pwm_enable();	Activa la sortida del PWM
Altera_avalon_pwm_disable();	Desactiva la sortida del PWM
Altera_avalon_pwm_change_duty_cycle();	Canvia el cicle de treball del PWM

Figura 4.14 Funcions del *driver* PWM

Aquestes funcions permeten la comunicació entre l'alt nivell i el component, facilitant la feina del dissenyador. Un cop s'està treballant a alt nivell, a través del NIOS II IDE, i durant el procés de disseny d'una lògica de control es poden utilitzar aquestes funcions. Tal i com la descripció indica, cada funció realitza el treball indicat.

Un cop arribats en aquest punt, ja es tenen totes les especificacions del *driver* PWM. Tan sols faltarà realitzar la unió per a obtenir un component únic, que serà el que es podrà unir al sistema, i utilitzar-lo segons les necessitats desitjades.

4.2.2 ***IP Core*: I2C (*Inter-Integrated Circuit*)**

El bus **I2C** és un bus de comunicacions *multi-master* sèrie inventat per Philips. El seu nom prové de *Inter-Integrated Circuit*. S'utilitza per afegir perifèrics de baixa velocitat a la placa base, al sistema o en altres dispositius.

És un bus molt utilitzat en la indústria, principalment per a comunicar microcontroladores i els seus perifèrics en sistemes empotrats (*embedded),* i generalitzant més, per a comunicar circuits integrats entre sí que normalment resideixen en un mateix circuit imprès. És un bus que encaixa molt bé en aplicacions que requereixen la comunicació entre dispositius que estan en distàncies curtes. A més a més, detecta col·lisió, arbitra i prevé de la corrupció de les dades si dos dispositius *master* intenten agafar el control del bus de forma simultània.

Els dispositius connectats al bus *I2C* tenen una adreça única. Cada un d'ells pot ser *master o slave*. El dispositiu *master* inicia la transferència de dades i és a més a més el què genera el senyal de rellotge., però no és necessari que el *master* sigui sempre el mateix dispositiu, ja que és una característica que poden adoptar tots els dispositius que tinguin aquesta capacitat, i per això se l'anomena *multi-master.*

La principal característica del I2C és que tan sols utilitza dos fils els quals li permeten transmetre la informació de forma bidireccional. Per un van les dades i per l'altre el senyal de rellotge que serveix per a sincronitzar-les. També és necessària una tercera línia que és la de terra. Com s'acostumen a comunicar circuits d'una mateixa placa que comparteixen el mateix terra, aquesta línia no sol ser necessària.

Les línies s'anomenen SDA (*Serial DAta line*), i SCL (*Serial CLock line*). Les dades són transmeses entre el dispositiu *master* i el dispositiu *slave* de forma síncrona a través de la línia SCL cap a la línia SDA *byte a byte*. Cada *byte* de dades té una mida de 8 *bits*. Cada pols de la línia de rellotge SCL es transmetrà un bit i l'ordre serà del

més significatiu al menys. Un bit d' *acknowledge* seguirà cada transferència de *bytes.* Cada bit és transmès durant el període alt de la línia SCL; per tant la línia SDA ha de canviar únicament durant el període de baixa de la línia SCL que ha de mantenir-se estable, durant el seu període a alta. Una transició en la línia SDA mentre la línia SCL està a alta és interpretat com un senyal de inici o fi de transmissió.

D'aquesta manera el protocol de comunicacions té la forma següent, i es detalla a continuació:

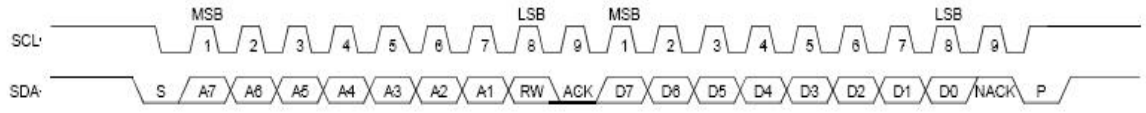

Figura 4.15 Protocol I2C

- El bus està lliure quan les línies SDA i SCL estan l'estat lògic alt.
- En l'estat de bus lliure, qualsevol dispositiu pot ocupar el bus I2C com a *master.*
- El *master* comença la comunicació enviant un senyal que indica l' inici d'una transmissió de dades. Aquest senyal consisteix fer una transició de altes cap a baix de la línia SDA mentre la SCL segueix estan a l'estat lògic alt. Això activa els dispositius esclaus que esperen una transmissió. La repetició d'un senyal de inici sense generar un senyal de STOP, significaria que el *master* vol comunicar-se amb un altre dispositiu *slave* o amb el mateix en una direcció de transmissió diferent, sense alliberar el bus.
- El primer byte de dades transmès pel *master* immediatament després del senyal d'inici és l'adreça del dispositiu esclau. Hi ha 7 bits d'adreça seguits per un bit de lectura/escriptura. Aquest bit indica la direcció de la transmissió de dades. Dos components esclau no poden tenir la mateixa adreça en el mateix sistema. D'aquesta manera tan sols el esclau que tingui

l'adreça corresponent transmetrà un bit d' *acknowledge* activant la línia de dates a baixa al novè cicle de rellotge del SCL.

- Una vegada finalitzada l'etapa d'adreçament ja es pot procedir a la transferència de dades *byte a byte*, en la direcció especificada pel bit de RW enviat pel *master*. Cada transferència de *bytes* és seguida per un bit d' ACK al novè cicle de rellotge de la línia SCL. Si per algun motiu, el dispositiu esclau envia un senyal de NACK, el dispositiu *master* pot generar un senyal de STOP que abortarà la transferència de dades o generarà un altre START per iniciar un nou cicle de transferència. Per a escriure dades a l'esclau, s'ha d'emmagatzemar les dades a ser transmeses en el registre de transmissió i activar el bit de WR.
- Finalment, el *master* pot acabar una comunicació, generant un senyal de STOP. Un senyal de STOP es pot generar realitzant una transició de baixa a alta de la línia SDA mentre la línia SCL està en l'estat lògic 1.

Un cop vist el protocol de transmissió de dades a través del bus, i tal i com s'ha comentat en l' inici, el bus *I2C* és un bus que permet que varis dispositius *master* estiguin connectats a ell. Si dos o més dispositius intenten controlar el bus de forma simultània, un procediment de sincronització del rellotge es posa en marxa i determina el rellotge del bus. Les transicions d'alta a baixa afecten a tots els dispositius connectats al bus, per tant un cop un dispositiu hagi fet la transició a baixa mantindrà la línia SCL en aquest estat fins que el rellotge no arribi a l'estat d'alta.

Veient això s'entén que la línia SCL és una línia que és la conjunció de totes les sublínies SCL que cada dispositiu *mater* té. En la figura que es mostra a continuació és pot observar aquest procediment.

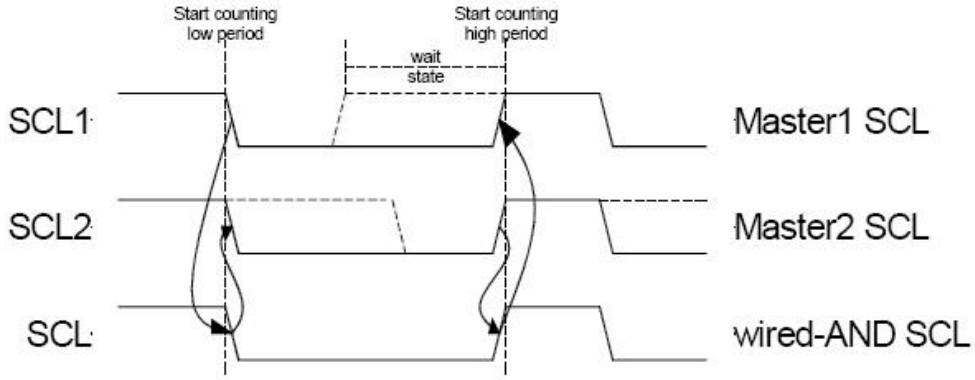

Figura 4.16 Arbitratge línia SCL

D'altra banda els dispositius esclau poden utilitzar el mecanisme de sincronització del rellotge per a alentir la transferència de bits. Una vegada el *master* a posat a baixa el senyal SCL, l' esclau pot posar el senyal SCL a baixa per un període de temps requerit i tot seguit alliberar-lo. Si el període a baixa del senyal SCL de l'esclau és més gran que la del *master*, el resultat és que el senyal SCL del bus a baixa s'estira. És així com s'insereix un estat d'espera.

Així doncs, després de comentar com funciona el bus *I2C* anem a centrar-nos en l'arquitectura del component i al què faran cada una de les parts d'aquest. El *IP Core* construït a partir de 3 blocs: el primer el formen el generador de rellotge, la controladora de *bytes*, la controladora de *bits*, que bàsicament corresponent a la tasca lògica del component. El segon bloc és que s'encarrega de comunicar-se amb els registres, o sigui que transmet els senyals des de la lògica de control fins a l'exterior, i finalment la interfície del bus Avalon, que és el tercer bloc i s'utilitzarà per a realitzar la transferència requerida a través de la tasca lògica tal i com s'ha definit en l' **Apartat 3**.

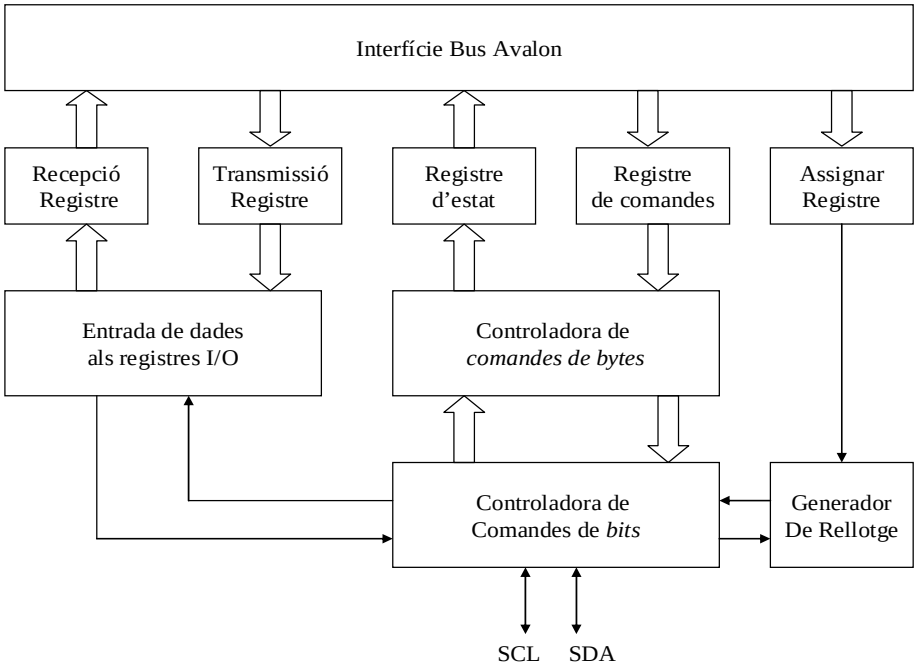

Figura 4.17 Arquitectura del IP Core I2C

El generador de rellotges s'encarrega de generar els senyals d'*enable* que sincronitzarà els elements a transmetre en el transmissor de bits. D'altra banda també s'encarrega d'alentir les transferències per a dispositius més lents, aguantant el cicle de rellotge.

El controlador de *bytes* agafa el trànsit de dades de nivell de byte. Agafa les dades del registres de comandes i les tradueix en seqüències basades en la transmissió d'un *byte.* Tot seguit s'encarrega d'enviar aquest *byte* trossejat en forma de bits cap a la controladora de comandes de bits.

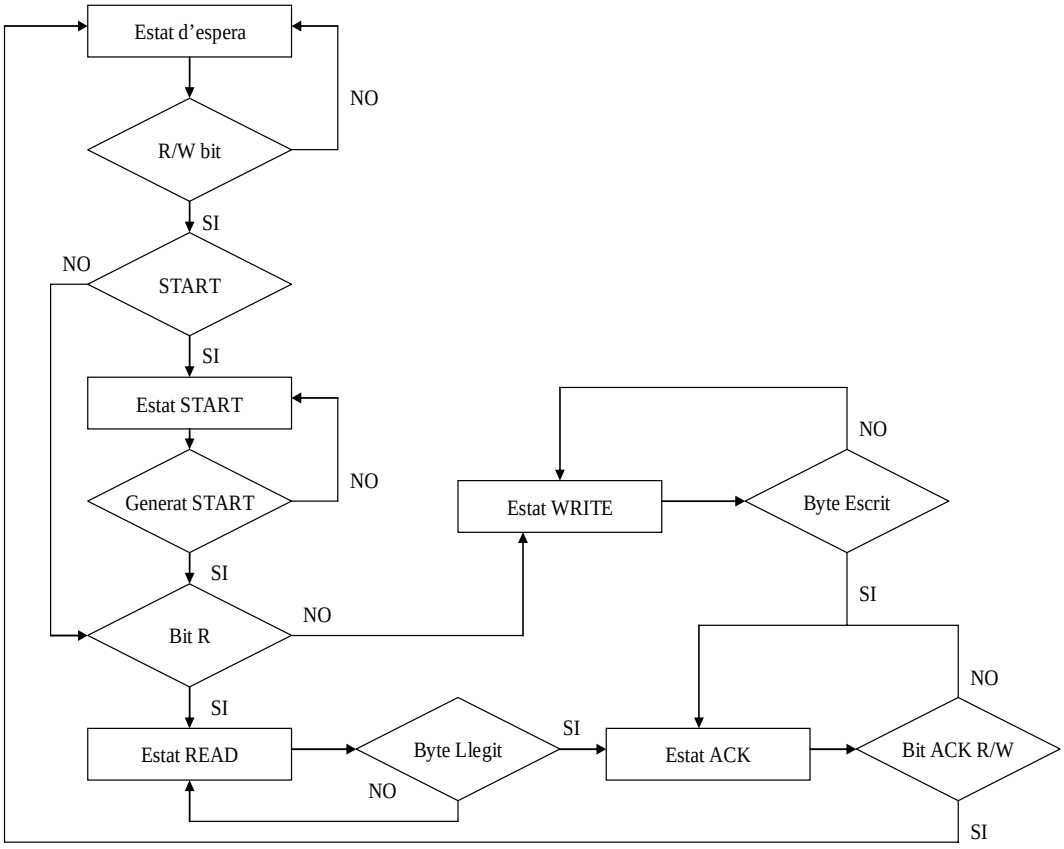

Figura 4.18 Diagrama de flux de la controladora de *bytes*

La controladora de comandes de bits agafa la transmissió actual de dades i genera els senyals específiques de inici, un inici repetit i aturada controlant les línies SCL i SDA. La controladora de *bytes* li diu a la de bits quina operació ha de realitzar. Per a una simple lectura, la controladora de bits rep 8 comandes de lectura . Cada operació de bit és dividida en 5 parts, exceptuant la de STOP que és divideix en 4 tal i com es veu en la figura següent, on les parts són *idle*, A, B, C i D.

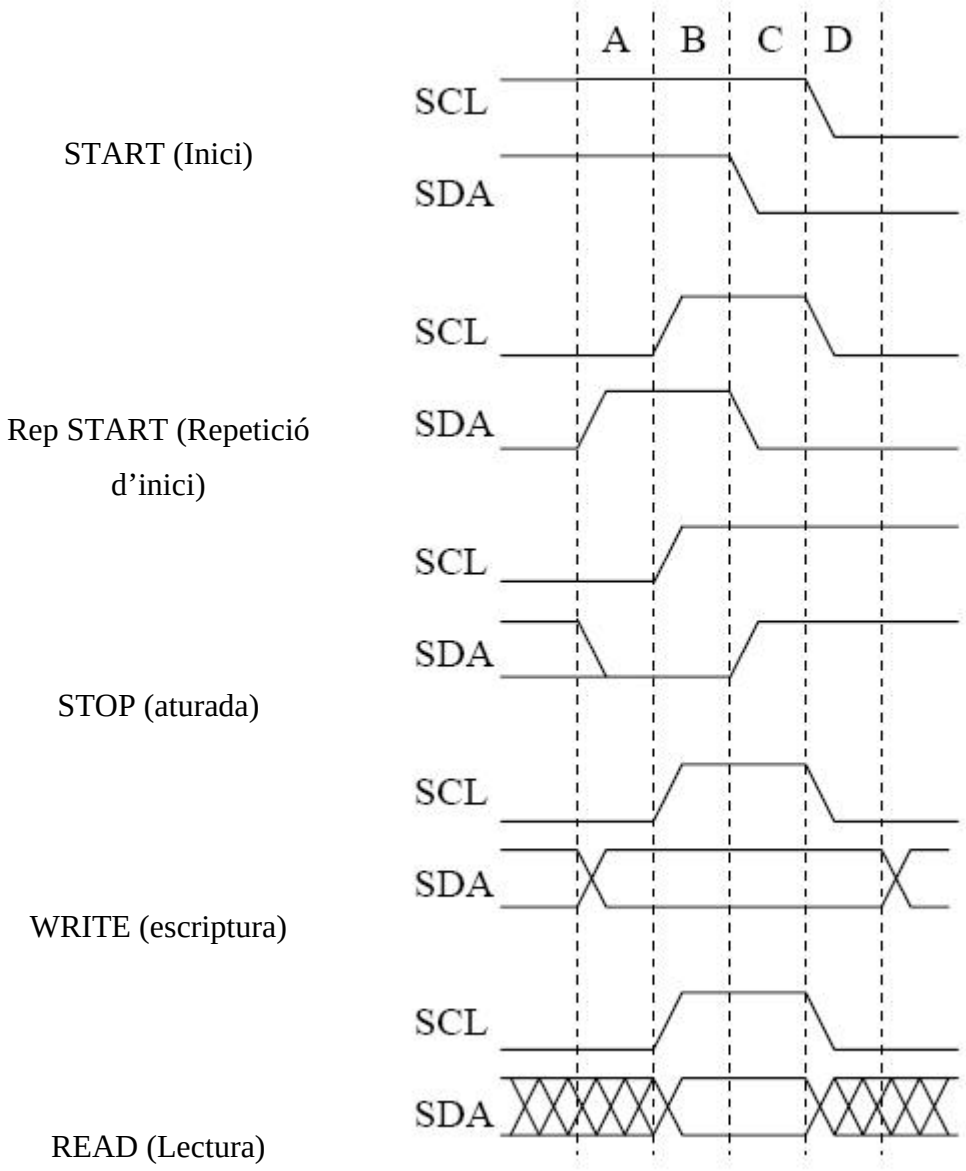

Figura 4.19 Controladora de comandes de bits

El component d'entrada de dades al registre conté les dades associades a la transferència actual. Durant una acció de lectura, entra en la línia SDA. Després que un *byte* ha estat llegit els continguts són copiats al registre de recepció. Durant una acció d'escriptura, els continguts del Registre de Transmissió són copiats en aquest mòdul i són transmesos a la línia SDA.

Així doncs, un cop analitzat el component, a nivell funcional, s'ha d'examinar el conjunt de senyals que el componen. Els primers senyals que toca analitzar són les referents a la interfície i el mapeig que es fa d'elles per tal que el bus Avalon pugui comunicar-s'hi, així com les que té la interfície i utilitza per comunicar-se amb el registres.

Nom dels senyals en HDL	**Tipus de senyal Avalon**	**Amplada de bits**	**Direcció**	**Descripció**
wb_clk_i	Clk	1	Entrada	Rellotge *master*
wb_rst_i	'0'	1	Entrada	Senyal síncrona de *reset*, activa a alta
arst_i	reset_n	1	Entrada	Senyal de *reset* asíncrona
wb_adr_i	address	3	Entrada	Adreça de bits
wb_dat_i	writedata	8	Entrada	Dades cap al dispositiu
wb_dat_o	readdata	8	Sortida	Dades des del dispositiu
wb_we_i	write	1	Entrada	Senyal d'*enable* d'entrada
wb_stb_i	chipselect	1	Entrada	Senyal de selecció d'entrada
wb_cyc_i	chipselect	1	Entrada	Senyal de cicle de bus vàlid
wb_ack_o	waitrequest_n	1	Sortida	Senyal d' ACK de sortida
wb_inta_o	irq	1	Sortida	Senyal d'interrupció de sortida

Figura 4.20 Senyals de la interfície de comunicació amb els registres

Pel què fa al bus Avalon només te contacte amb els senyals exteriors. Els senyals que necessita són els SDA i SCL ja comentats amb anterioritat. Com es pot veure en la figura següent hi ha dos senyals, més per línia, que s'utilitzaran en cas que hi hagi retorn de dades per part del dispositiu.

Nom del senyal en HDL	**Tipus de senyal Avalon**	**Amplada de bits**	**Direcció**	**Descripció**
scl_pad_i	scl_pad_i	1	Entrada	Línia d'entrada del senyal sèrie de rellotge
scl_pad_o		1	Sortida	Línia de sortida del senyal sèrie de rellotge
scl_pad_oe		1	Sortida	Línia de sortida d'*enable* del senyal sèrie de rellotge
Sda_pad_i	sda_pad_i	1	Entrada	Línia d'entrada del senyal sèrie de dades
Sda_pad_o		1	Sortida	Línia de sortida del senyal sèrie de dades
Sda_pad_oe		1	Sortida	Línia de sortida d'*enable* del senyal sèrie de dades

Figura 4.21 Senyals de la interfície amb el bus Avalon

Els senyals SDA i SCL són bidireccionals. Per tal de controlar-los per realitzar transferència de dades en ambdós sentits cal introduir un *buffer tri-state*, per a aquests senyals com a nivell jeràrquic més alt. Les connexions es faran d'acord a la figura que es mostra a continuació:

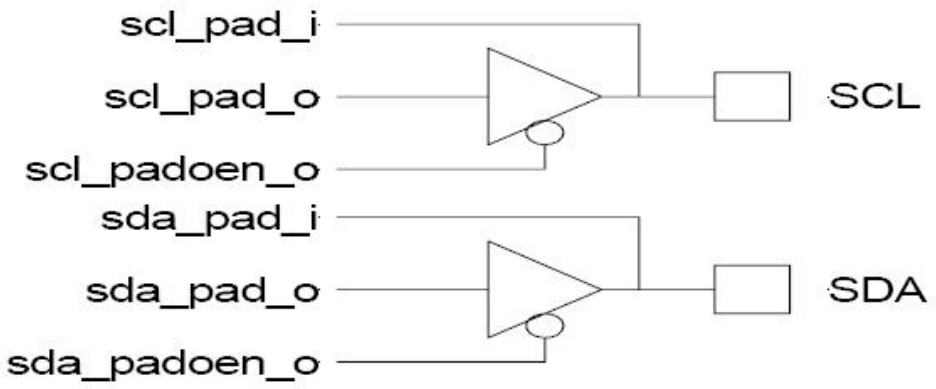

Figura 4.22 Bidireccionalitat senyals I2C

A través d'aquests *buffers*, es permet la transmissió de dades en un o altre sentit. D'altra banda, per a dissenys amb FPGA el compilador pot inserir de forma automàtica aquests *buffers* utilitzant codi en HDL com es mostra tot seguit a fi d'exemple.

```
scl <= scl_pad_o when (scl_padoen_oe = '0') else 'Z';
sda <= sda_pad_o when (sda_padoen_oe = '0') else 'Z';
scl_pad_i <= scl;
scl_pad_i <= sda;
```

Si es baixa de nivell s'arriba al nivell dels registres. En aquest nivell també hi ha una assignació d'aquests amb els mòduls que tracten la tasca lògica o *driver* del *IP Core.* Així doncs, a continuació s'enumeren el conjunt de registres del component i una breu explicació de cada un d'ells.

Nom del registre	Offset	Accés	Descripció
PRERlo	8	Lectura/Escriptura	Estableix el registre del rellotge a *byte* baix
PRERhi	8	Lectura/Escriptura	Estableix el registre del rellotge

			al *byte* alt
CTR	8	Lectura/Escriptura	Control del registre
TXR	8	Escriptura	Registre de Transmissió
RXR	8	Lectura	Registre de Recepció
CR	8	Escriptura	Registre de comandes
SR	8	Lectura	Registre d'estat

Figura 4.23 Descripció dels registres

El registre d'assignació de registre PRERlo i PRERhi, s'utilitzen per assignar el rellotge a la línia SCL. S'assignen les freqüències i es canvien els valors assignats del registre tan sols si el bit d'*enable* està actiu.

El registre de control respon a les noves comandes només quan el bit d'*enable* està actiu, i informa quan les comandes han finalitzat. El senyal d'*enable* desactivat indicarà que no hi ha transferències actives.

Pel que fa als registres de transmissió i de recepció contenen la informació en forma de *byte* que s'ha transmès o rebut, en cada cas. En el cas de la transmissió té la particularitat que el bit menys significatiu indica que si hi ha un 1, és que la transmissió és per a llegir de l'esclau, mentre que si hi ha un 0 és per escriure-hi.

4.3 Resultats, problemes i solucions

Durant el desenvolupament d'aquest projecte han anat sorgint diversos imprevistos que es comentaran en aquest apartat. Com ja s'ha comentat tan en la introducció, i tal i com s'havia contemplat en l'estudi de viabilitat del projecte, el fet de treballar amb hardware és una dificultat afegida, ja que aquest sempre pot provocar algun error, així com que el software que el gestioni tingui algun tipus d'error a causa d'una mala programació.

El primer problema greu, es va trobar al principi. Durant el procés d'anàlisi, en el qual es varen realitzar diversos sistemes de prova, l'entorn d'Altera va començar a actuar de forma estranya. El problema residia en la impossibilitat de poder descarregar el sistema creat a la placa, i un cop la llicència de l'aplicació es va renovar per caducada el comportament encara va empitjorar. La reinstal·lació de l'entorn i de tots els components instal·lats, va solucionar el problema. A data d'avui, encara es desconeix l'arrel del problema, però aquest cas es va reproduir per a 3 ordinadors més, fet realment sorprenent i que encara va desconcertar més. La pèrdua de temps en aquest cas va ser notable.

Els problema generat a partir del punt anterior té l'estructura que es mostra a continuació. Durant el pas del sistema a la placa en a través del NIOSII IDE es produïa el següent error, tot i compilar tot el sistema de forma correcta.

```
Using cable "ByteBlaster [LPT1]", device 1, instance 0x00
Pausing target processor: OK
Reading System ID at address 0x02120FFF: verified

Downloading 0A000000 ( 0%)
Downloading 0A000020 (99%)
Downloaded 86KB in 1.1s (78.1KB/s)
```

```
Verifying 03000020 ( 0%)
Verify failed
Leaving target processor paused
```

Per tant, el processador mai acabava d'arrencar i executar el sistema. D'altra banda, també s'ha donat el problema següent referent a problemes amb JAVA.

```
(SEVERE) generate: java.lang.IllegalStateException:
java.lang.IllegalStateException
```

La reinstal·lació després d'aquest tipus d'errors es fa necessària, ja que tenint tots els components instal·lats correctament, i sense haver efectuat cap acció susceptible d'error, aquests problemes resulten greus i de difícil solució.

Permisos de lectura i escriptura. La majoria de fitxers que gestionen les aplicacions QuartusII, SOPC Builder i NIOSII IDE necessiten permisos d'escriptura. És necessari, ja que aquestes eines escriuen en els fitxers com a part del procés de generació i compilació, i per tant si no tenen algun d'aquests permisos les eines fallen, provocant errors inesperats. Un error d'aquest estil s'ha trobat intentant obrir el projecte: *Error: Can't open project -- you do not have permission to write to all the files or create new files in the project's database directory.* Aquest error, el provoca el Windows XP en conjunció amb Altera. El sistema operatiu, canvia els permisos del directori del projecte a *read only*. Si s'intenta modificar els permisos a mà, a través de les propietats de carpetes i fitxers del S.O, el Quartus a l' intentar obrir el projecte retorna els fitxers amb permisos de només lectura. La solució consisteix en crear un executable .bat, que contingui la següent informació:

```
attrib -R /S /D *
attrib -R /S /D .
```

Aquestes dues comandes faran que el sistema operatiu canviï realment els permisos dels fitxers que resideixen en el directori del projecte.

Cada sistema que es pugui crear en l'entorn d' Altera va directament lligat a la placa que s'utilitza. En el cas d'aquest projecte s'ha utilitzat la placa de la família Cyclone i més concretament el model EP1C12Q240C8. Així doncs, s'ha hagut d'adaptar el sistema a la placa en alguns aspectes, que en un principi no s'havien tingut en compte, així com indicar-li a l'aplicació de quina placa es tracta. El sistema creat conté un mòdul de memòria SDRAM, que no funcionava correctament al intentar interactuar-hi. El problema s'ha solucionat modificant la fase del rellotge a -90º, en comptes de -60,75º que es el valor amb el que es carrega el mòdul per defecte, i que provoca problemes crítics amb els *timings* del sistema.

De la mateixa manera, i per les particularitats del model de la placa s'ha tingut que modificar les connexions entre els pins d'aquesta. Per defecte es realitzen les assignacions que determina el QuartusII, encara que en ocasions no són correctes, o bé no s'adapten al sistema creat.

Els problemes esmentats en els últims dos paràgrafs es poden trobar documentats als *datashits* d'errors que es poden trobar a la pàgina web d' Altera.

La inclusió d'un element *Avalon Tristate Bridge*. Aquest component connecta dispositius *off-chip* cap a sistemes *on-chip* a través del bus Avalon. Aquest component ha solucionat el problema de comunicació de components *off-chip* inclosos al sistema com ara la pantalla LCD o memòria SRAM, ja que sense el dispositiu el sistema no permetia generar una solució correcta donant error. El pont en qüestió doncs, crea adreces i pins de dades que poden ser compartits per a múltiples dispositius *off-chip*. Aquesta opció permet conservar els pins del processador quan aquest es connecta a múltiples dispositius amb exclusió mútua d'accés.

La placa UP3 en l'entorn educatiu presenta un inconvenient de treball greu quan es volen fer aplicacions autònomes. Durant la planificació inicial d'aquest projecte es va plantejar la possibilitat que el sistema creat, gràcies als dos nous cores creats pogués ser autònom i actuar com un robot intel·ligent. La opció es mantenir fins que es va veure que era inviable. La problemàtica recau en el fet que la placa UP3 ha d'estar permanentment connectada al PC a través del port paral·lel. Aquesta connexió ha d'existir ja que sinó la placa deixa de funcionar. És un problema pel fet que no dóna un grau més de llibertat per a l'experimentació, amb aquest tipus de dispositius, i el fa menys flexible.

L' entorn d'Altera no destaca precisament per fer un ús racional o òptim de la memòria. Els entorns de programació que descarreguen els sistemes a la placa, utilitzen molts recursos, que les FPGA no acaben de gestionar de la millor manera. El mal ús d'aquesta memòria pot provocar problemes d'*overflow*.

Com a nota positiva, val a dir que es pot descarregar el sistema en el tipus de memòria que es desitgi poden interactuar amb la que més interessi segons el sistema programat. No obstant, les memòries que puguin tenir els sistemes no poden comunicar-se entre elles de forma directa, en cas de necessitar simultaneïtat en l'ús de memòries durant l'excussió d'una aplicació. Això, és un problema ja que condiciona el programador a l'hora de crear una aplicació fent que aquesta hagi de crear ponts de comunicació entre els diferents tipus de memòries que es volen utilitzar i esperar que no succeeixi cap error.

4.4 Proves

Aquest apartat pretén comentar les proves realitzades sobre l'entorn d'Altera. Per a comprovar el correcte funcionament dels *IP Cores* s'ha creat un sistema senzill que es composa d'elements de memòria, el processador, el connector JTAG, entre d'altres, així com els dos components que s'han creat.

Primer de tot, s'ha provat que el sistema funcionés correctament a nivell de totes les memòries que el composen, LEDs i pantalla LCD. Un cop comprovat que el sistema funciona correctament, s'han afegit els dos nous components. D'aquesta forma, un s'assegura que el sistema és correcte i que els possibles problemes que sorgeixin un cop afegits els components, no siguin a causa d'un sistema corrupte.

Per a provar el *driver* de PWM s'ha utilitzat un servo-motor. Aquest servo-motor funciona a través de la modulació de pols, cosa que encaixa amb el *core* creat. Les proves amb aquest component han funcionat de forma parcial. A través de les verificacions fetes sobre el *driver* s'ha comprovat que aquest funciona correctament, però al passar-lo com a component a la placa, aquest actua de forma estranya a causa del soroll que el servo provoca. El soroll EMI (*Electromagnetic Interference*) és tal que fa que el servo no acabi de girar en les direccions indicades, i per tan s'ha necessitat d'un altre sistema per a validar-lo.

Per visualitzar millor el comportament del *driver* en la placa UP3, s'ha redirigit la sortida del PWM a un LED situat a la placa. S'ha anat modificant el cicle de treball del PWM i d'aquesta manera es veu com el LED es va apagant i encenent segons el pols enviat. Així doncs, s'ha pogut validar el correcte funcionament d'aquest *IP Core*.

Per altra banda, per la validació del *driver* de I2C s'ha utilitzat un compàs magnètic. Aquest component es comunica a través d'aquest bus amb el sistema. S'han efectuat les proves pertinents, consistents en llegir les dades que transmet el compàs a través del bus.

5. CONCLUSIONS I MILLORES

5.1 Conclusions

Altera és l'empresa líder en la programació de dispositius lògics i programables, així com també, pionera en la utilització d'aquests dispositius. Sabent aquesta dada, hom té clar que una empresa d'aquesta envergadura ajustarà els seus productes a les necessitats dels usuaris, però partint d'una sèrie d'especificacions que es poden permetre el luxe d'escollir. Així doncs, la definició metodològica per a la creació de *IP Cores* ha estat basada a partir de l'estudi i la realització d'aquests cores, partint de les especificacions i necessitats que un entorn peculiar com el d'Altera proporciona.

La creació d'un manual, o d'una definició metodològica complerta a seguir per a la creació d'un component en aquest entorn, ha esdevingut una tasca realment complexa. A pesar de la gran quantitat d'informació que proporciona Altera, no hi ha un manual complet on es puguin trobar els passos concrets per a la creació d'un *IP Core*. Conseqüentment, la tasca d'analitzar l'entorn i els documents, així com el munt d'idees que es desprenen d'ells, ha dificultat el fet d'escriure aquests passos en un full de paper, a fi de tenir aquesta informació centralitzada.

La utilització de la solució SOPC (*System On a Programmable Chip*) que proporciona Altera ha fet que s'aprengués una manera de resoldre els problemes de disseny en la creació de sistemes. Utilitzant el SOPC en el sistema creat, s'han pogut veure el conjunt d'avantatges i inconvenients que són conseqüència de l'ús d'aquest entorn, així com de la seva arquitectura. Gràcies als dissenys de múltiples mòduls IP per a optimitzar el disseny hardware, s'ha pogut simplificar molt la feina de creació d'un sistema per a poder adaptar els nous *IP Cores* creats.

Pel que fa al disseny s'ha comptat amb unes eines de desenvolupament realment bones, a pesar dels problemes que han generat. El QuartusII i el SOPC Builder faciliten l'ús i la modificació de canvis en el hardware, depenent de l'aplicació que es vulgui crear. Per exemple, en el sistema desenvolupat en aquest projecte, la flexibilitat de l'entorn ha facilitat el fet de poder afegir dos nous components al sistema, així com decidir quins components s'hi afegien i quins no. Aquesta forma de disseny s'entén que és més econòmica que d'altres, ja que no s'ha de comprar hardware addicional per a completar el sistema creat, sinó que tan sols requereix canviar la configuració del software del processador NIOS.

No obstant, dóna la sensació que les eines de desenvolupament del software per a NIOS podrien ser molt millor. Per exemple, la codificació, la compilació i les eines per depurar estan separades, fet que dificulta al dissenyador treballar d'una forma eficient. Integrant aquestes eines, es garantiria una millora en l'eficiència de disseny.

Utilitzant el *kit* de desenvolupament de la plataforma NIOS, s'ha trobat relativament fàcil el fet de connectar els nous components a la lògica establerta per Altera. La interfície del bus Avalon no ha estat complicada d'entendre, encara que diferís de l'arquitectura tradicional. D'altra banda, i a pesar dels problemes, el *kit* UP3 es pot considerar com a una bona placa a nivell estudiantil, per la quantitat d'opcions que ofereix i la personalització que qualsevol usuari li pot donar per a crear el seu propi sistema.

D'aquesta manera, es considera que l'ús del *kit* de desenvolupament NIOS ha facilitat el desenvolupament del sistema plantejat a partir de la metodologia creada.

També és molt important remarcar el fet que els *IP Cores* dissenyats, poden ser utilitzats per a altres dissenyadors de components en els seus sistemes. La grandesa d'una descripció genèrica de creació d'aquests *drivers* recau, en gran part, en el fet de poder compartir dissenys perquè d'altres puguin gaudir d'aquest treball. És important,

en una societat com la d'avui dia, compartir, ja que no cal reinventar la roda cada vegada.

5.2 Millores

Es pot considerar que els objectius principals plantejats a l'inici s'han complert. La definició d'una metodologia per a la creació d' *IP Cores*, era el punt clau sobre el qual girava el projecte i s'ha pogut realitzar de forma satisfactòria. Per contra, hi ha hagut punts que són susceptibles de millora, sobretot els referents a la component pràctica d'aquest estudi.

A partir del projecte es deriva la metodologia que cal seguir pel desenvolupament de *drivers* més complexes com ara: un lector de targetes SD, el d'una pantalla LCD o el d'un teclat, podrien ser *IP Cores* més robusts que haurien aportat més sofisticació al projecte.

Quant al sistema creat, s'havia planejat un disseny de molta més envergadura o si més no, més robust a l'inici, però va resultar impossible de dur-lo a terme per culpa dels temps establerts i dels recursos dels que es podia disposar. En un altre ocasió, es farà un sistema més potent.

6. BIBLIOGRAFIA

Altera i el seu entorn

· *Altera*
http://www.altera.com/

· *Altera*
http://en.wikipedia.org/wiki/Altera

· *QuartusII HandBook version 7.0*
http://www.altera.com/literature/hb/qts/qts_qii54007.pdf

· *Entorn d'Altera*
http://forum.niosforum.com/forum/

· *Forum d' Altera*
http://www.alteraforum.com/index.php

· *NIOS Forum Community*
http://www.niosforum.com/

· *Soft-cores*
http://cephis.uab.es/resources/pdf/papers/JCRA_2006_UFCpp.pdf

· *NiosII assembler examples*
http://instruct1.cit.cornell.edu/courses/ece576/NiosII_asm/index.html

· *IP Core*
http://whatis.techtarget.com/definition/0,,sid9_gci759036,00.html

· *NIOSII IDE*
http://www.altera.com/literature/ug/ug_nios2_getting_started.pdf

· *NIOSII IDE*
http://www.altera.com/support/software/embedded/ide/sof-nios2_ide.html

· *SOPC Builder*
http://fpga4u.epfl.ch/wiki/SOPC_Builder

. *Rapid Prototyping of Digital Systems*
J.O. Hamblen, T.S Hall, and M.D. Furman, 2005

IP Cores

PWM

· *Altera PWM*
http://www.altera.com/

· *QuartusII HandBook version 7.0*
http://www.altera.com/literature/hb/qts/qts_qii54007.pdf

· *Pulse width modulation*
http://en.wikipedia.org/wiki/Pulse-width_modulation

· *Pulse width modulation*

http://es.wikipedia.org/wiki/Modulaci%C3%B3n_por_anchura_de_pulsos

· *Example of PWM*

http://modeemi.fi/~tuomov/ion/pwm.html

· *PWM Core overview*

http://www.opencores.org/projects.cgi/web/ptc/overview

· *PWM Control*

http://www.cpemma.co.uk/pwm.html

· *NIOS Forum*

http://www.niosforum.com/

I2C

· *Altera I2C*

http://www.altera.com/

· *I2C drivers*

http://www.linuxjournal.com/article/7136

· *I2C driver interface*

http://tmd.havit.cz/Projects/I2C/i2cdriver.htm

· *I2C driver*

http://mhonarc.axis.se/dev-etrax/msg06565.html

· *Nios Forum*

http://forum.niosforum.com/forum/

· *I2C opencores*

http://www.opencores.org/projects.cgi/web/i2c/overview

· *I2C Master*

http://www.slscorp.com/pages/ipi2cmastersls.php

· *I2C Master/Slave controller*

http://www.arasan.com/products/prod_overview/I2C_APB_ds_v1.0.pdf

· *I2C Slave*

http://www.slscorp.com/pages/ipi2cslavesls.php

· *I2C Bus*

http://www.i2c-bus.org/slave/

· *I2C Master Mode*

http://ww1.microchip.com/downloads/en/devicedoc/i2c.pdf

· *I2C interface*

http://www.lammertbies.nl/comm/info/I2C-bus.html

· *Using I2C*

http://www.robot-electronics.co.uk/htm/using_the_i2c_bus.htm

HDL i C++

· *HDL*
http://en.wikipedia.org/wiki/Hardware_description_language

· *VHDL Manual*
http://mikro.e-technik.uni-ulm.de/vhdl/anl-engl.vhd/html/vhdl-all-e.html

· *VHDL Manual*
http://www.ehu.es/Electronica_EUITI/vhdl/pagina/inicio.htm

· *VHDL Manual*
http://www.fm.vslib.cz/~kes/data/vhdl_ref.pdf

· *Verilog Manual*
http://www.inf.pucrs.br/~moraes/topicos/hdls/ver.pdf

· *Verilog Manual*
http://www.gte.us.es/usr/chavez/verilog.pdf

· *C++ NIOS IDE Manual*
http://www.quantum-leaps.com/doc/QDK_Altera-NiosII.pdf

. The Designer's Guide to VHDL
Peter J. Ashenden, 1995