RESUM

Aquest projecte consisteix en el desenvolupament d'estructures hardware digitals, sintetitzables sobre FPGA i realitzades des d'un entorn gràfic de disseny a nivell de sistema (alt nivell). S'ha escollit el *Simulink* (entorn gràfic que treballa sobre el software matemàtic *Matlab* de *Mathworks*) com a entorn de disseny, i que gràcies a la interfície proporcionada per *Altera* (*DSPBuilder*) és capaç de generar codi VHDL sintetitzable.. Concretament ens centrarem en la gestió d'un sistema capturador d'imatges de comptadors del cabal d'aigua, en el qual volem fer la caracterització del comptador. Aquest capturador consta bàsicament d'un sensor d'imatge i una FPGA. En aquesta caracterització el que es pretén es ajustar els diferents paràmetres del sistema per fer que la lectura sigui òptima per a cada model de comptador que existeixen al mercat, com ara l'exposició del sensor, el guany d'un color, la realització d'un filtrat de la imatge, etc.

RESUMEN

Este proyecto consiste en el desarrollo de estructuras hardware digital, sintetizable en una FPGA y realizado desde un entorno de programación gráfico a nivel de sistema (de alto nivel). Se ha escogido el *Simulink* (entorno gráfico que trabaja sobre el software matemático *Matlab* de *Mathworks*) como entorno de diseño y que gracias a la interfície proporcionada por *Altera* (*DSPBuilder*) que es capaz de generar código VHDL sintetizable. En concreto el proyecto se centrará en la gestión de un sistema capturador de imágenes de contadores del caudal de agua, y lo que se pretende es hacer la caracterización de los contadores. Este dispositivo capturador consta básicamente de un sensor de imagen y de una FPGA. Lo que se pretende con la caracterización es ajustar los diferentes parámetros del sistema

para hacer que la lectura sea lo más óptima posible para cada modelo de contador que existe en el mercado. Como parámetros a modificar tenemos la exposición del sensor, la ganancia de un color, el filtrado de la imagen, etc...

SUMMARY

This project consists in the development of digital Hardware structures that can be synthetist in an FPGA a designed with a high level environment like the *Simulink* of *Matlab*, all due to a toolbox called *DSPBuilder* from Altera. This project will be centered in the management of system that make captures of images water flow meters, and make the characterization of the flow meters. This device consists basically in an image sensor and a FPGA. What is wanted with the characterization is to fit the different parameters from the system to cause that the reading of the digits of the flow meter is most optimal possible for each model that exists in the market. As parameters to modify we have the exhibition of the sensor, the gain of a color, the filtrate of the image…

Aquest projecte va dedicat a tots aquells

que m'han suportat, m'han ajudat o recolzat,

a la meva família, als amics, als companys

de carrera i especialment a la Cristina

Els signants Eloi Ramon i Garcia, professor del Departament de Electrònica, i Jordi Carrabina i Bordoll, professor del Departament de Microelectrònica i Sistemes Electrònics, de la Universitat Autònoma de Barcelona

Certifiquen

Que el treball corresponent a la present memòria ha estat realitzat sota la nostra direcció per Andreu Marzal i Marquet. I perquè així consti ho signem

Eloi Ramon i Garcia

Jordi Carrabina i Bordoll

Bellaterra, a 10 de Setembre de 2007

AGRAÏMENTS

Al director i co-directors del projecte, Eloi Ramon i Garcia, Jordi Carrabina i Bordoll i Víctor Montilla i Gispert per l'ajut prestat durant la realització del projecte, i en general a tots els integrants de Cephis, per les seves ajudes puntuals o assídues.

Índex

Índex de figures

Objectius.

L'objectiu d'aquest projecte és realitzar un entorn de disseny i verificació que acceleri el procés de caracterització de dispositius d'adquisició d'imatges (comptadors de fluids que mostren les lectures en dígits), de les quals n'hem d'extreure un conjunt de dígits, quan aquests estan sotmesos a toleràncies mecàniques i òptiques. La plataforma per a realitzar aquest entorn es basa en una FPGA la qual ens permet llegir una imatge del sensor, que cal poder configurar per realitzar després un tractament de la imatge en l'entorn *MATLAB*, des del qual generarem les diferents configuracions del codi hardware necessàries.

Com ja s'ha esmentat l'entorn de programació serà el de *Simulink*, integrat dins de *Matlab*. *Matlab* és un entorn de simulació o plataforma virtual de verificació que no estava inicialment orientat per fer interfícies Hardware, tot i que donat l'increment del nivell d'abstracció del disseny electrònic (s'abstreuen bussos, rellotges, etc.) els fabricants de FPGA i altres de CAD per a microelectrònica, han anat afegint eines (toolboxes en la nomenclatura de *MATLAB*) que fan el nexe entre el *Simulink* i els dispositius FPGA. En el cas d'Altera, la toolbox s'anomena *DSPBuilder*, i que un cop instal·lada sobre l'estructura de *Simulink* amb una sèrie de llibreries pròpies a les quals trobarem components i subsistemes hardware predissenyats per Altera o altres proveïdors de components virtuals (IPs) que podrem utilitzar per implementar el nostre disseny i sintetitzar-lo sobre la FPGA. Per això, s'ha d'aprofundir sobre el funcionament d'aquesta eina que ens ofereix Altera a més de la utilització d'un mòdul específic que altera ha desenvolupat pel *DSPBuilder* que es el *HIL* (Hardware In the Loop), que ens permet utilitzar l'execució de la FPGA com un element més de la verificació (multinivell) gestionada dins de *Simulink*. Aquest hardware accelera el temps de simulació, i a la vegada permet accedir realment al Hardware en la simulació.

Tot això ens donarà com a objectiu final el poder aplicar aquest disseny a la caracterització automàtica dels diferents tipus de comptadors que trobem al mercat.

Capítol 1.
Introducció i anàlisi

1.1. Plantejament general del problema

Fa anys la manera de llegir els comptadors de cabal de fluids era la d'enviar a una persona a que s'apuntés manualment el valor que indicava el comptador (de fet encara es fa en algunes situacions, com per exemple els comptadors domèstics del gas). Més tard es va anar introduint l'automatització amb comptadors de cabal que no només eren mecànics sinó que també ens donaven alguna magnitud elèctrica per poder mesurar el cabal, ja sigui per mitja de polsos (aquest tipus de comptadors solen tenir les sortides lliures de potencial, i l'únic que fan es crear o un contacte obert o un de tancat), de sortides analògiques de corrent (de 4 a 20 mA), o bé per bussos (com el cas del comptador de la casa espanyola ContaZara), però aquests comptadors per si sols no enviaven les lectures dels comptadors a l'empresa pertinent per que aquestes poguessin cobrar l'import del servei, sinó que es precisen de "Dataloggers" (enregistradors de dades) que el que fan es anar emmagatzemant les dades i cada cert temps envien la informació de diverses maneres per exemple per la xarxa telefònica commutada (XTC), via GSM, radio, etc. A més a més aquests Dataloggers no solen ser gaire econòmics, i com ja s'ha comentat també faria falta un altre canvi en la infrastructura important, com és el fet de canviar els comptadors que són simplement mecànics per uns que a més tinguin la esmentada interfície elèctrica.

Sabent tot això es va plantejar, fa us anys, fer un sistema que fos capaç de capturar la imatge dels dígits dels comptadors i enviar-la a un concentrador de lectors de comptadors, per comunicació inalàmbrica, i després descarregar aquestes fotografies

amb un PC o una PDA connectats a dispositiu capaç de rebre aquestes imatges. Tot això s'havia de realitzar amb un sistema que produït en sèrie no fos molt costós. Això és Mirakonta.

Mirakonta consta bàsicament d'una FPGA, un sensor d'imatge, uns leds per il·luminar, i una part de RF per l'enviament de la imatge, però tot això s'explicarà àmpliament més endavant.

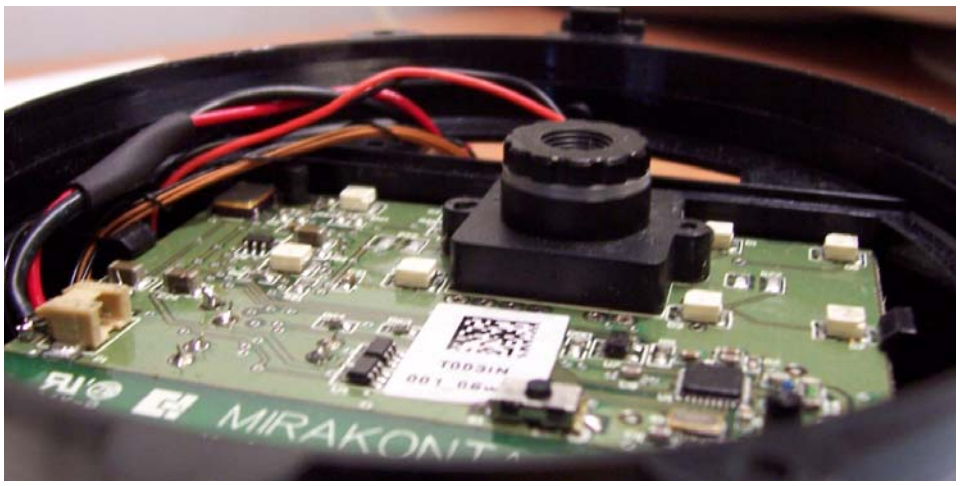


Figura 1. Fotografia de la Placa del sistema MiraKonta

1.2. Problemàtica específica de la caracterització

El principal problema és que s'ha de configurar el sistema en general per a cada model de comptador en particular, donat que cada comptador té unes característiques diferents als altres, com per exemple les dimensions dels dígits, les posicions d'aquests mateixos, la distància entre ells, etc.

Altra tema es el fet que també s'hauran de configurar els diversos registres del sensor per a que la imatge sigui òptima. Aquests paràmetres que ajustarem per a realitzar la caracterització, són:

- Guany general (de tots els colors).
- Guany del color vermell

- Saturació
- Brillantor
- Exposició

Tampoc ens es pot oblidar de configurar la lluminositat dels leds que tenim fins a tres ajustos diferents per led, a més a més de poder encendre i apagar els leds al nostre gust, és a dir, poder il·luminar uns o altres leds per així evitar reflexos indesitjats, com els de la següent imatge

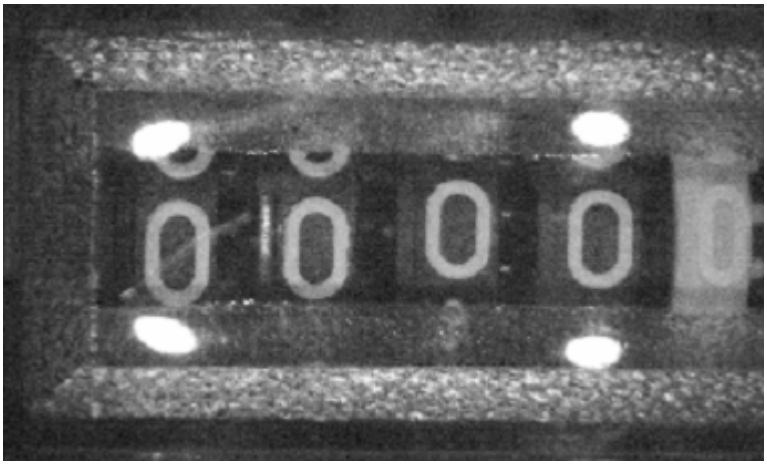


Figura 2. Imatge d'una captura dels dígits d'un cabalímetre

En conclusió veiem que per adequar el sistema MiraKonta a cada comptador hi ha uns quants paràmetres per poder modificar-los fins a obtenir els resultats desitjats, o en altres paraules, que la imatge resultant de la captura tingui la qualitat esperada.

1.3. Proposta de treball

Un cop vista la problemàtica que pot haver-hi en la caracterització dels diferents comptadors de cabal, el que hem de fer és un sistema que es puguin variar tots aquests paràmetres de manera ràpida, comprovar els resultats i emmagatzemar les imatges al PC, per si després es vol fer algun tipus de tractament. Per això va néixer la idea de fer tot des de *Matlab*/*Simulink*, donat que existeix una eina que s'executa sota la part més

gràfica de l'entorn, el *DSPBuilder* d'Altera que ens permetrà des de l'entorn de *Simulink* crear o importar codi Hardware. El principal avantatge de fer servir aquest entorn és que si un cop tenim la imatge emmagatzemada, volem fer un filtrat o comparar dos filtres, fer una detecció del contorn dels dígits de la imatge o qualsevol altra tractament de la imatge, amb el *Matlab*/*Simulink* és una feina relativament més assequible que si es fes escrivint el codi nosaltres mateixos, i a més a més *Matlab* també disposa d'una altra "toolbox" per exportar qualsevol de les seves funcions a codi HDL, és l'anomenat "HDL Coder", amb la qual cosa es pot preveure que estem dins un entorn molt potent.

Capítol 2.

Entorn de Desenvolupament

2.1. Plataforma HW

La plataforma HW del sistema és com s'ha dit abans el sistema Mirakonta, i les parts més importants del sistema són:

- Un sensor d'imatge amb la seva lent i porta lent, encarregat de realitzar la captura de la imatge.
- Una FPGA, que gestiona el sensor, tant la seva configuració, com la realització de la captura de la imatge, com el control del leds, el control del mòdul de RF.
- Un mòdul RF, que consta d'un µControlador de Nordic que incorpora ja el Hardware necessari per la comunicació RF, a excepció de l'antena

Per fer-ho més entenedor com és la plataforma a continuació es posen unes fotografies del sistema Mirakonta:

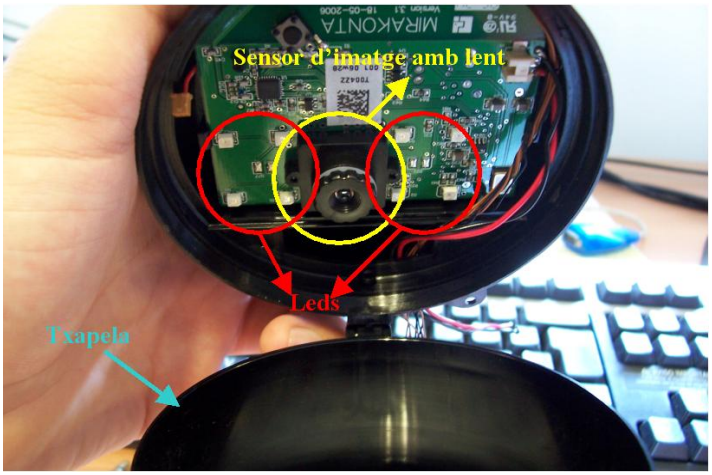


Figura 3.Imatge de la PCB muntat sobre la "txapela"

Figura 4.Imatge de la "txapela" del sistema Mirakonta muntat a un cabalímetre

Figura 5.Imatge del cabalímetre i la "txapela" de MiraKonta

- **El sensor d'imatge:** és un sensor de la casa OMNIVISION model OVT7640 i que té la següent distribució interna:

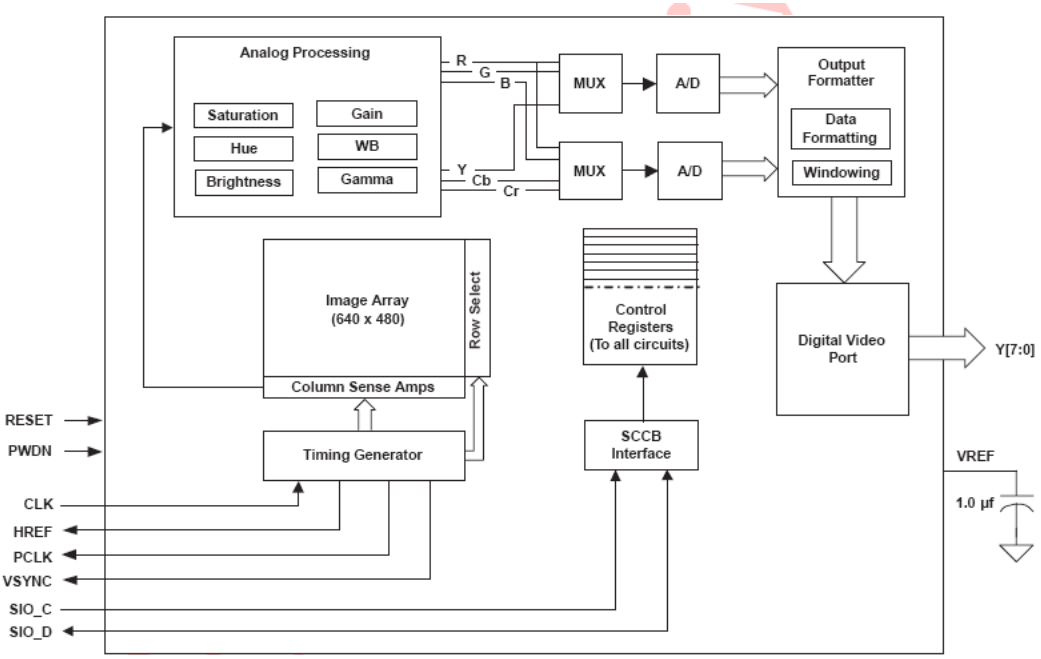


Figura 6.Diagrama de blocs del sensor d'imatge OVT7640

Observant aquest gràfic es pot extreure les següents conclusions sobre el sensor utilitzat:

- Té una resolució de 640 x 480 píxels
- Cada píxel és de 8 bits
- L'estructura de dades de la imatge és el d'una matriu i el que el sensor fa es anar traient les dades primer per columnes. Cada cop que acaba una fila passa a la següent fins arribar a la última, que llavors, sinó es desconnecta el sensor (per mitjà del senyal PWDN), torna a capturar una altra imatge i així cíclicament.

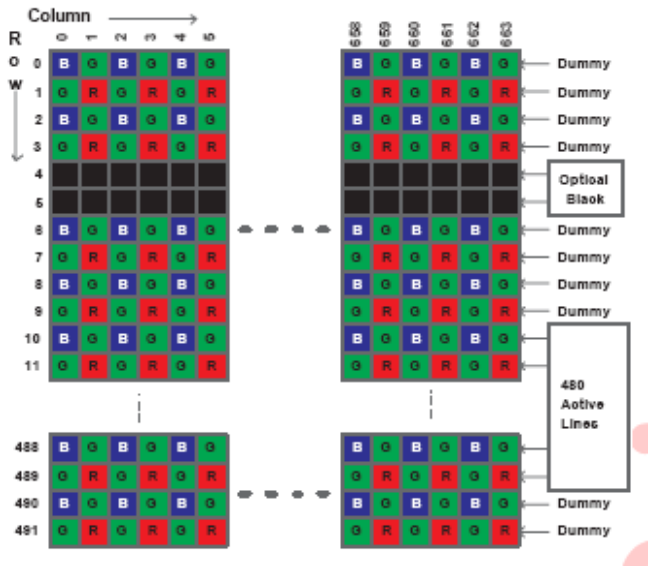


Figura 7.Distribució del píxels del sensor d'imatge en Bayer Pattern

- Disposa d'una sèrie de senyals de control, com son:
  - ***HREF*:** Ens fa d'enable de les dades per saber en quin moment les dades que treu el sensor son les que toquen llegir, s'ha de dir que aquest senyal és actiu a alta i que roman activat al llarg de totes les columnes de cada fila, després es desactiva fins que es passa a la següent fila i així successivament.

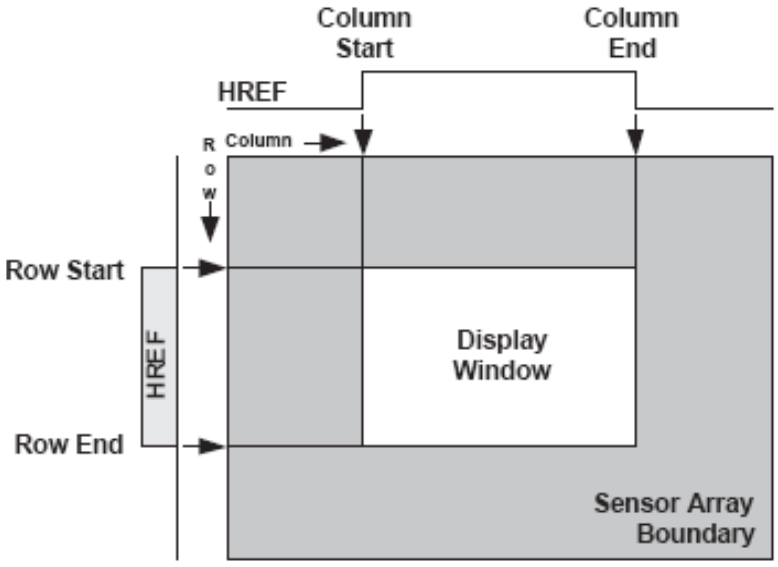


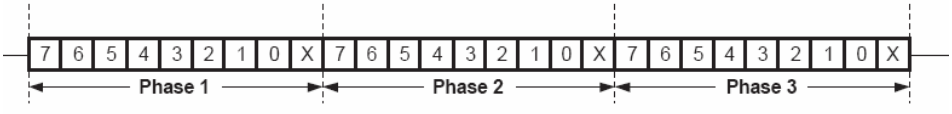
Figura 8.Diagrama de funcionament del senyal d'enable de les dades "*HREF*"

- RGB full resolution raw data single-line format (total 492 HREFs)
  - First HREF Y channel output B_{11} G_{12} B_{13} G_{14} ...
  - Second HREF Y channel output G_{21} R_{22} G_{23} R_{24} ...
  - Third HREF Y channel output B_{31} G_{32} B_{33} G_{34} ...
  - PCLK rising edge latches data bus.

Figura 9.Seqüència de les dades de sortida del sensor en mode Raw Data

- **VSYNC:** és un senyal que normalment es troba desactivat i quan fa un pols indica que tot seguit arribarà una nova imatge
- **PCLK:** és el Píxel Clock, per tant marca la freqüència a la que surten les dades que provenen del sensor.
- **RESET:** en cas de que el sistema es trobi en una situació no desitjada tornarem el sistema a condicions inicials.
- **CLK:** és el senyal de rellotge que s'ha d'entrar per a que el sensor pugui funcionar i per tant poder generar els senyals de sincronisme
- **SIO_D i SIO_C:** són senyals per regular els diferents registres del sensor, són com el protocol de comunicació I2C, però propi d'OMNIVISION, i es diu SCCB. Essent el senyal *SIO_D* les dades en mode sèrie i el *SIO_C* el rellotge. Si es mira el datasheet del mòdul SCCB d'Omnivision es veu que la manera de configurar els registres del sensor ve descrit per la següent imatge:

Transmission Phases



Phase 1: ID Address
Phase 2: Sub-address / Read Data
Phase 3: Write Data

Figura 10.Diagrama de les fases de la transmissió SCCB

Segons aquest diagrama es veu que primer de tot s'ha d'escriure l'adreça del dispositiu, és a dir, del sensor d'imatge, que pel cas del OVT7640 és la 42 per escriure i la 43, per llegir. Després s'ha d'escriure la direcció del registre a modificar/llegir, i per últim si s'escau el valor a escriure al registre corresponent.

- **PWDN:** és un senyal per encendre/apagar el sensor, per això es diu PowerDown, i si aquest senyal roman activat (a alta) el sensor està en mode Power Down i si està a baixa està encès.

- **La FPGA:** és de la casa Altera i més concretament el model EP2C5T144C8, és a dir es una Cyclone II model C5 amb encapsulat de 144 pins, només 89 accessibles per a l'usuari, ja que la resta estan destinades a alimentacions i masses. Disposa de:
  - 119.808 bits de memòria
  - 26 Multiplicadors de 9 bits
  - 2 PLL
  - 4.608 elements lògics, els quals estan formats per blocs combinacionals i registres i la família Cyclone II té la característica que es poden utilitzar la part combinacional i la seqüencial per separat. També té la particularitat de que cada LE (Lògic Element que és la suma de la part combinacional més el registre) pot ser configurat com un biestable tipus D, T, JK o SR. A més a més cada registre té les següents entrades:
    - Dades
    - Rellotge
    - Enable del rellotge
    - Reset

Per que es vegi millor com està composat cada LE tot seguit es mostra una imatge del seu esquemàtic:

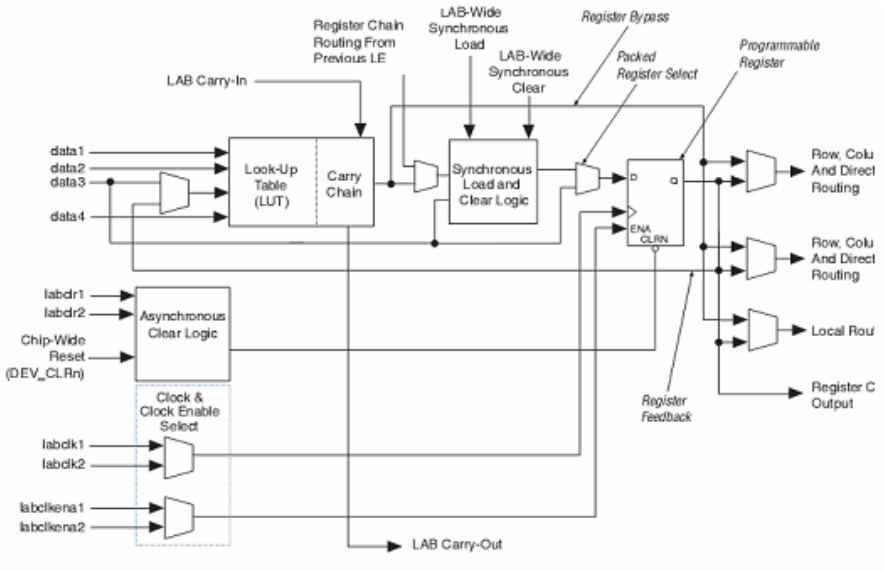


Figura 11.Diagrama de blocs de un LE (Logic Element)

- **<u>Els Leds:</u>** són els encarregats de la il·luminació en el moment de la captura de la imatge. Són 8 leds i mitjançant dues sortides de la FPGA per led, son dues donat que tenen diferents resistències associades per modificar el corrent cap el led, per tant es pot tenir fins a quatre estats diferents, és a dir, tenim l'estat d'apagat, el d'encés però amb baixa lluminositat, amb lluminositat alta i amb lluminositat molt alta. Aquests leds són de color vermell, i són d'aquest color bàsicament per que el sensor d'imatge que utilitzem, l'OMNIVISION OVT7640 té una gran resposta a la llum de color vermell, com es pot veure en el següent gràfic:

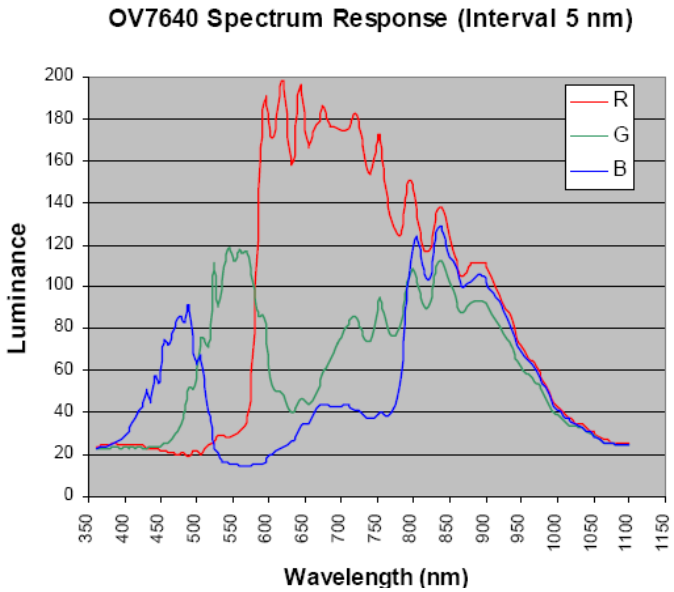


Figura 12.Gràfic de la resposta a les diferents components de color de la llum

Un altre motiu per el qual són de color vermell és degut a que la alimentació dels leds és de 2.5V aproximadament, i no com en el cas dels blancs per exemple que és d'aproximadament 3,2-3,4V, a més de ser de l'ordre de 10 a 50 cops més econòmics que els blancs. El principal avantatge d'utilitzar els de color blanc és que el blanc està composat per tots els colors i per tant li arribaria llum a tots els píxels.

- **El mòdul de radio freqüència:** està composat per un µControlador de la casa Nordic model nRF9E5, i com aquesta part no l'utilitzem ni programem res només es veurà a gran trets les característiques principals. No es fa servir donat que per aquest projecte es pretén que la imatge ens vingui pel mateix port pel que es configura la FPGA, és a dir, el JTAG, i per tant ens estalviem tota la part de comunicació inalàmbrica. Aquest microcontrolador també té la funció de mantenir tant la FPGA com el sensor d'imatge en mode standby, mentre el sistema MiraKonta no detecti una petició de captura d'imatge del comptador. És clar llavors que gràcies a aquest procés el sistema en global té un millora en l'aspecte energètic, doncs recordem que tot el sistema és alimentat per una bateria no recarregable de Liti, i per tant s'aconsegueix una major durada d'aquesta.

  A grans trets mirant el datasheet es pot dir que es tracta d'un microcontrolador de baix consum que consta d'un processador 8051 amb una sèrie de perifèrics com són un ADC de 10 bits de resolució i de quatre canals, i un transceiver de la casa Nordic nRF905 que realitza una modulació GFSK (Gaussian Frequency Shift Keying), codificació tipus Manchester i amb un data-rate100kbps. Es pot fixar-nos també que pot emetre en 3 bandes diferents com són la de 433 MHz, 868 MHz o 915 MHz.

  Vistes les diferents parts del sistema MiraKonta, ara només ens resta descriure el seu funcionament. El sistema bàsicament comença fent una captura d'imatge, però aquesta imatge es tractada abans del seu enviament per radiofreqüència a un concentrador de lectors de comptadors. Aquest tractament consta de:

- Primer de tot de fer un downsampling eliminant tots el píxels que no siguin vermells, donat que com ja s'ha vist el sensor treu Bayer Pattern, i a la vegada observem un guany elevat al color vermell per part del sensor. Això i el fet que els leds que il·luminen també són de color vermell, fan que ens decantem per aquest color a l'hora d'escollir-ne un.
- A continuació després de reduir la resolució de la imatge de 640 x 480 a 320 x 240 el que es fa és fixar un àrea d'interès (AOI), que serà la corresponent a la dels dígits, és a dir que s'agafen només els espai que ocupin els dígits, s'eliminaran els espais que hi ha entre els dígits donant com a resultat una imatge més petita i més compacta. Aquesta àrea actualment en el sistema MiraKonta és de 128 x 64, i com que hi ha 5 dígits ens queda una resolució de 26 x 64 píxels per 3 dígits i per als altres dos dígits 25 x 64, que a la pràctica no hi ha diferencies substancials en quan a resultats. L'àrea que s'agafa queda representada en el següent gràfic pels rectangles grocs:

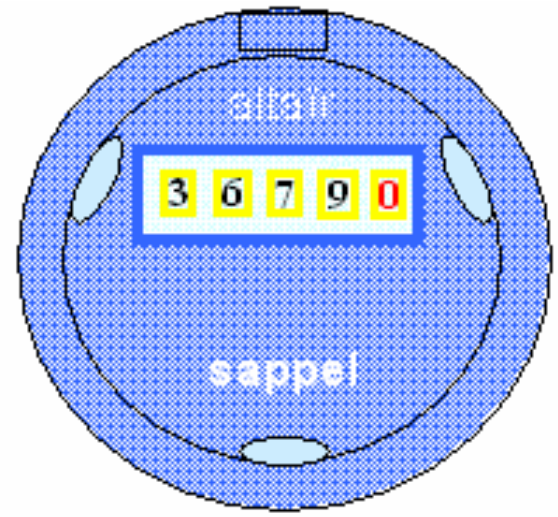


Figura 13.Gràfic dels dígits d'un cabalímetre amb l'àrea d'interès

- Després d'agafar l'àrea d'interès el que es fa és una binarització dels píxels, és a dir el que es fa és passar de 8 bits per píxel a només 1 bit.
- Per finalitzar el que es fa amb la imatge és emmagatzemar-la en una memòria Dual Port RAM, i s'envia al concentrador.

2.2. Entorn CAD

En l'entorn CAD es veuran els diferents entorns de programació per poder dur a terme aquest projecte, donat que per el desenvolupament s'ha hagut de passar per diverses etapes i per cadascuna d'elles s'ha utilitzat un entorn diferent per la realització dels diferents objectius. Primer de tot es veurà l'entorn de simulació *ModelSim*, a continuació un altre entorn de simulació com és el *Matlab*, i aquest entorn també s'utiñlitzarà per a la síntesi lògica gràcies a la toolbox *DSPBuilder* i a l'entorn de *QuartusII*.

2.2.1. *ModelSim*

És l'entorn que s'ha utilitzat per fer la part de simulació, que en el nostre cas s'ha utilitzat per simular codi de descripció de Hardware, VHDL, és a dir, es realitza una simulació funcional i comportamental. S'ha de dir que en el cas de *ModelSim* la llista d'estímuls s'han de descriure en un fitxer de VHDL. La simulació funcional i comportamental**,** es realitza generalment abans del *Place&Route* i testeja el correcte funcionament lògic del disseny. La simulació és independent de qualsevol arquitectura empleada per Altera o qualsevol altre fabricant. Un cop verificat el funcionament, el següent pas serà sintetitzar el disseny per obtenir un disseny a nivell de portes lògiques i utilitzar el *QuartusII* per a un correcte *Place&Route*.

En el nostre cas el primer que es va fer va ser crear un emulador del sensor d'imatge de la placa del lector de comptadors Mirakonta, el OVT7640. Per fer aquest emulador s'han de tenir clares quines són les característiques principals d'aquest sensor, per poder emular-lo, cosa que ja s'ha esmentat en l'apartat en el qual es tractava la interfície gràfica, però tot i així és recordaran els senyals a simular:

- *VSYNC*
- *HREF*
- *PCLK*
- *Y[7..0]*
- *RESET*

Un cop ja han quedat clar quins són els senyals a implementar i quines són les seves característiques, només resta fer-lo i simular-lo amb el *ModelSim*. Per entendré més com és *ModelSim* es posaran a continuació algunes imatges del programa:

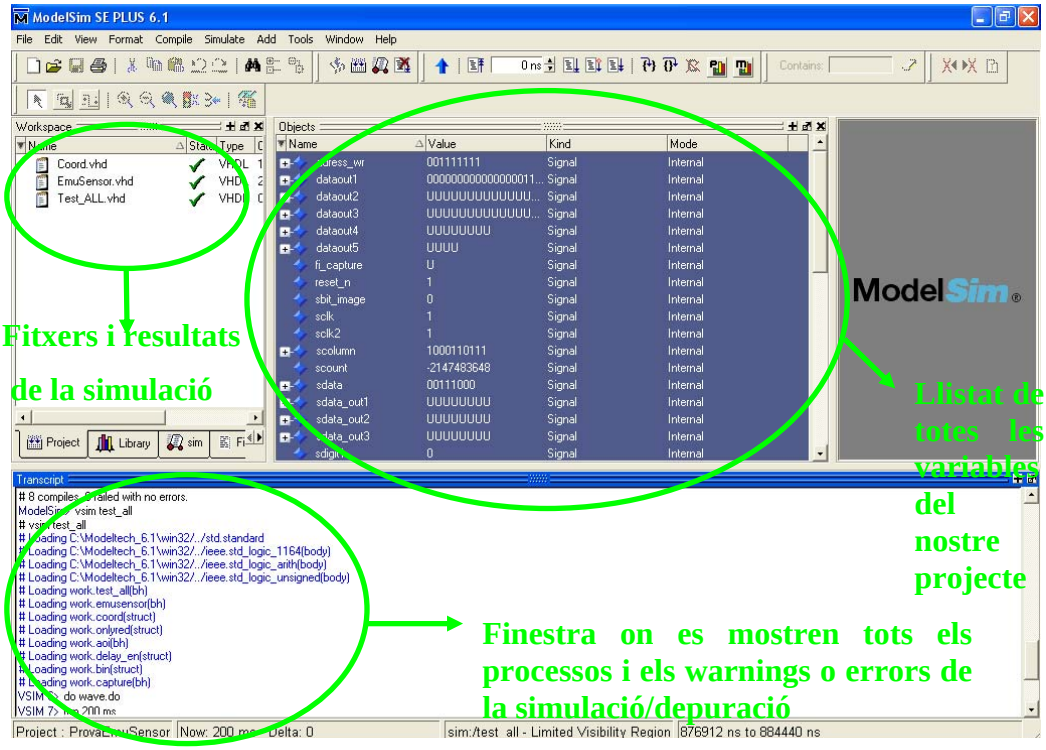


Figura 14.Aspecte general del programa *ModelSim*

El visualitzador d'ones permetrà veure els resultats simulats de totes les interfícies de sortida del disseny realitzat:

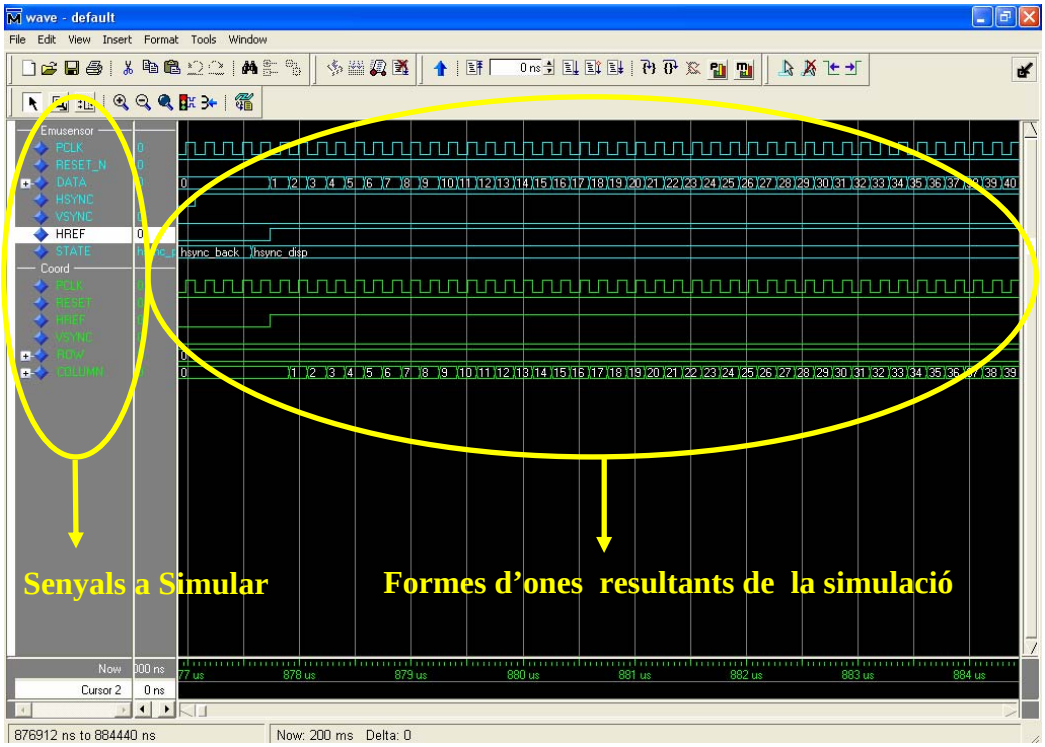


Figura 15.Aspecte del visualitzador de formes d'ones de *ModelSim*

2.2.2. *MATLAB*

Com més que probablement ja es conegui l'entorn de *Matlab*, el que es fa és una breu descripció. El primer que s'ha de dir és que es tracta d'una eina matemàtica de càlcul numèric, i amb això es pot fer des del càlcul d'equacions fins a aplicar filtres a una imatge. Cal dir que *Matlab* disposa d'una eina més gràfica que és el *Simulink*, en la qual la programació es fa mitjançant blocs, que cadascun fa una cosa, i no es fa mitjançant un fitxer escrit com es fa amb el cas d'utilitzar *Matlab* sense *Simulink*. Dins de l'aplicatiu *Simulink*, s'ha d'instal·lar una tollbox anomenada *DSPBuilder* de la casa Altera, però això s'explicarà amb més detall a continuació en l'apartat corresponent.

2.2.3. *DSPBuilder*

El *DSPBuilder* d'Altera és la eina (toolbox en el llenguatge de *Matlab*) encarregada de fer d'interfície entre el software *QuartusII* d'Altera i d'altres eines d'alt nivell des d'on es realitza el disseny del sistema d'aquest projecte mitjançant blocs funcionals: el *Matlab* i el *Simulink*. Es tracta d'un conjunt d'eines que permeten adjuntar blocs de funcionalitats diferents amb l'objectiu de formar un sistema més complex. Aquest fet

permet reduir el temps i les fases de disseny d'un DSP (Digital Signal Processor) en una FPGA d'Altera.

En el projecte s'utilitza la interfície proporcionada pel *Matlab-Simulink* per tal de realitzar el disseny del sistema. El *DSPBuilder* després de la seva instal·lació incorpora a l'entorn *Simulink* una sèrie de llibreries que inclouen tot un conjunt de funcions matemàtiques juntament amb models de simulació, llestes per a ser empleades per tal de crear esquemes de sistemes més complexos. Un cop acabat el disseny, el *DSPBuilder* s'encarregarà de generar la descripció en VHDL de l'esquema dissenyat, així com els testbench pertinents. D'aquesta manera el *DSPBuilder* aconsegueix combinar les capacitats dels algoritmes de desenvolupament, simulació i verificació del *Matlab* i del disseny de sistemes del *Simulink* amb els fluxos de disseny en VHDL i Verilog que inclou el *QuartusII*. A més a més és possible combinar les funcions existents del *Matlab* i els blocs de *Simulink* amb les noves llibreries i blocs esmentats del *DSPBuilder* i amb les IP Cores d'Altera (IP Megacore functions) per tal d'aconseguir sistemes més complexes de forma més senzilla.

La peça clau del *DSPBuilder* en l'entorn del *Simulink* és el bloc SignalCompiler, que s'encarrega de la traducció de l'estructura en forma de blocs del *Simulink* a blocs de funcions VHDL. Així generarà els codis VHDL pertinents de cada bloc de l'entorn *Simulink*.

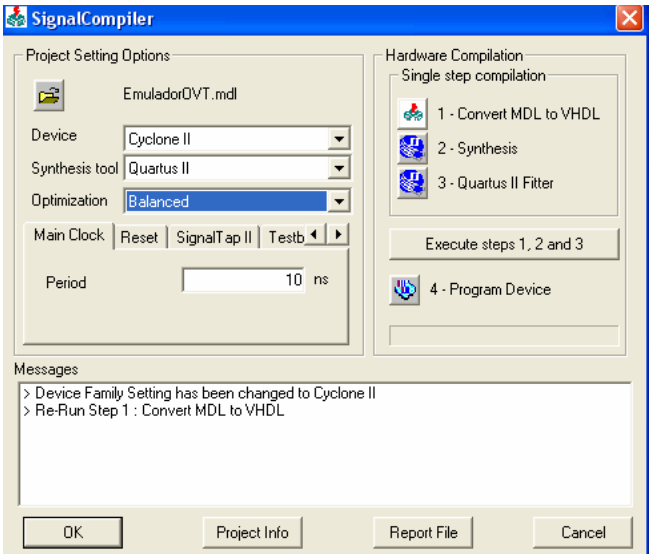


Figura 16.Captura de pantalla del SignalCompiler

- **Procés de flux de disseny:**

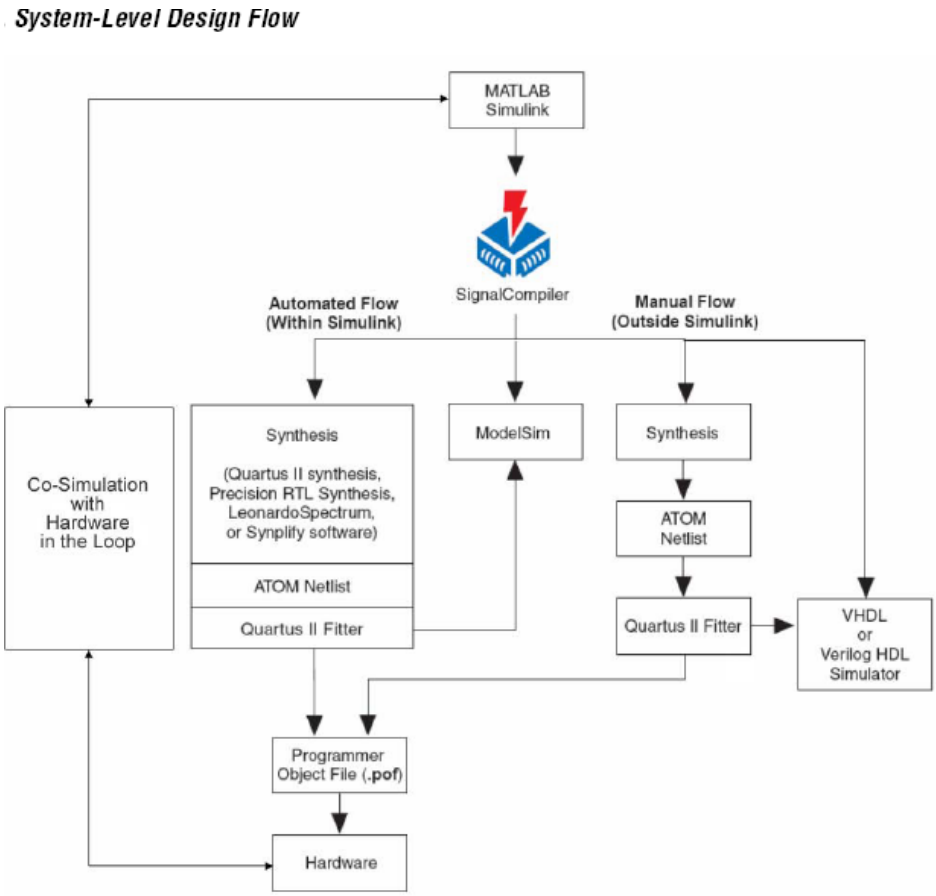


Figura 17.Diagrama de flux de la toolbox *DSPBuilder*

El flux de disseny típic del *DSPBuilder* és el que es descriu a continuació:

1. Crear el model amb *Simulink* mitjançant els blocs del *DSPBuilder* d'Altera incorporats a l'entorn *Simulink*. Crear les interconnexions necessàries mitjançant bussos, indicar els bussos d'entrada i sortida i configurar les variables d'entorn tipus velocitats dels clocks, durada dels comptadors, etc...
2. Síntesi RTL. En la opció de flux de disseny manual, s'utilitzarà el bloc del signal compiler donat per Altera en l'entorn *Simulink* per generar a la sortida els fitxers VHDL o Verilog i també per generar scripts TCL. La funció del signal compiler serà la de llegir els fitxers del *Simulink* ***.mdl** que contenen el disseny i mapejar els blocs de *Simulink* corresponents al *DSPBuilder* d'Altera a la llibreria VHDL del *DSPBuilder* corresponent. Així s'obtindran les funcions en codi VHDL que seran sintetitzades i posteriorment simulades mitjançant els scripts TCL. Els scripts TCL són fitxers que permeten

configurar les simulacions de tal forma que siguin automàtiques i per tant no ens caldrà haver de configurar pràcticament res del disseny. Les simulacions es podran executar mitjançant softwares com per exemple el *ModelSim* que és el que utilitzarà aquest projecte. Els scripts TCL són una gran ajuda, ja que permeten obtenir de forma més senzilla i ràpida els resultats de la simulació del disseny. El signal compiler també té la capacitat de realitzar el place&route del disseny amb l'ajut del *QuartusII*. Cal dir també que encara que en aquest projecte sempre s'ha utilitzat el *QuartusII* per fer la síntesi, el signal compiler també ens permet escollir l'opció d'altres programes com ara el Precision RTL, el LeonardoSpectrum o el Synplify. A més a més també ens deixa triar el tipus d'optimització que volem, per velocitat, per àrea, un terme mig entre aquestes dues (balancejat), etc..

3. Simulació amb el *ModelSim*. Es tracta d'una simulació funcional que verifica la funcionalitat del disseny RTL. La simulació també podria ser realitzada mitjançant el *QuartusII*.

4. Descàrrega a la FPGA.

- **Simulació dirigida per events VS simulació basada en etapes:**

El *DSPBuilder* utilitza les característiques del *Simulink* per simular el comportament de components hardware que seran posteriorment descarregats a FPGA com ja s'ha descrit. Hi han varis diferencies entre la simulació dirigida per events (típica de simulacions HDL o Verilog) i simulacions per etapes (típiques del *Simulink*). El *DSPBuilder* omple aquest buit establint un conjunt de semàntiques temporals per a traduir entre el *Simulink* i l'entorn HDL.

  - **Model de simulació del *Simulink*.** El mode de temps recomanat per a ser utilitzar entre *Simulink* i el *DSPBuilder* és el fixed-step simulation, proporcionat en la configuració del *Simulink*. Al principi de cada etapa (pas) el *Simulink* li proporciona a cada bloc les entrades que fins al moment són conegudes. Un cop es tenen les entrades es realitzen les

funcions pertinents que hagi de realitzar cada bloc i es propaguen de nou les sortides per a que a la següent etapa serveixin d'entrada.

- **Models de simulació HDL.** Aquest tipus de simulacions són realitzades mitjançant la senyal d'un clock i l'habilitat per a detectar estímuls d'entrada. Els scripts de testbench generats pel SignalCompiler, serveixen com a senyal d'entrada per al simulador HDL amb l'objectiu de mantenir la correlació entre el disseny en HDL i el *Simulink*. Cadascuna de les sortides s'actualitza en els flancs positius de rellotge

- **Altres característiques i software necessaris:**

De les vàries característiques del *DSPBuilder* ens caldria destacar al menys dues. La primera és el fet de que ofereix possibilitat de fer l'anàlisi de nodes interns del disseny, mitjançant el que s'anomena SignalTap II Logic Analysis. Es tracta de col·locar uns blocs de *Simulink* per a tal efecte dins del node que interessi i amb posterioritat amb l'objectiu de corregir errors de disseny aquests nodes podran ser vistos des de l'exterior a través de les I/O que presenti la FPGA on s'implementarà el sistema. La segona característica destacable és el que s'anomena *HDL Import*, que no és res més que la possibilitat d'afegir un disseny propi realitzat amb codi VHDL. Per acabar amb el *DSPBuilder* remarcar que donada la quantitat de versions dels programes empleats pel disseny amb *DSPBuilder* descrit, es fa necessari disposar al menys de les següents versions per a que tots els processos funcionin correctament, inclòs les IP Megacores:

- PC amb sistema operatiu Windows 2000/XP
- *Matlab* versió 7.0 (R14) o superior
- *Simulink* versió 6.0(R14) o superior
- *QuartusII* versió 6.1 o superior.
- *DSPBuilder* 6.1

- **Llibreries addicionals: *DSPBuilder* + MegaCore IP's**

La instal·lació del DSP-Builder inclou un conjunt de llibreries que s'incorporen a l'entorn *Simulink* del *Matlab* i que serviran per a crear amb posterioritat els fitxers en VHDL, seguint el procés descrit anteriorment.

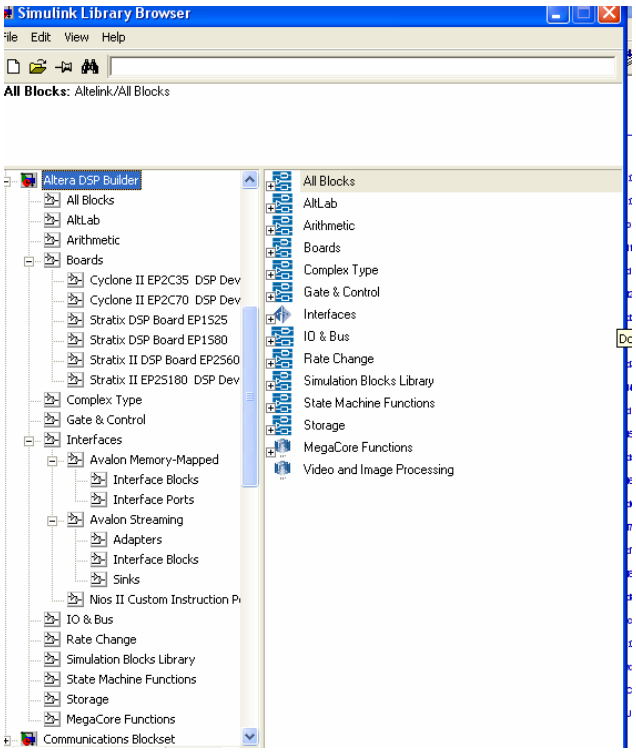
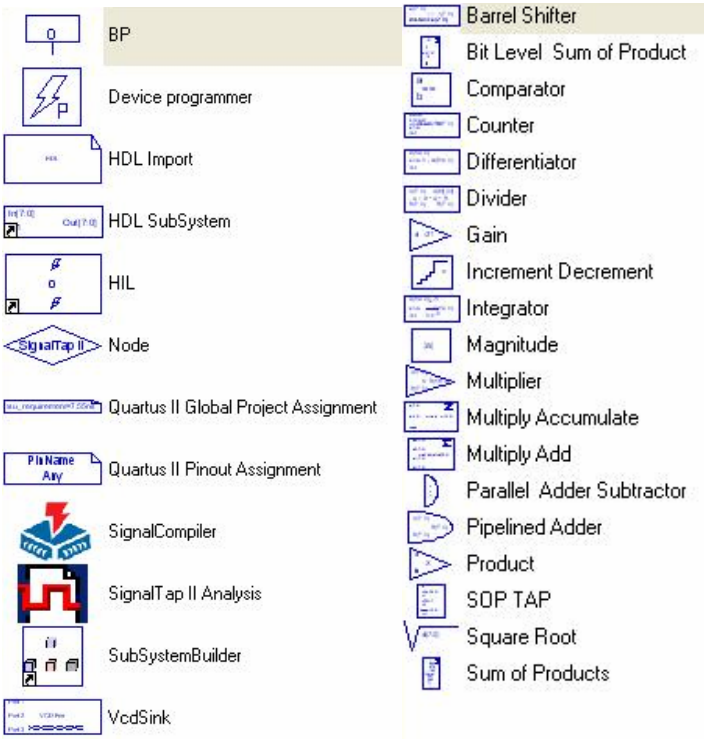


Figura 18.Aspecte de la toolbox *DSPBuilder* de *Simulink*

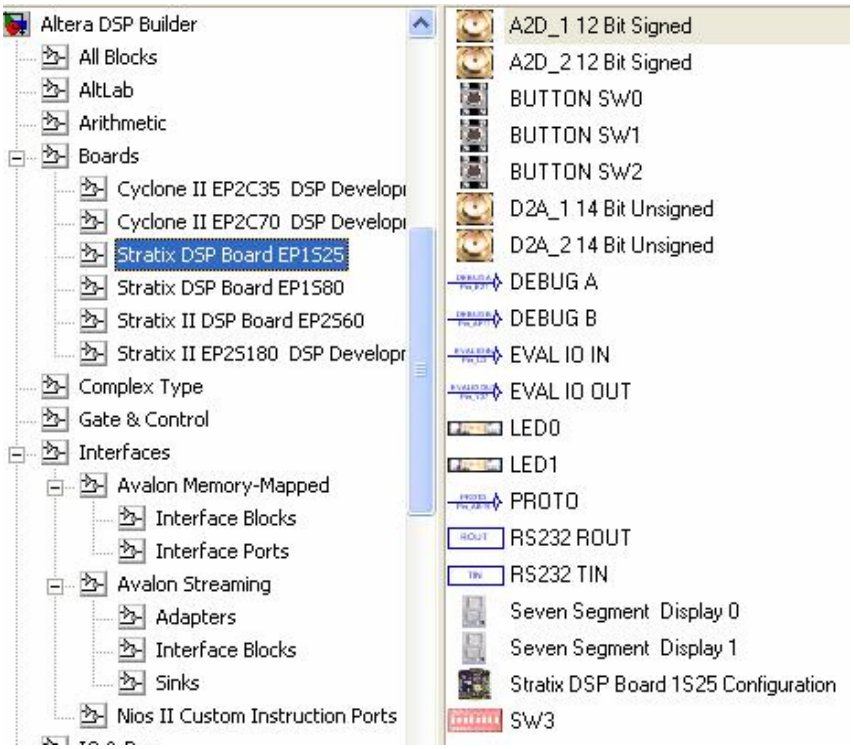
Les llibreries estan dividides en categories segons la seva funcionalitat. Així per exemple la llibreria Altlab conté els elements que descriuen quins són els bussos d'entrada o sortida del disseny. També conté el tan important SignalCompiler i el SignalTap II Logic Analysis. De les altres llibreries es pot parlar de la *DSPBuilder* Board, que conté tots els blocs necessaris per a la correcta configuració dels pins de la placa de desenvolupament, clocks, I/O, etc... Conté configuracions per a tres plaques distintes, entre elles la EP1S25. D'entre les altres llibreries restants la més especial de totes és la de les Megacore Functions que com es veurà a més endavant són blocs específics parametritzables per a la realització de funcions determinades.

Les llibreries de funcions predefinides pel *Simulink* que incorpora les versió 6.1 del *DSPBuilder* són les que es mostren a continuació:

ALTLAB **ARITHMETIC**



BOARDS-STRATIX DSP BOARD EP1S25



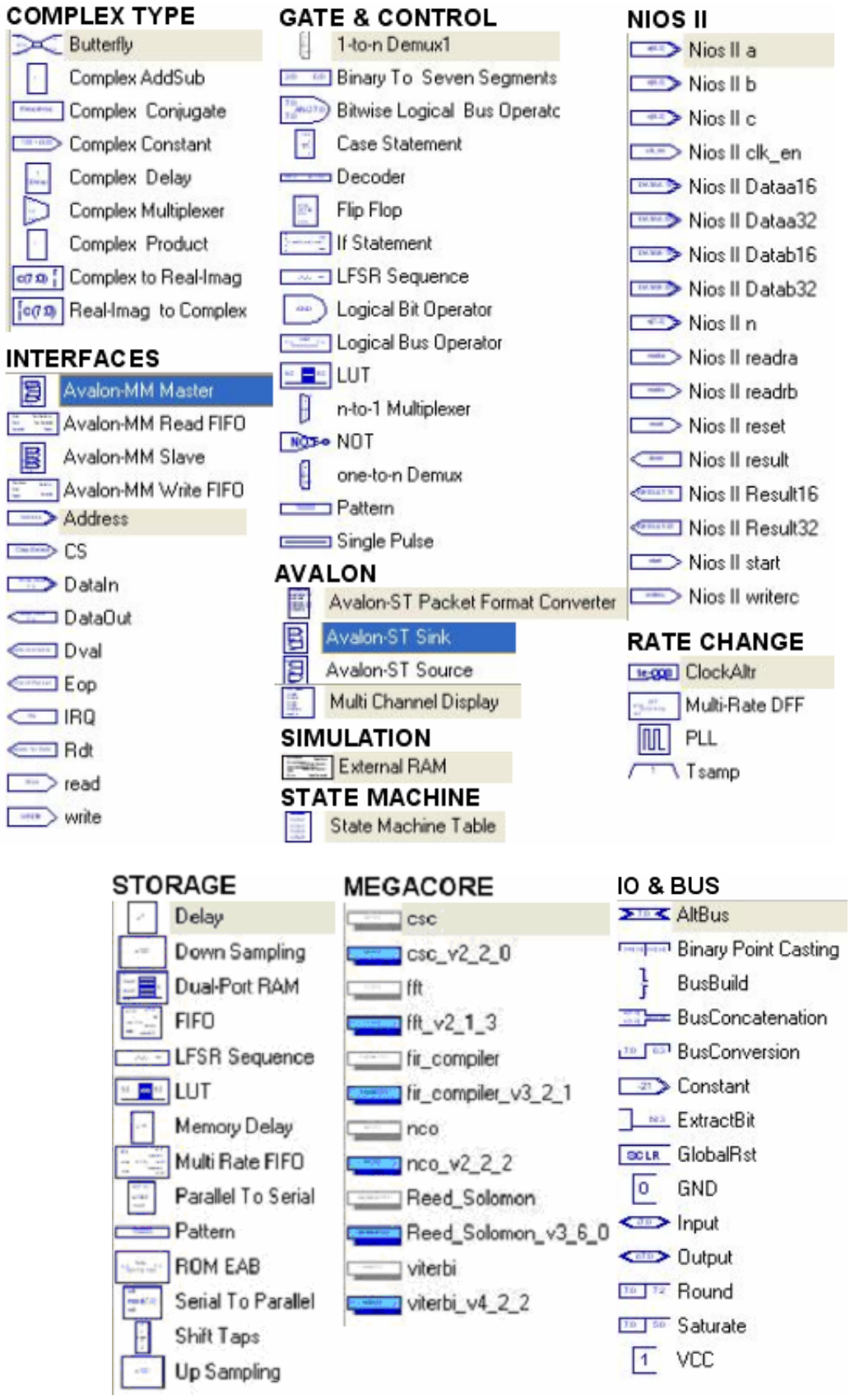


Figura 19. Resum de totes les llibreries del DSPBuilder 6.1 pel Simulink

Les IP Cores són funcions genèriques que ofereix la companyia de FPGAs amb l'objectiu de facilitar la feina al dissenyador de sistemes. Altera ofereix una llarga selecció de funcions IP (Intellectual Property Functions) llestes per a ser descarregades directament des de la seva pàgina web: http://www.altera.com/products/ip/ipmindex.html. Els dissenyadors poden incorporar aquests blocs parametritzables fàcilment al seu disseny, reduint temps de testeig i optimitzant recursos, ja que les IP Cores venen completament acabades i verificades. Dins dels *Simulink* aquestes funcions venen anomenades com a Altera MegaCore Functions, es representen mitjançant blocs i estan localitzades dins una llibreria en concret. A través del MegaWizard Plug-in Manager que ve amb el *QuartusII* es crearan, modificaran i parametritzaran totes les Megacore Functions. El fet de poder parametritzar les diverses funcions Megacore permet implementar-les en molts sistemes digitals i les fa compatibles per a una gran varietat d'aplicacions. A continuació es poden veure els diversos camps pels que Altera ofereix IP Cores:

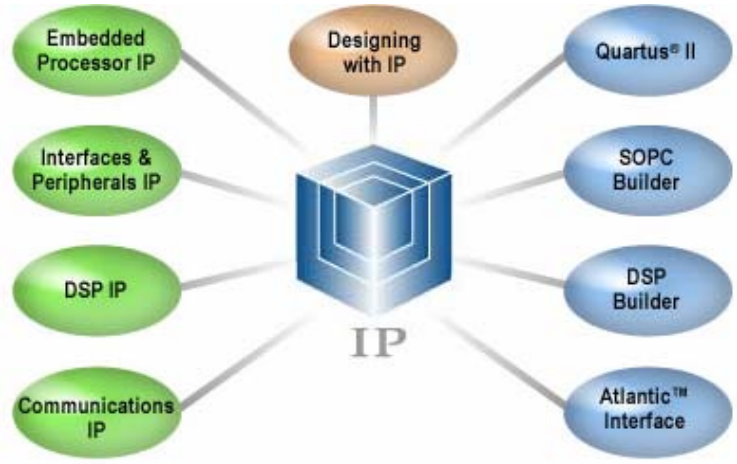


Figura 20.Aplicacions de les IP Cores d'Altera

2.2.4. *QuartusII*

És l'entorn de treball propi de la casa Altera per a les seves FPGAs i CPLDs. És una eina que fa tots els processos necessaris per que un codi Hardware pugui ser sintetitzat i descarregat sobre una de les seves FPGAs. Des de la compilació i els seus resultats, com ara l'àrea de la FPGA utilitzada, blocs lògics utilitzats, warnings i errors, fins

l'assignament de les entrades i sortides del sistema a implementar amb els pins físics del dispositiu. Per que s'entengui millor ara es mostraran unes imatges per aclarir-ho:

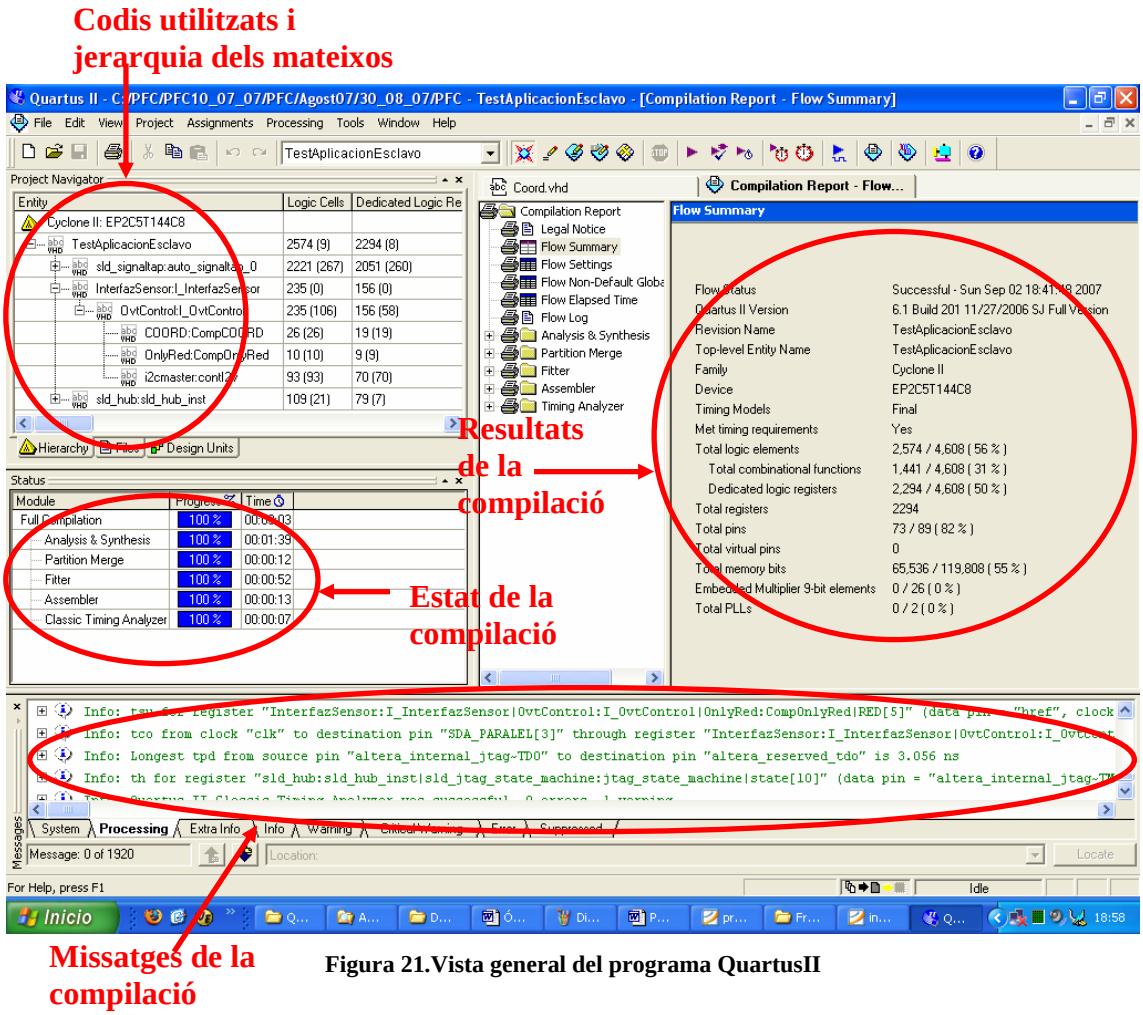
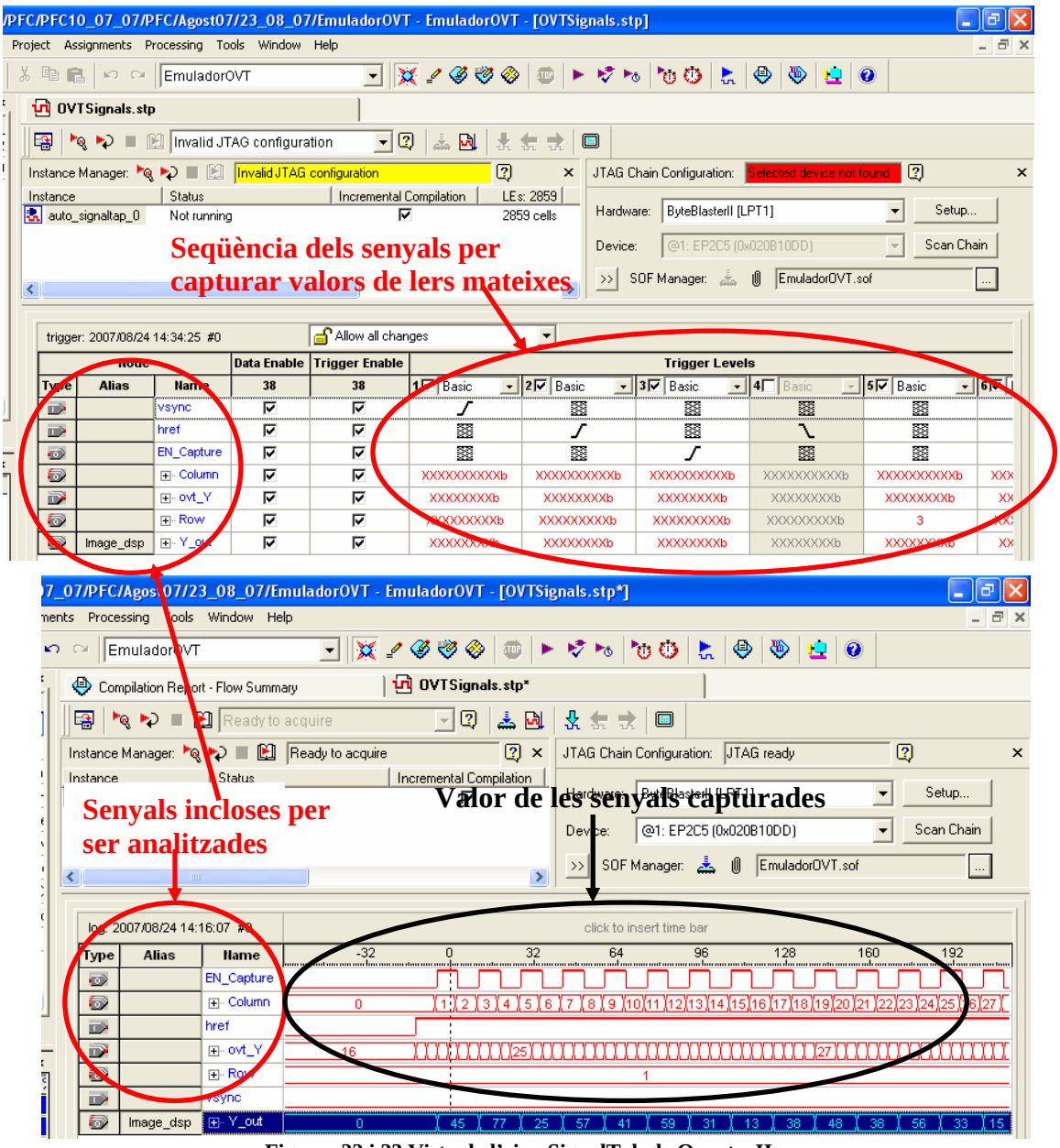


Figura 21.Vista general del programa QuartusII

Una de les eines més utilitzades del *QuartusII* és el SignalTab, que no és altra cosa que un analitzador lògic que Altera ha implementat per poder incorporar a la FPGA, que el que fa és utilitzar espai de memòria de la FPGA per emmagatzemar la seqüència de les senyals a mostrar. El funcionament és ben senzill i és el següent:

- Primer s'han d'afegir els senyals a analitzar
- Després s'escolleixen diversos paràmetres com ara el rellotge de sistema per fer el mostreig, el tamany en bits que es vol capturar de cada senyal, si es vol que emmagatzemi el senyal a partir de que es doni la seqüència programada, que ho faci al final d'aquesta, al mig, etc..

- També es pot variar la llargada de la seqüència
- Un cop fet tot això s'ha de compilar el projecte amb el fitxer de SignalTab inclòs, i després es programa la seqüència, i s'executa.



Figures 22 i 23.Vista de l'eina SignalTab de QuartusII

Quan es diu codi Hardware el que es vol dir es que els fitxers que s'utilitzen per a donar com a resultat el Hardware desitjat poden ser del tipus text, és a dir HDL (VHDL o Verilog), o be del tipus gràfic, que no es altra cosa que un esquemàtic de portes

lògiques o blocs predefinits per el fabricant o l'usuari, ja siguin multiplexors, biestables, comptadors, etc.. Pel que fa a la síntesis inclou:

- Placement
- Routing
- Verification
- Programming functions

2.3. Metodologia de treball

L'objectiu principal de fer desenvolupament del codi hardware amb el *Matlab* és el fet que es tracta d'un entorn de prototipat ràpid amb els avantatges de poder utilitzar eines de més alt nivell com serien les de creació i tractament d'imatges. La idea es arribar a fer una plataforma que vagi realitzant fotografies i vagi modificant els valors dels diferents registres del sensor. D'aquesta manera es pot arribar a aconseguir un sistema de caracterització automàtic.

La metodologia de treball la es pot dividir en cinc grans fases:

- La primera fase és implementar un emulador del sensor d'imatge OVT7640 en codi VHDL, i per realitzar això es fa sota l'entorn *ModelSim*. S'ha de tenir present que bàsicament per emular aquest sensor el que s'ha de generar són els senyals pertinents, que recordem son:
    - Un pols de *VSYNC* quan es genera l'enviament de les dades que pertanyen a una nova imatge.
    - Un senyal que roman activa mentre les dades són valides, anomenada *HREF*. Cal fer que sigui activa al llarg de les 640 columnes i de les 480 línies o files (recordem que es pot fer una analogia entre la imatge i una matriu).
    - Un rellotge anomenat PixelClock (*PCLK*) que ens fa de sincronisme de la sortida de dades del sensor.
    - La sortida de dades de la imatge, que és de 8 bits.

- La segona fase és conèixer l'entorn del *DSPBuilder* ja que és un entorn en el qual no s'haa treballat mai en el nostre grup. El primer que es va fer va ser lògicament llegir-se la documentació que s'adjunta amb l'aplicació, que són el *DSPBuilder Reference Manual*, *DSPBuilder Release Note* i *DSPBuilder User Guide*. A continuació el que es va fer va ser començar a implementar alguns dels exemples que venen amb el *DSPBuilder* per a practicar amb les eines disponibles. Comentar que es va fer amb una placa de desenvolupament que es disposa a Cephis, la "Stratix EP1S25 DSP

Development Board" ja que el programa conté una llibreria específica d'aquest kit, en la que s'inclouen els perifèrics de la FPGA com ara els leds, els polsadors, el port RS-232, els convertidors analògics-digitals i digitals-analògics, els switchos, etc..

Figura 24.Components al DSPBuilder de la "Stratix EP1S25 DSP development board"

A continuació es mostra una imatge del kit de desenvolupament DSP Stratix EP1S25:

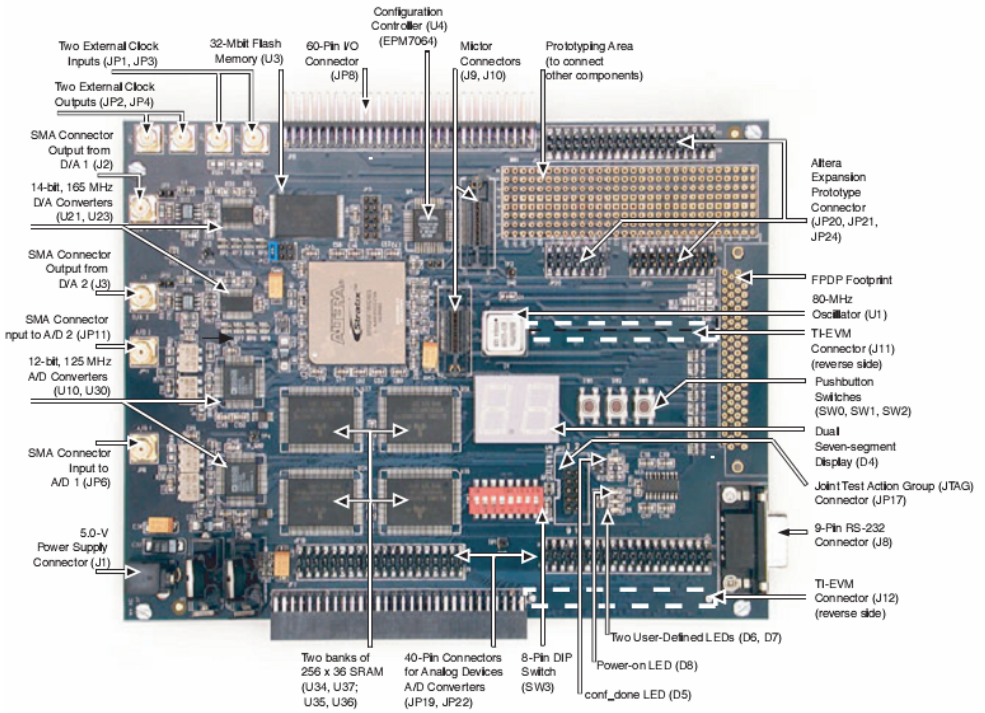
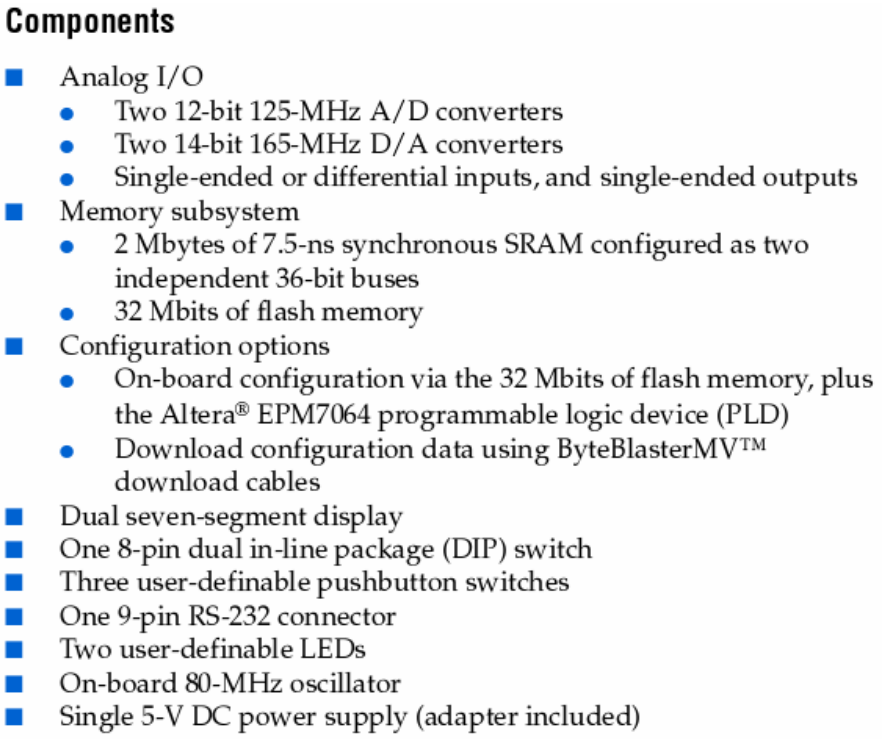


Figura 25.Vista general de la placa "Stratix EP1S25 DSP development board"

Però per que es puguin veure millor les característiques generals del kit, s'adjunta una captura del datasheet:



Components

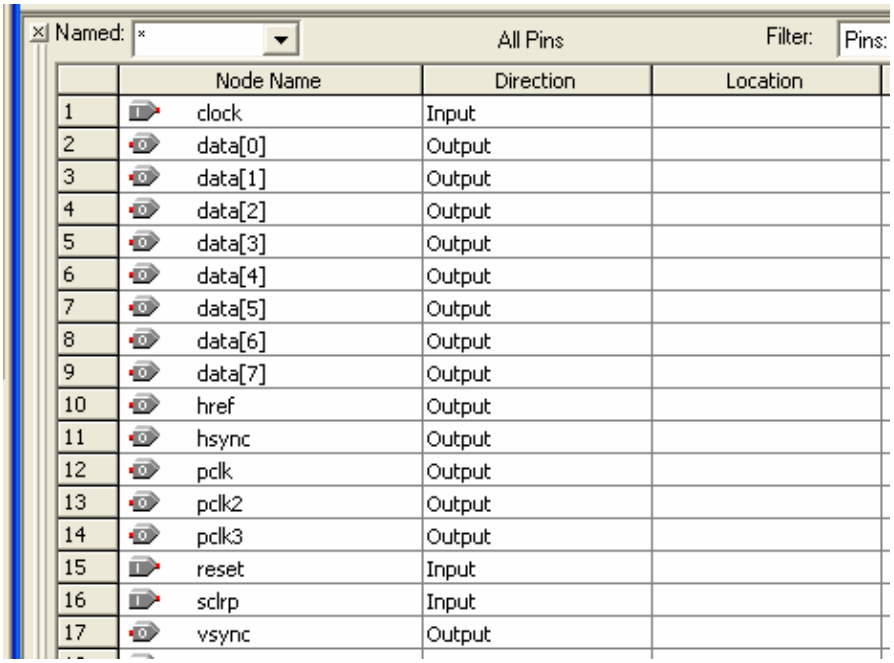
- Analog I/O
  - Two 12-bit 125-MHz A/D converters
  - Two 14-bit 165-MHz D/A converters
  - Single-ended or differential inputs, and single-ended outputs
- Memory subsystem
  - 2 Mbytes of 7.5-ns synchronous SRAM configured as two independent 36-bit buses
  - 32 Mbits of flash memory
- Configuration options
  - On-board configuration via the 32 Mbits of flash memory, plus the Altera® EPM7064 programmable logic device (PLD)
  - Download configuration data using ByteBlasterMV™ download cables
- Dual seven-segment display
- One 8-pin dual in-line package (DIP) switch
- Three user-definable pushbutton switches
- One 9-pin RS-232 connector
- Two user-definable LEDs
- On-board 80-MHz oscillator
- Single 5-V DC power supply (adapter included)

Figura 26.Llistat dels components de la "Stratix EP1S25 DSP development board"

Cal dir que el principal avantatge de tenir el kit de desenvolupament a les llibreries es que al afegir qualsevol dels perifèrics que hi ha per connectar a la FPGA no cal fer l'assignació de Pins, ja que ja va implícit al incloure el component, i com a conseqüència d'això ens estalviem aquesta fase de fer el Pinout o assignació de pins. També es necessari dir que no només es pot gestionar els perifèrics que es disposen en el kit, sinó que també es pot fer ús dels pins d'expansió d'entrada/sortida que disposa el kit, per poder connectar-hi hardware extern, mitjançant dues maneres:

- La primera seria fent ús d'un dels components de les llibreries d'aquest kit anomenat "EVAL IO IN/OUT", en el qual només s'ha de mirar el datasheet del kit i assignar a la sortida que volem dels diversos ports d'expansió.
- La segona és de caràcter general per a qualsevol FPGA de la casa Altera, i és utilitzar el "*QuartusII* Pinout Assignment", la qual cosa permet assignar qualsevol Pin de la FPGA, que no sigui d'us restringit.

Per tant veient això podríem dir que amb el *DSPBuilder* es pot implementar a qualsevol FPGA encara que aquesta no estigui en una placa de desenvolupament de les que es troben a les llibreries del programa, però res més lluny de la realitat, com a mínim en la versió 5.1 del *DSPBuilder* que s'utilitzava al principi, al fer l'assignació de la segona manera descrita, amb el "*QuartusII* Pinout Assignment", vèiem que un cop compilat el disseny si aquest s'obria amb el programa *QuartusII* no s'havia fet correctament l'assignació de pins, ja que duplicava els ports d'entrada/sortida posant un dels ports amb el nom correcte però sense assignar-li si era una entrada o sortida i un altre amb el nom del port però amb el prefix de i/o en funció de si era entrada o sortida. Donat que això succeïa cada cop que es feia l'assignació manual amb independència del dispositiu escollit, es va decidir posar-se amb contacte amb Altera a través del seu sistema de suport als clients, i després de fer diverses proves que es varen fer com a conseqüència de les suggerències del servei de suport, es va rebre la solució i no era altra cosa que fins la versió 6.0 del programa no s'havia resolt el problema d'assignació dels Pins. Per tant es va decidir canviar a la versió 6.1 donat que es disposava del *QuartusII* 6.1 que es la versió mínima per poder utilitzar el *DSPBuilder* 6.1. Tot seguit es veurà una imatge amb el problema descrit:


Named: * All Pins Filter: Pins:

	Node Name	Direction	Location
1	clock	Input	
2	data[0]	Output	
3	data[1]	Output	
4	data[2]	Output	
5	data[3]	Output	
6	data[4]	Output	
7	data[5]	Output	
8	data[6]	Output	
9	data[7]	Output	
10	href	Output	
11	hsync	Output	
12	pclk	Output	
13	pclk2	Output	
14	pclk3	Output	
15	reset	Input	
16	sclrp	Input	
17	vsync	Output	

18		~ASDO~	Input	
19		~nCSO~	Input	
20		odatas[0]	Bidir	PIN_C18
21		odatas[1]	Bidir	PIN_D18
22		odatas[2]	Bidir	PIN_D17
23		odatas[3]	Bidir	PIN_E17
24		odatas[4]	Bidir	PIN_F18
25		odatas[5]	Bidir	PIN_E19
26		odatas[6]	Bidir	PIN_F15
27		odatas[7]	Bidir	PIN_F20
28		ohsyncs	Bidir	PIN_D19
29		ovsyncs	Bidir	PIN_D20
30		opclks	Bidir	PIN_F17
31		opclk2s	Bidir	PIN_E18
32		opclk3s	Bidir	PIN_F16
33		iresets	Bidir	PIN_Y4
34		ohrefs	Bidir	PIN_C19
35		<<new node>>		

Figura 27.Assignació de Pins errònia del DSPBuilder 5.1

Un cop es va compilar amb la versió 6.1 es va veure que aquests errors estaven pràcticament solucionats, degut a que en el projecte que es va compilar no tenia entrades a excepció de la senyal de rellotge i la de reset, i l'assignació d'aquestes dues entrades i totes les sortides es fa correctament, però quan es posen més entrades no acaba de fer correctament tampoc l'assignació d'aquestes entrades, però si fent la resta de l'assignació correctament.

- La tercera fase en el desenvolupament d'aquest projecte final de carrera és pujar codi VHDL al *Simulink*/*DSPBuilder*, per la qual cosa es va fer ús del bloc "*HDL Import*" que el que fa es convertir els codis escrits en llenguatge de descripció de hardware (VHDL i Verilog) o un projecte ja creat amb *QuartusII* a un model de *Simulink*. A continuació es veuen unes captures de les parts més importants del *HDL Import*:

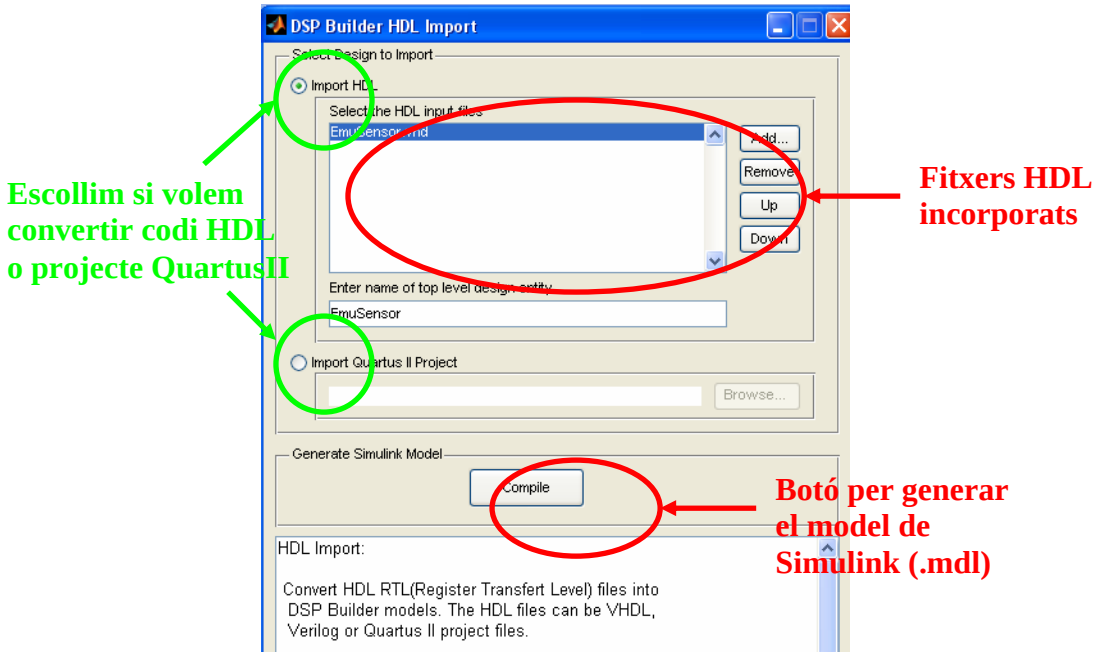


Figura 28.Vista de l'exportació del "HDL Import"

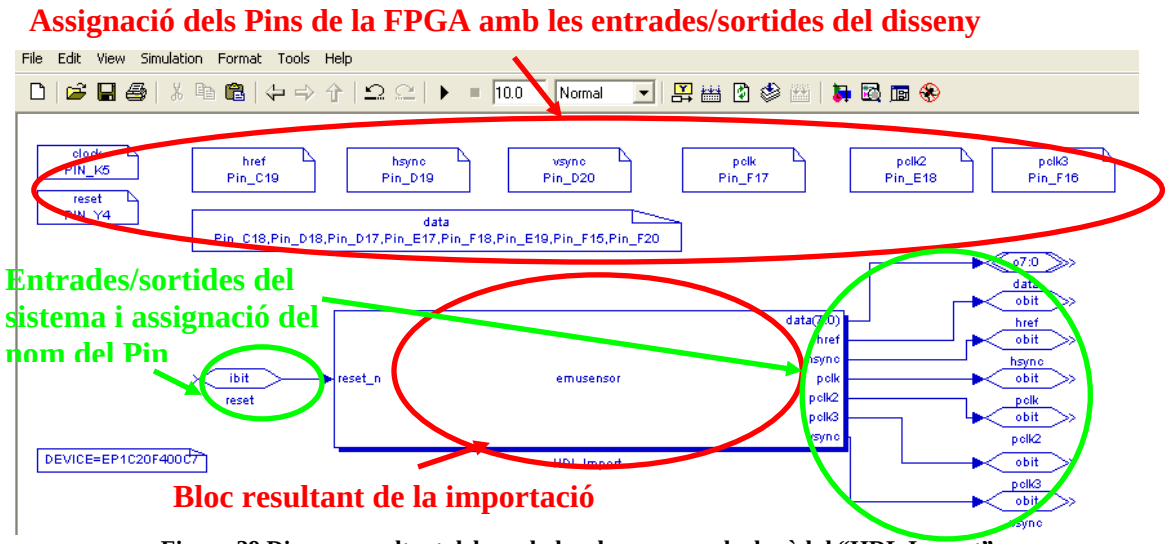


Figura 29.Disseny resultant del emulador de sensor amb el mòdul "HDL Import"

Cal dir que *HDL Import* té una sèrie de limitacions a l'hora d'importar fitxers VHDL i és que s'han d'utilitzar el tipus std_logic_1164, si el nostre disseny utilitza algun altre tipus de definicions, com ara l'aritmètic o el tipus numèric s'ha de fer una conversió a std_logic o std_logic_vector, ja que sinó la compilació ens donarà errors.

Una altra limitació de la importació de codi HDL és que només suporta dissenys amb sols un rellotge, entenent que un rellotge serà qualsevol senyal que en el nostre codi funcioni per flancs no per nivells, no només un o més senyals de rellotge

pròpiament dits. En cas que el sistema incorpori més d'un senyal que el *HDL Import* entengui com un senyal de rellotge, el manual del programa ens diu que un d'ells s'utilitzarà com a rellotge i la resta apareixen com a entrades del nostre sistema, però això no és del tot cert quan es posa en pràctica, ja que aquestes entrades després no es veuen en el bloc resultant de la realització del model de *Simulink*.

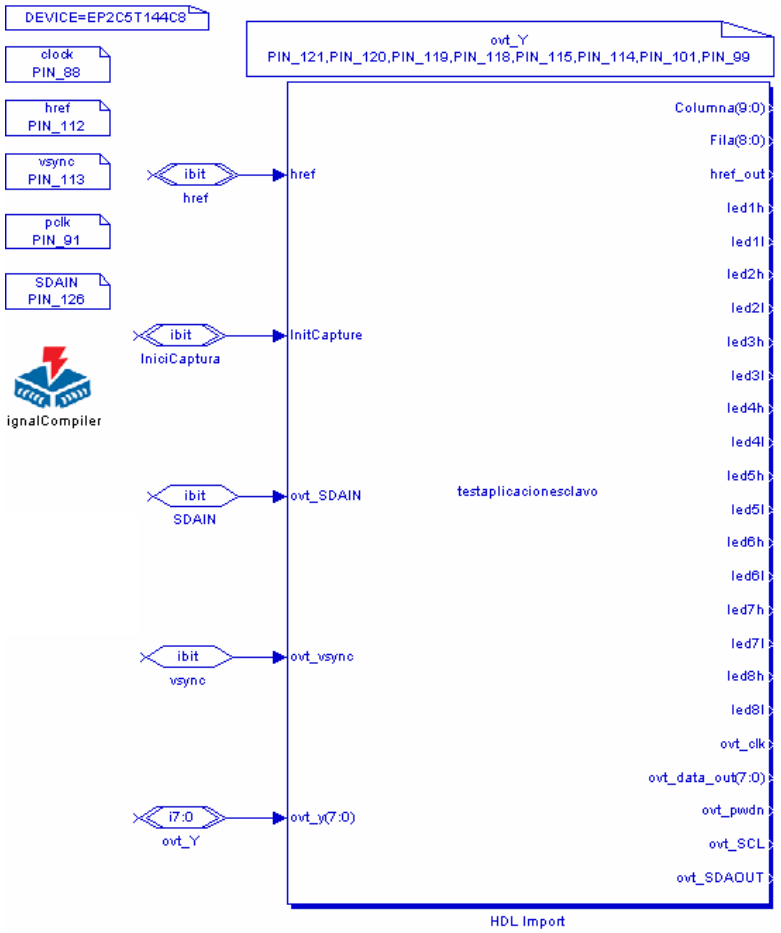


Figura 30.Disseny resultant del nostre sistema final

La imatge anterior ens mostra que no tenim accessible els nostres senyals de rellotge en el bloc resultant de la nostre importació de codi HDL, ni *CLK* i *PCLK*, però el que si que es pot fer és assignar un Pin a aquestes entrades, ja que els senyals d'entrada segueixen estant en el nostre codi.

Altres restriccions importants del *HDL Import* son per exemple el fet que no es poden importar dissenys que continguin senyals bidireccionals, cosa que no ens anirà

gaire be donat que per configurar el sensor d'imatge OVT7640 necessitarem d'una comunicació que ells anomenen SCCB, que no és altra cosa que un I2C propi, en el qual tenim dos senyals un de rellotge i un altra de dades que és bidireccional. S'ha de dir que el senyal de dades ha de ser bidireccional degut a que tant es pot escriure la configuració que volem en els diferents registres com llegir la configuració existent. En un principi el que s'ha fet ha sigut deixar el senyal com a sortida cap el sensor d'imatge, degut a que no era tan necessari llegir la configuració que té el sensor, però si en canvi ho és més el fet de poder escriure la nova configuració, per tant s'obviarà la rebuda del ACK. També s'ha de dir que no qualsevol Megafunció o LPM (Library of Parameterized Modules) es pot implementar, sinó que només es pot fer les descrites a la següent taula:

Supported Megafunctions and LPM Functions

Megafunctions		LPM Functions	
a_graycounter	altshift_taps	lpm_abs	lpm_mult
altaccumulate	altsyncram	lpm_add_sub	lpm_mux
altmult_add	parallel_add	lpm_compare	lpm_ram_dp
		lpm_counter	

Figura 31.Taula de components suportats per el *HDL Import*

I en canvi no es poden implementar les següents Megafuntions i LPM:

Unsupported Megafunctions and LPM Functions

Megafunctions		LPM Functions	
alt3pram	altmemmult	lpm_and	lpm_inv
altcam	altmult_accum	lpm_bustri	lpm_latch
altcdr	altpll	lpm_clshift	lpm_or
altclklock	altqpram	lpm_constant	lpm_pad
altddio	altsqrt	lpm_decode	lpm_ram_dq
altdpram	alt_exc_dpram	lpm_divide	lpm_ram_io
altera_mf_common	alt_exc_upcore	lpm_ff	lpm_rom
altfp_mult	dcfifo	lpm_fifo	lpm_shiftreg
altlvds	scfifo	lpm_fifo_dc	lpm_xor

Figura 32.Taula de components no suportats per el *HDL Import*

Un cop s'ha aconseguit importar satisfactòriament el codi del emulador del sensor OVT7640 al *DSPBuilder*, fent les pertinents modificacions degudes a les restriccions abans esmentades, i comprovar el seu correcte funcionament, es procedeix a fer el següent pas en el desenvolupament del projecte.

- La quarta fase del projecte és aconseguir sintetitzar l'eina de *Matlab*/*Simulink*/*DSPBuilder* Hardware In the Loop (*HIL* a partir d'ara) sobre la FPGA i que els resultats que doni siguin els esperats, és a dir que el sistema faci una captura de la imatge, la mostri per pantalla i l'usuari des de *Matlab* pugui modificar els registres de configuració del sensor d'imatge. Abans d'explicar el que s'ha fet es fa una introducció al que és el mòdul *HIL*.

El mòdul *HIL* és un bloc que permet al *Simulink* co-simular un disseny creat pel software *QuartusII* físicament en una FPGA implementant una part o tot el nostre disseny. Recordem que un disseny fet amb el *DSPBuilder* i compilat amb el SignalCompiler també és un projecte de *QuartusII*. El principal motiu d'utilitzar el bloc *HIL* és que el nostre disseny s'està executant directament sobre la FPGA amb les lògiques avantatges que suposa fer-ho d'aquesta manera, a més de poder utilitzar totes les llibreries i blocs de *Simulink* com per exemple la utilització de filtres d'imatge, fer qualsevol tractament de les dades de la imatge o per altres aplicacions diferents a la que ens pertoca en aquest projecte col·locar un analitzador d'espectre, utilitzar generadors de funcions...

Per crear un disseny que inclogui el mòdul *HIL* el que hem de fer és:

- Com ja s'ha avançat, crear el projecte de *QuartusII* que es vol co-simular.
- Un cop ja es té dissenyat, s'ha de compilar amb el *QuartusII/DSPBuilder.*
- A continuació s'ha d'afegir el bloc *HIL* en el model de *Simulink* i connectar-hi la instrumentació pertinent al bloc de *HIL*, per exemple els ports d'entrada i sortida.
- S'ha d'especificar els diversos paràmetres del bloc *HIL*:
  - El projecte de *QuartusII* compilat que defineix la funcionalitat del sistema
  - Les característiques dels ports d'entrada/sortida
- Tot seguit s'ha de prosseguir amb la compilació del bloc de *HIL* per la FPGA escollida.

- Un cop el bloc estigui compilat sense errors s'ha de programar la FPGA via JTAG
- Només resta simular el sistema al *Simulink*

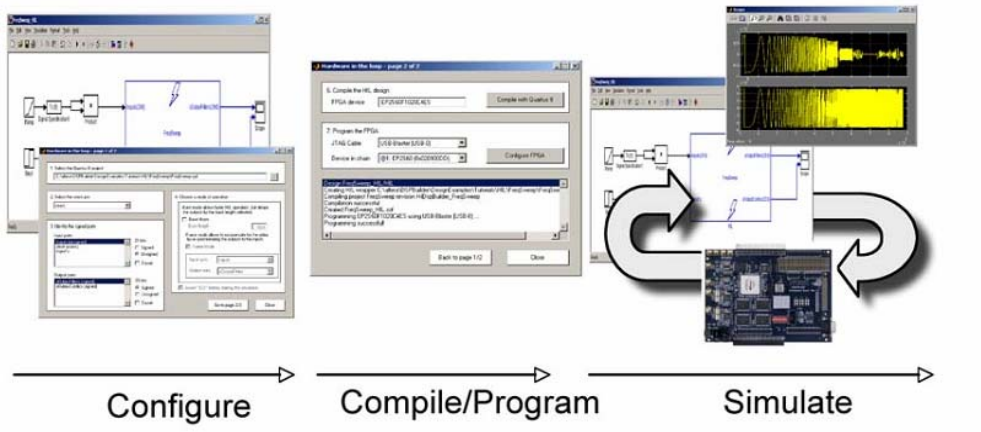


Figura 33.Diagrama de flux d'un disseny Hardware In The Loop

Assignació del projecte de QuartusII que volem incloure al *HIL*

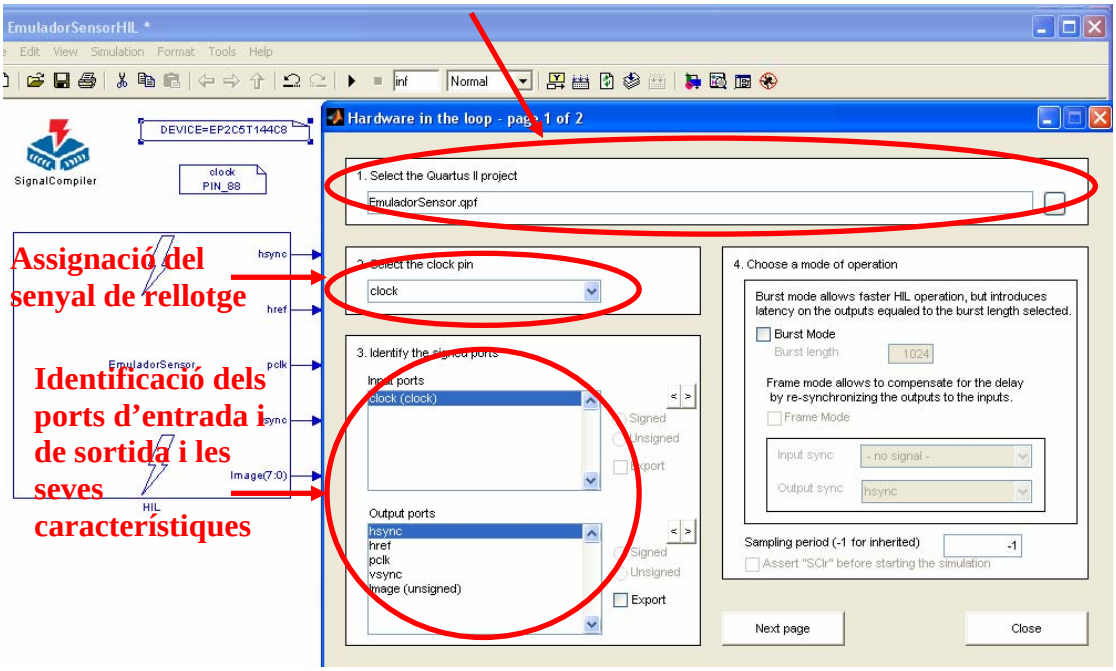


Figura 34.Vista de l'assignació del projecte a fer el HIL

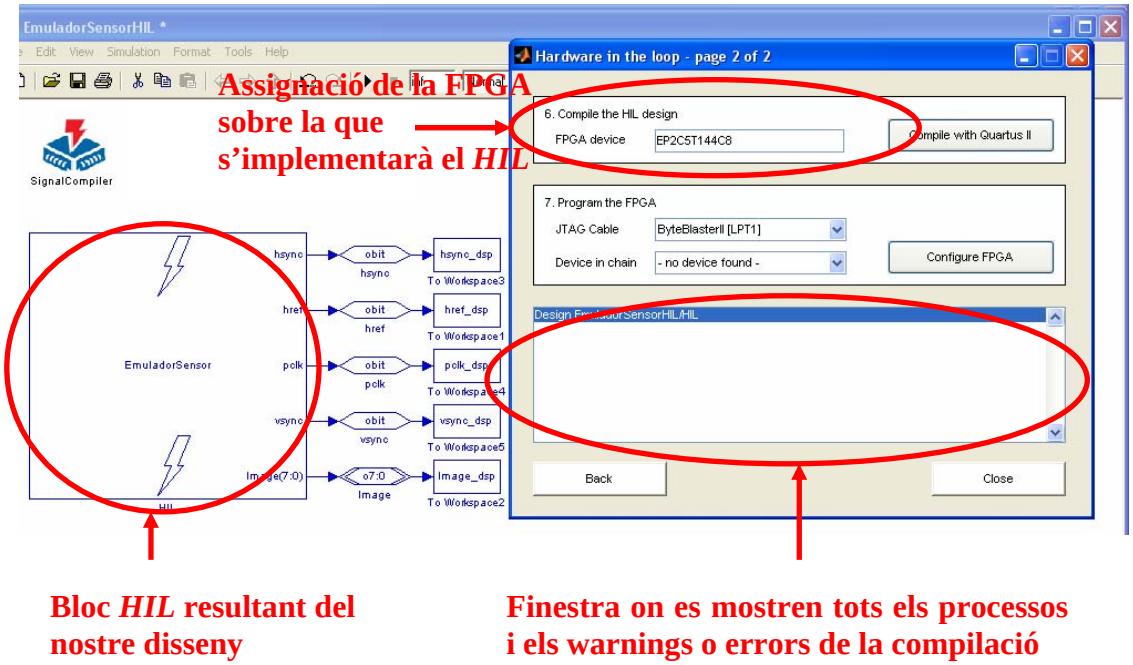


Figura 35.Vista de l'assignació de la FPGA a fer el HIL

A les imatges anteriors s'ha vist com es fa per crear un disseny que compleixi els requisits per poder ser un projecte de *HIL*.

Per poder avaluar si el sistema *HIL* funcionava el que primer es va fer es utilitzar un dels exemples que acompanyen al programa *DSPBuilder*, que a més a més ens anava molt be, doncs es tracta d'enviar una imatge des de *Matlab* i fer que la FPGA fes una sèrie de tractaments de la imatge i després enviï les dades de la imatge i el *Matlab* les converteixi en imatges visualitzables de nou. Un cop es compilat i co-simulat via *HIL* es pot comprovar que funciona sota aquestes característiques de llegir una imatge en format ".jpg", fer un tractament d'aquesta i posteriorment enviar la imatge resultant, però s'ha de dir que per poder veure aquestes imatges el *Matlab* s'ha d'executar una StopFunction , que no es altra cosa que un o més fitxers escrits en llenguatge de *Matlab* per que aquests s'executin un cop es para la simulació que en el cas de l'exemple el que fa és convertir les dades de la imatge que són d'un array d'una dimensió en una matriu de dues dimensions, que es la que dona la resolució de la imatge (640 x 480 en el nostre cas i també en el del exemple).

Un cop comprovat aquest exemple, es va fer primer la prova de crear un disseny *HIL* que inclogués el nostre emulador de sensor i que ens envies imatges al PC. És van crear diverses imatges predefinides com s'explicarà més endavant.

Un cop validada la prova de fer el *HIL* amb l'emulador de sensor, es decideix fer el *HIL* amb el sistema complert, és a dir, la FPGA adquirint directament del sensor OVT7640. Per fer això es necessita implementar un mòdul en VHDL que ens permeti configurar els sensor amb la comunicació SCCB (recordem que és igual a la comunicació I2C), que controli els leds del sistema MiraKonta, que adquireixi les dades quan correspongués, i que faci un downsampling per deixar la imatge amb una resolució de 320 x 240. Per fer això el que fa es agafar només els píxels vermells, donat que com recordem els leds són de color vermell, i degut a això la resta de píxels (els verds i els blaus) no tenen gaire informació sobre la imatge.

S'ha de dir que com es pot veure en l'Annex 1 no s'ha pogut realitzar la implementació del disseny amb el mòdul *HIL*, donat que tenim la FPGA connectada a un dispositiu hardware en el qual la freqüència és un apartat especialment sensible, com és el sensor d'imatge, ja que l'execució de la simulació *HIL* ralentitza el sistema, si el comparem amb el funcionament normal i això juntament amb el tema de la impossibilitat de crear ports bidirecionals no han permès dur a terme el bloc *HIL* del disseny global.

- En un últim apartat tindríem la utilització del programa *QuartusII*, que s'ha de dir que s'ha fet servir de manera paral·lela al *DSPBuilder*, per diverses raons una d'elles es el ja esmentat problema amb l'assignació de pins de la versió 5.1 del *DSPBuilder*, un altra raó ha sigut per contrastar els resultats obtinguts amb el mòdul *HIL*, a la vegada que per inspeccionar variables o senyals del nostre projecte de *QuartusII*, donat que encara que el *DSPBuilder* ens ofereix una sèrie d'eines per fer això s'ha preferit l'utilització directament del *QuartusII* donat que aquest entorn és més potent de cara a fer segons quines coses que el *DSPBuilder*, ja que aquest últim s'ha intentat

fer en un entorn més “amigable” i fàcil d’utilitzar, però amb l’inconvenient de no poder escollir totes les opcions que ens dona un entorn com el *QuartusII*.

Capítol 3.
Disseny i Implementació

3.1. Disseny de components/mòduls

El que hem de fer per a poder controlar les dades del sensor d'imatge i la seva configuració és crear uns mòduls que ens permeten dur a terme el projecte:

- **Mòdul I2C/SCCB**: Aquest mòdul s'encarrega de fer la conversió de les dades que arriben del mòdul de configuració que són en mode paral·lel al mode sèrie propi del *I2C*, és a dir, amb les característiques pròpies del protocol, per exemple després d'enviar una comana espera un *ACK* (Acknowledgement).
- **Mòdul Configuració:** S'encarrega d'enviar al mòdul de I2C el valor dels diferents registres, com els del guany de color vermell, el format de les dades, o la divisió del rellotge intern (prescaler). També realitza la configuració de la il·luminació dels leds, és a dir, és el responsable de quins leds s'il·luminen i amb quina intensitat.
- **Mòdul de Downsampling:** Fa la conversió de la resolució que dona el sensor de 640 x 480 a la meitat. Per fer-ho lògicament necessita d'un mòdul calculador de la coordenada actual
- **Mòdul de coordenades:** Com el seu nom indica la seva funció es calcular la posició actual a la que està la imatge.

Ara un cop s'han explicat els diversos mòduls per entendre millor com interactuen entre ells el que es mostra a continuació és un diagrama de blocs dels mòduls del disseny del hardware final de la FPGA.

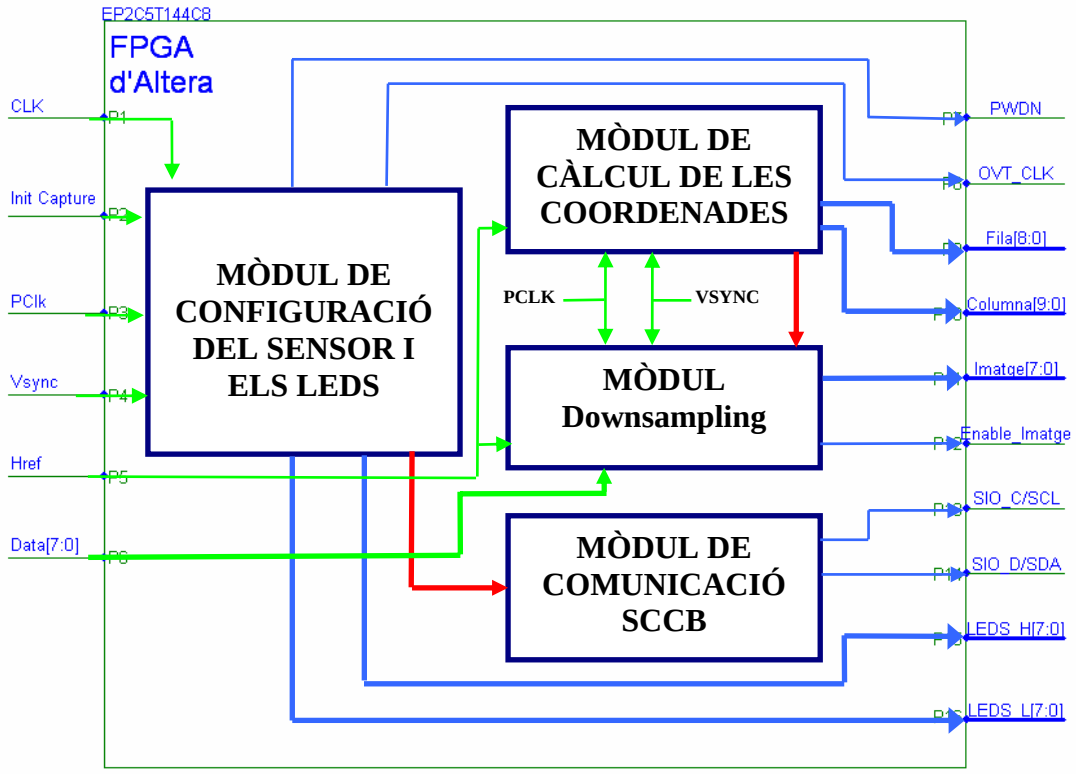


Figura 36.Diagrama de blocs del disseny Hardware sintetitza a la FPGA

S'ha de dir que la idea inicial era fer que els registres de configuració del sensor fossin accessibles per l'usuari, entenent com a registres modificables per l'usuari aquells que canvien la lluminositat del sensor, el temps d'exposició, etc, no aquells que modifiquen paràmetres com ara el format de les dades o la velocitat de sortida de les mateixes. Al final no s'ha fet degut als problemes que s'han fet referència en el apartat on es parla de les incompatibilitats del mòdul *HIL* amb aquest projecte, i al no existir una interfície per a poder comunicar-nos amb la FPGA, el valor dels registres s'han fet per mitjà de constants que es predefineixen.

Ara tot seguit es mostra un diagrama de blocs del sistema dinal resultant incloent-hi tant el sensor d'imatge OVT7640 com la FPGA:

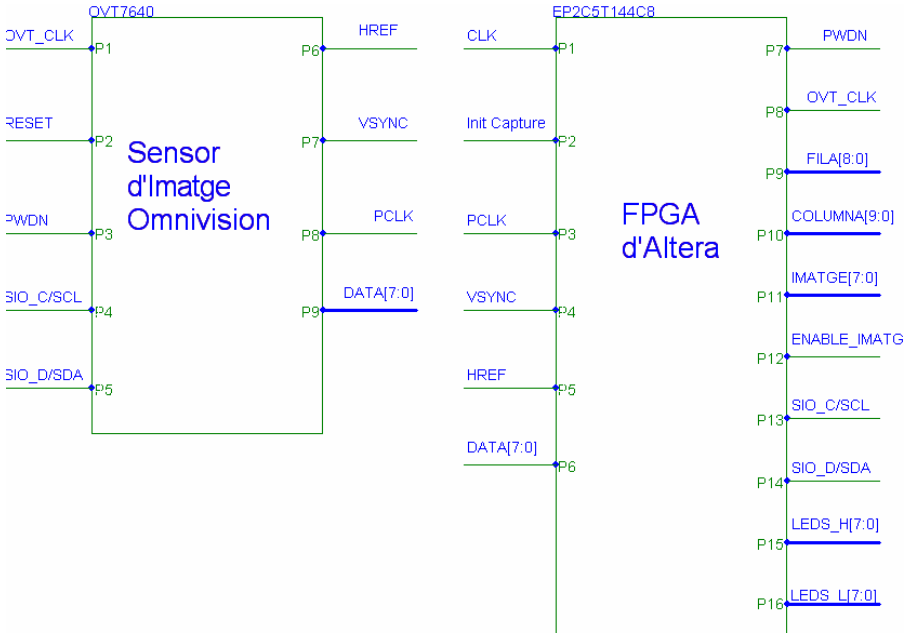


Figura 37.Diagrama de blocs general del sistema

La següent imatge és una captura de pantalla de l'esquemàtic del sistema utilitzat:

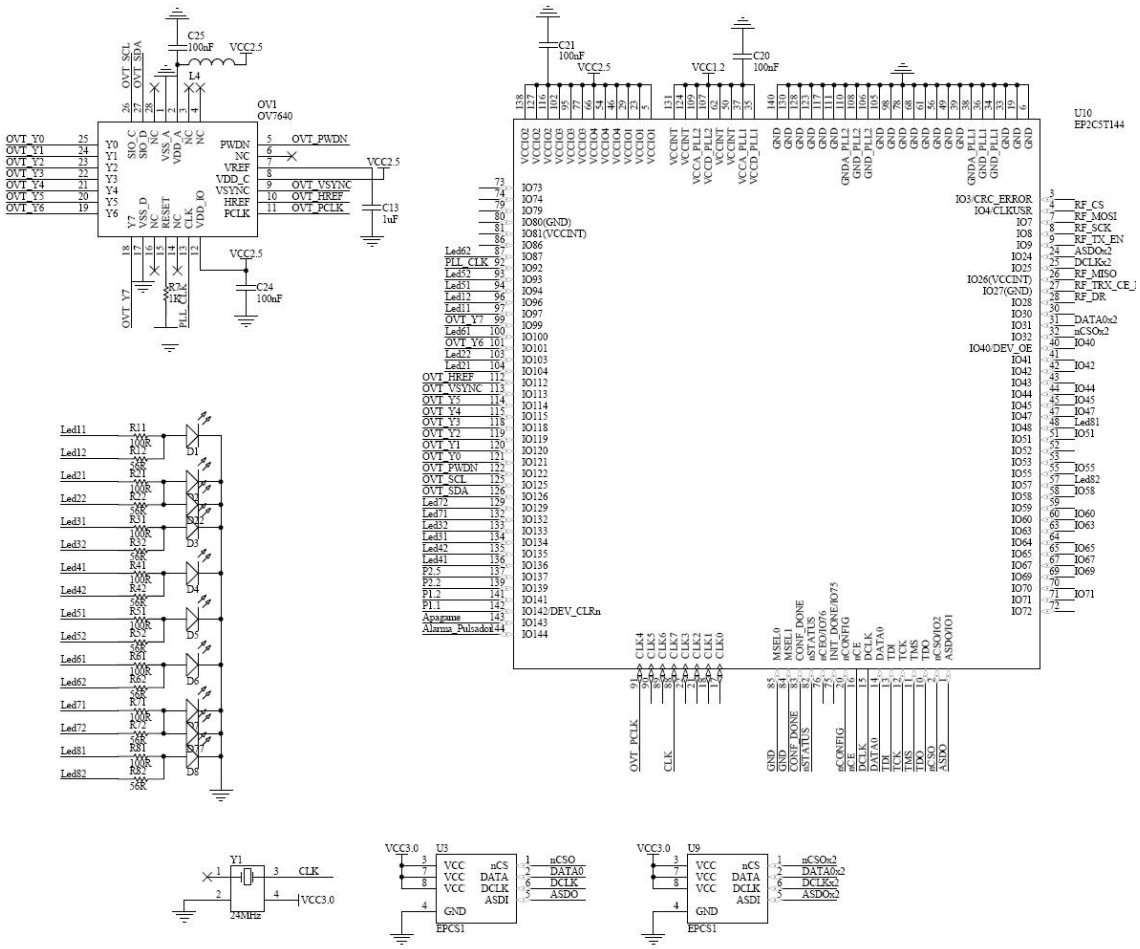


Figura 38.Diagrama de blocs general del sistema

Capítol 4.

Test i Resultats

4.1. Resultats de simulació

Com ja s'ha avançat en l'apartat de metodologia el primer que es va simular va ser el codi de l'emulador de sensor. Per fer-ho es va utilitzar l'entorn de *ModelSim* que recordem que ens donarà el valor dels resultats en formes d'ona, és a dir podrem mirar el valor de les diferents senyals en cada instant de temps però no podrem composar un imatge com a resultat de simulació. Per això i per que les imatges les havíem de predefinir mitjançant codi VHDL, s'han hagut de crear una sèrie d'imatges senzilles de programar, per exemple es van crear imatges en blanc i negre, en les quals en una d'elles es va fer que la meitat dreta fos negre i l'altre meitat blanca. Un altra en que fos una meitat superior blanca i la meitat inferior negre, una que fos dos píxels negres dos blancs i així successivament.

Aquestes imatges també tenen un altra perquè, i és que al fer el downsampling de la imatge observarem si ho hem fet malament de seguida, donat que veurem dues meitats exactes de la imatge o dos píxels del mateix color seguit, en el cas de l'altre tipus d'imatge, ja que recordem que el downsampling el fem per només utilitzar els píxels vermells, agafem un píxel sí un altre no, i això només a les files senars:

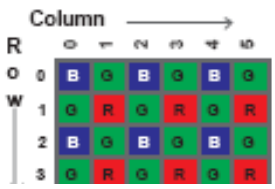


Figura 39.Vista de la distribució dels píxels en mode Bayer Pattern

Com acabem d'avançar també s'ha de simular un parell de components més, i son el mòdul que fa el downsampling, que s'ha anomenat "*ONLYRED*" donat que només deixa passar els píxels vermells, i per fer-ho necessitarà d'un altre mòdul que li vagi indicant les coordenades de les dades que ens arriben, que s'ha anomenat "*COORD*".

A continuació es pot veure una captura de pantalla de les formes d'ona de l'emulador de sensor amb els mòduls ja esmentats "*COORD*" i "*ONLYRED*", a on es pot observar que el senyal de *HREF* ens fa d'enable de les dades de la imatge. Aquest senyal roman actiu al llarg cada Fila i de les 640 Columnes, es a dir, que es posa actiu un total de 480 vegades, cadascuna d'elles té una durada de 640 cicles de rellotge. També es pot observar com les dades de sortida de la imatge són la primera meitat de cada fila tots els bits a '1', essent les dades de 8 bits, correspon al número 255, i aquest valor correspon al color blanc, i la segona meitat tots els bits es troben a '0', i correspon al color negre.

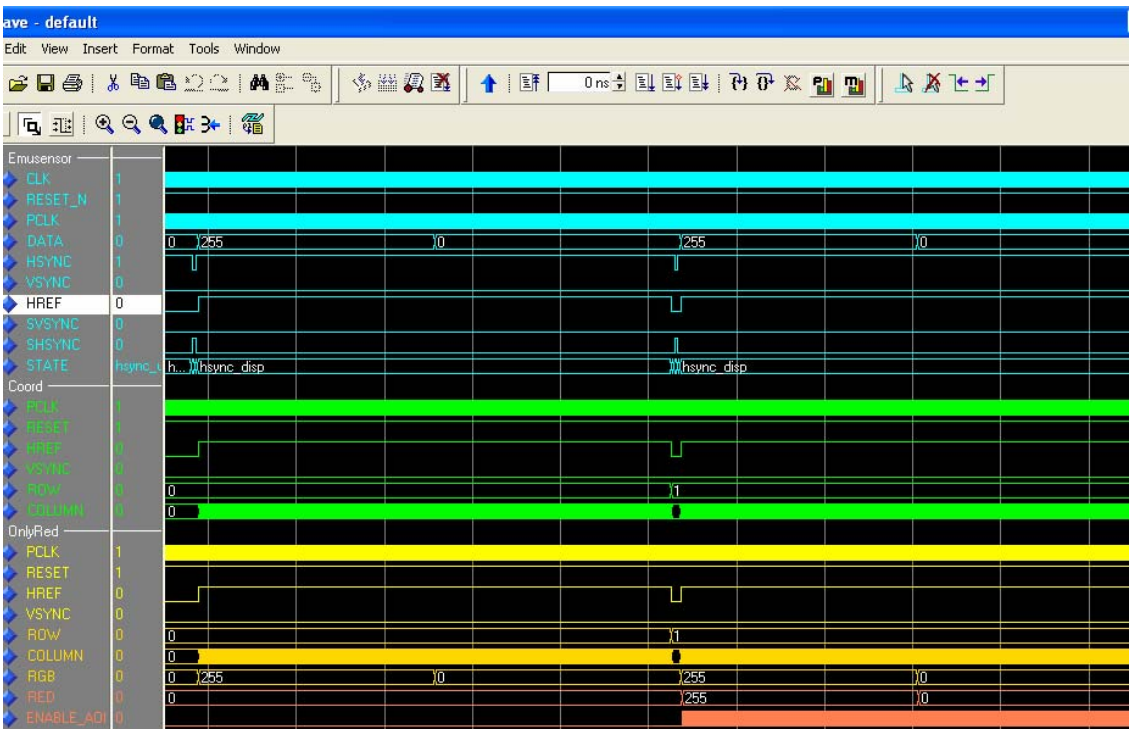


Figura 40. Resultats de la simulació amb ModelSim del sistema amb l'emulador de sensor

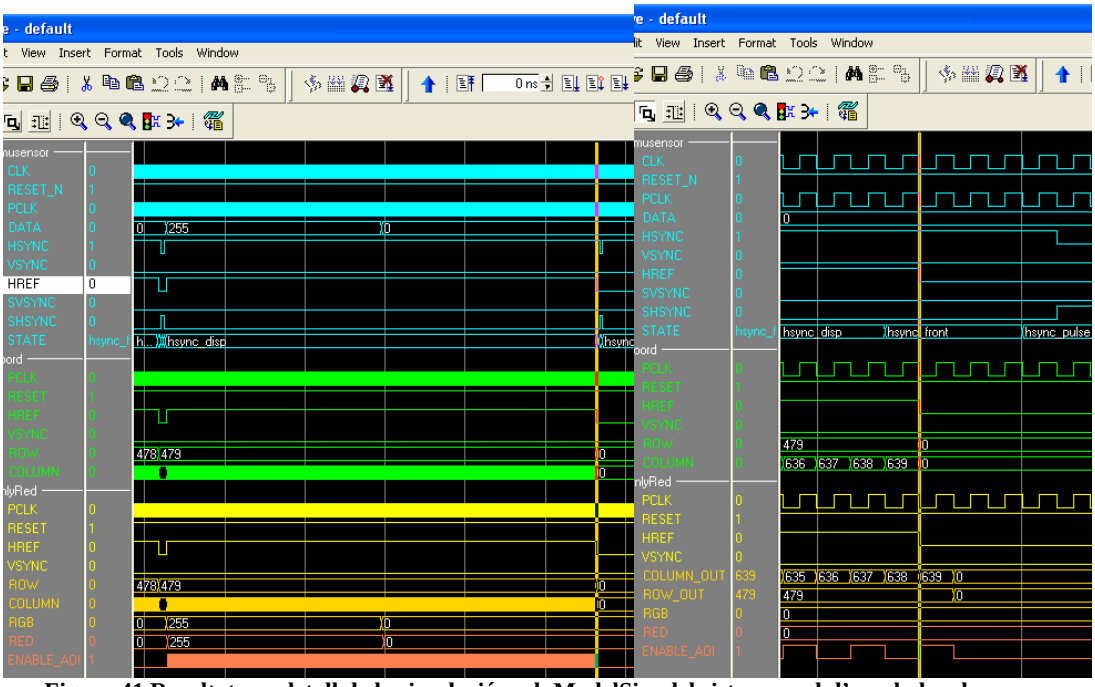


Figura 41.Resultats en detall de la simulació amb ModelSim del sistema amb l'emulador de sensor

En aquesta segona i tercera captura (les dues imatges que es troben juntes a la pàgina anterior), s'observa com l'emulador del sensor arriba a comptar les 640 columnes i les 480 files. Observem també que la senyal "*ENABLE_AOI*" que fa d'enable per dir que les dades que hi ha a la sortida son vàlids, és a dir ,quan són senars tant les files com les columnes.

Un cop el nostre codi funciona correctament en l'entorn de *ModelSim* s'ha de confirmar que també ho fa en l'entorn de *Matlab*/*Simulink*. Cal dir que per la simulació s'ha optat per utilitzar visualitzadors de les formes d'ona, per comprovar el seu funcionament valor a valor comparat amb el *ModelSim*. A la següent imatge es veu com el senyal de dades va fent polsos que van de 0 a 255 donat que els resultats pertanyen a una imatge que la meitat esquerra és blanca i l'altra meitat és negre :

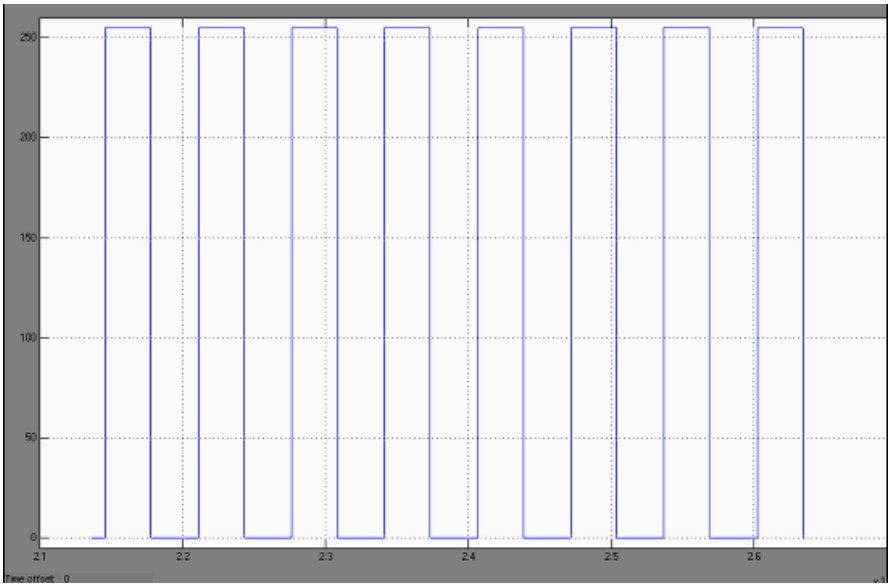
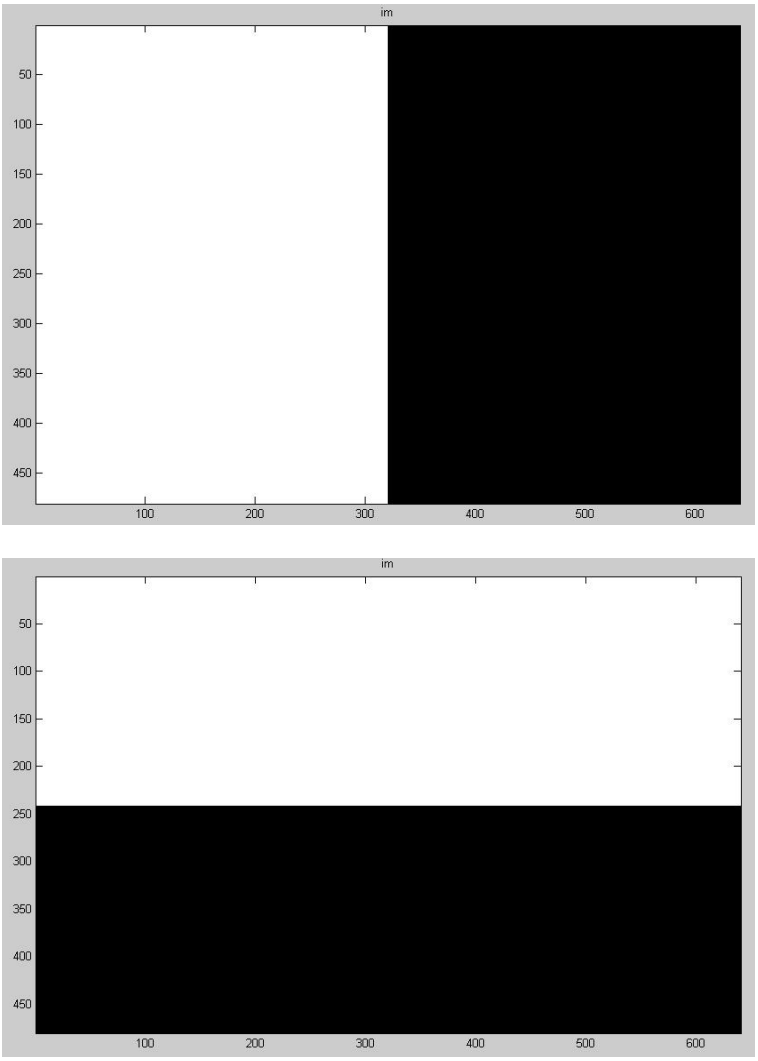


Figura 42. de la simulació amb Matlab de l'emulador de sensor

Com arribats a aquest punt ja es té simulat la part de l'emulador del sensor OVT7640, junt amb la part que extreu únicament els píxels vermells, només resta simular la part de configuració del sensor. Aquesta part el que s'ha fet és un component de comunicació I2C en codi VHDL, i un mòdul que contingui aquest component i que configuri el sensor amb els paràmetres desitjats. Un cop simulat i vist el seu correcte funcionament s'ha passat a fer el següent apartat que és sintetitzar els codis i baixar-los a la FPGA.

4.2. Resultats de síntesi

Donat que fins ara només s'han vist els resultats del sistema simulat, resta per veure els resultats de la seva síntesi física en la FPGA. El primer que es s'implementa com ja s'ha avançat amb anterioritat és el mòdul *HIL* amb l'emulador de sensor. Per fer-ho es s'implementen un parell d'imatges que ja s'han comentat amb anterioritat, que són les imatges en les que una meitat és negre, i l'altra blanca, amb la diferencia que una es fa la meitat vertical i a l'altra la horitzontal. Tot seguit es poden veure les imatges que ens arriben des del *Matlab* tot passant per la FPGA:



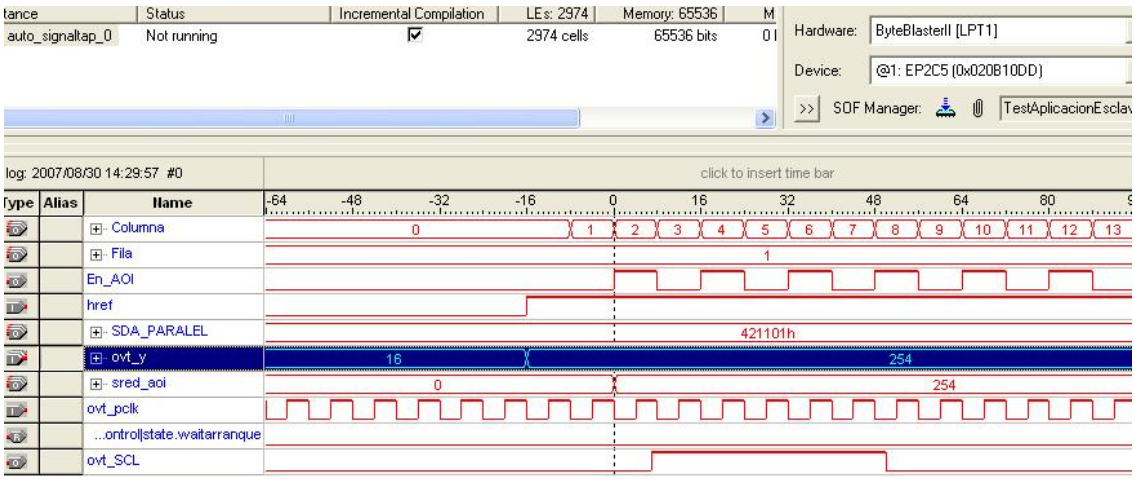
Figures 43 i 44.Imatges resultants del HIL dels diferents emuladors de sensor

Cal dir que per poder veure les imatges s'ha d'inclòs alguna funció més al fitxer de StopFunction per entre d'altres coses només agafar les dades quan el senyal d'enable estigui actiu, convertir les dades a 8 bits o donar la resolució de la imatge resultant. També s'ha de fer referència al fet que al realitzar aquesta prova del *HIL* només s'ha fet de l'emulador de sensor no de la part que fa el downsampling, i és per això que si ens fixem en les imatges, aquestes tenen una resolució de 640 x 480 píxels. Un cop vists els resultats es pot començar a intuir que el nostre sistema dins el mòdul *HIL* podria funcionar correctament. Així que s'ha de fer el mòdul *HIL* que ens inclogui la part de configuració del sensor i de l'adquisició de la imatge, per tant haurà de tenir un component que agafi les dades quan toqui i faci el downsampling, a més de tenir un altre component de comunicació per I2C i una part que enviï la configuració desitjada al sensor. Quan diem que enviï la configuració pertinent ens referim a :

- Que les dades tinguin el format RGB
- Desactivar els controls automàtics de guany i exposició, doncs és un dels paràmetres que volem variar per la caracterització.
- Escollir la freqüència del sensor mitjançant un preescaler, que el que fa és dividir per la freqüència del senyal de rellotge d'entrada.
- Configurar les dades de sortida en formar RAW (Read After Write).
- Per últim configurar el sensor amb els paràmetres ajustables per a realitzar la caracterització, que són:
  - Guany general (de tots els colors).
  - Guany del color vermell
  - Saturació
  - Brillantor
  - Exposició

Un cop s'ha creat el mòdul *HIL* del nostre disseny, i ens posem a simular-lo en la FPGA, ens adonem que les imatges no són com esperàvem, és a dir, surten totes

blanques de dalt a baix i si ens dediquem a observar el valor de les dades veiem que aquestes solen estar al voltant del valor 254, la qual cosa vol dir que els píxels estan saturats de lluminositat, per tant el que es decideix fer és la prova de tapar la lent del sensor, però el resultat és el mateix, així que es decideix mirar el bloc amb deteniment des de *QuartusII* i lògicament sense el mòdul *HIL*, és a dir, el que hem fet és agafar el projecte que es fa servir per crear el mòdul *HIL* i s'ha afegit el SignalTab amb les variables que ens interessa observar, com són el senyal de sincronisme *HREF*, les dades que provenen del sensor, les que es treuen després de fer el downsampling , el senyal de dades que s'envien per I2C, els senyals que ens indiquen en la fila i la columna en la que estem. El primer que s'observa es que efectivament amb la lent destapada i a plena llum del dia al realitzar una fotografia aquesta dona valors molt propers a la saturació de color, es a dir, està sobre el valor 254, sempre parlant dels píxels vermells doncs són aquests els que acabem configurant. Cal dir que aquesta situació podria ser entesa com normal, donat que en un principi es parteix de la configuració que es té per el sistema MiraKonta, que no oblidem treballa quasi a les fosques, ja que l'única font de llum en aquestes condicions són els leds, i per tant si s'il·lumina amb llum natural ens trobem amb reaccions com la descrita, de valors propers al màxim, com es pot veure a continuació en les dues imatges següents del SignalTab:



Figures 45 i 46.Captura dels resultats del SignalTab amb la lent destapada

En aquestes imatges cal explicar que el senyal *EN_AOI* es l'enable de que les dades de la sortida són les correctes, aquesta senyal de sortida, anomenada *SRED_AOI* són les dades corresponents als píxels vermells, per això només es treuen les dades corresponents a fila i columna senar. Observant les dades que surten del sensor sense fer el downsampling veiem que tots els valors son alts, però els més elevats corresponen als píxels vermells, i això es com ja s'ha comentat degut al guany que li configurem al sensor, i per això veiem que el valor d'aquests píxels és sempre de 254. També cal comentar que fem la inspecció de la senyal *SDA_PARALEL* que no és altra cosa que tota la comanda que es fa al sensor per I2C, però enlloc d'anar serialitzada s'ha tret de forma paral·lela per poder fer-la més entenedora al observar-la, per exemple veiem que en la segona imatge té el valor de "4213A0", això vol dir que es vol escriure, donat que l'adreça "42" és la corresponent a escriure en el sensor OVT7640 segons el datasheet, al registre "13" hi es vol escriure el valor de "A0".

Un cop s'ha fet aquesta prova es passa a fer tot el contrari, que és tapar la lent, i veure que és el que passa. El primer que veiem ara amb el *QuartusII* és que si que hi ha variació dels valors registrats en la captura, donat que ara el valor dels píxels vermells estan al voltant de 30 o menys en alguns casos. El fet que doni al voltant de 30 i no de 0, com en un principi es podria pensar al tapar la lent, segurament ve donat per que no s'ha tapat la lent amb quelcom que sigui totalment opac i al estar configurat en un principi per treballar en ambients foscos dona com a resultat el que s'acaba d'esmentar. De fet s'ha comprovat que baixant el guany general del sensor s'aconsegueixen valors més

propers a 0. Per poder corroborar el que s'ha comentat sobre la resposta del sistema quan es tapa la lent s'adjunta una imatge del SignalTab per poder veure amb més detall tot el que s'ha comentat:

Figura 47.Captura dels resultats del SignalTab amb la lent tapada

Si es comparen aquests resultats amb els obtinguts amb el *Matlab* s'observa que els que es treuen amb el *Matlab* sempre es troben a valors molt alts, quasi de saturació, més concretament al valor de 254, tant en el cas de fer la prova amb la lent destapada com tapada.

Fins ara s'ha parlat dels resultats numèrics enregistrats amb el *Matlab* al executar el mòdul de *HIL*, però com ja s'ha dit aquestes dades s'enregistren amb un array d'una dimensió, i la imatge necessita d'un array de dues dimensions, i també s'enregistren a cada cicle de rellotge del sistema, per tant per composar la imatge també s'ha d'eliminar els valors en el que el senyal d'enable no és actiu. Un cop fet això les imatges resultants són les següents.

Aquesta primera imatge correspon a el sistema capturant a plena llum del dia amb la lent destapada:

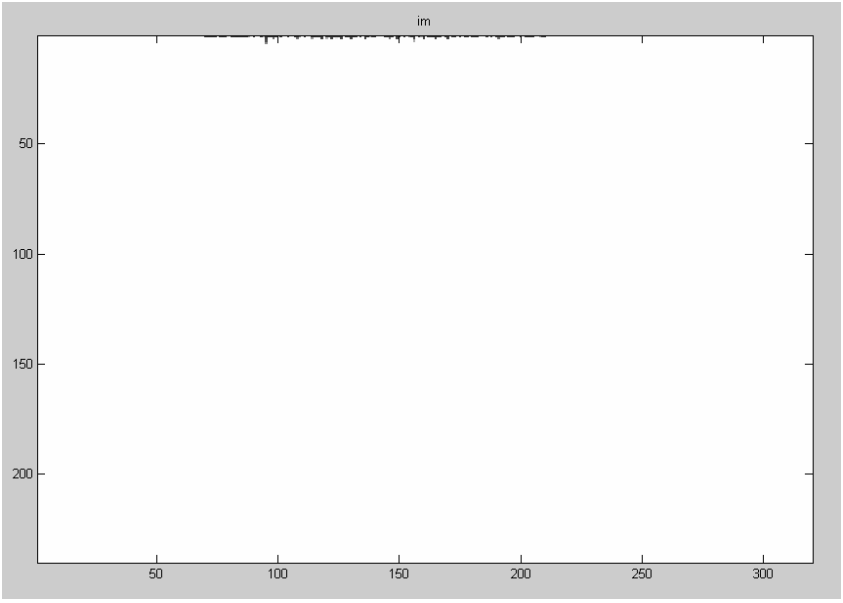


Figura 48.Imatge resultant del mòdul HIL amb la lent destapada

La segona imatge correspon al sistema amb la lent tapada:

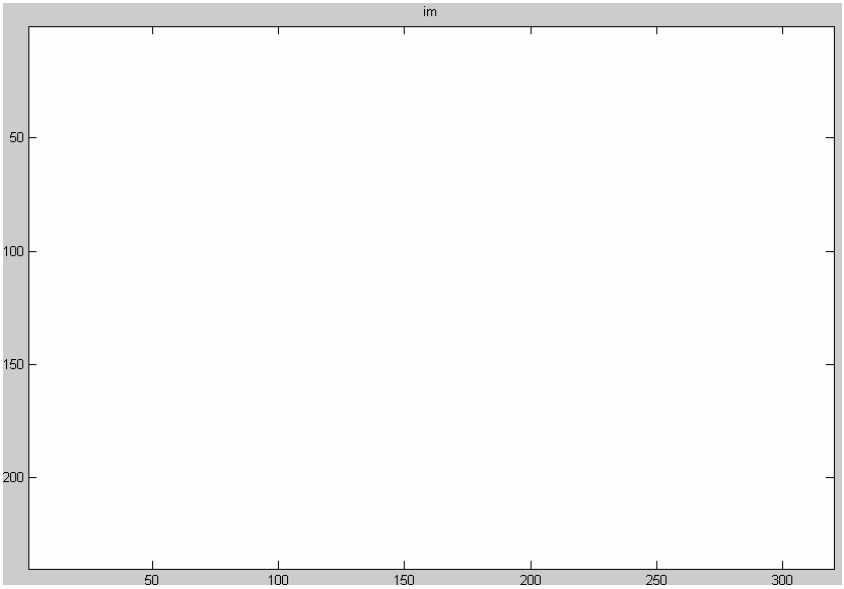


Figura 49.Imatge resultant del mòdul HIL amb la lent tapada

A trets generals podríem assegurar que amb les dades capturades al *Matlab* no trobem diferències substancials entre la prova de fer una captura d'imatge amb la lent destapada i una altra amb la lent tapada, o inclòs amb el sistema dins de la "txapela" capturant una imatge dels dígits del comptador.

Com que els resultats entre un programa i l'altre varien molt es va decidir, després de no trobar el perquè d'aquesta variació, posar-nos en contacte amb el fabricant de les dues eines, *QuartusII* i *DSPBuilder*, és a dir, amb Altera, i aquest després de comentar-li el que fèiem i quins dispositius s'utilitzaven, ens va aconsellar no utilitzar el mòdul *HIL*, per diversos motius que es resumeixen a continuació:

- No suporta dissenys amb múltiples senyals de rellotge, i el nostre disseny que va en la FPGA s'encarrega de crear el senyal de rellotge del sensor.
- No suporta dissenys amb ports bidireccionals, per tant la realització de la comunicació I2C amb el sensor queda tallada en un sentit. Com que s'havia d'escollir es va decidir que fos de sortida de la FPGA cap el sensor, donat que sinó no es podrien modificar els registres de configuració, però d'aquesta manera el protocol I2C no funciona degudament al no rebre l'ACK.
- El sensor d'imatge és un dispositiu altament sensible a la freqüència, i òbviament el fet que sigui el *Matlab* el que controli l'execució de la FPGA durant la simulació *HIL* fa que el sistema treballi a una velocitat inferior a la del seu funcionament normal, i per tant els resultats que ens dona el sensor no són correctes.

Les converses mantingudes amb el servei de suport d'Altera respecte d'aquest problema detectat poden ser consultades a l'annex 1.

4.3. Validació a nivell de sistema

Com es pot apreciar a les converses mantingudes amb el servei de suport de la casa Altera (veure annex), el projecte no s'ha pogut fer de la manera que inicialment s'havia previst, doncs hi ha una sèrie de limitacions que tenen tant el mòdul *HIL*, com el *HDL Import*. Aquestes limitacions fan referència a que el *HDL Import* no suporta els pins bidireccionals, i el mòdul *HIL* no pot contenir dissenys multirellotge. El fet de no suportar dissenys multirellotge ens és un problema, doncs la FPGA és l'encarregada de crear el senyal de rellotge del sensor d'imatge, el qual serà de la meitat de freqüència d'entrada, és a dir el senyal de rellotge del sensor d'imatge és de 12 MHz, ja que l'oscil·lador de la FPGA és de 24 MHz. Un altre problema és que el sensor d'imatge té les següents restriccions en quant a freqüència del rellotge es refereix:

Symbol	Parameter	Min	Typ	Max	Unit
Inputs (PWDN, CLK, RESET)					
f_{CLK}	Input Clock Frequency	10	24	27	MHz
t_{CLK}	Input Clock Period	100	42	37	ns
$t_{CLK:DC}$	Clock Duty Cycle	45	50	55	%
$t_{S:RESET}$	Setting time after software/hardware reset			1	ms
$t_{S:REG}$	Settling time for register change (10 frames required)			300	ms

Figura 50.Taula de freqüències acceptades pel sensor OVT7640

Observem doncs que el sensor té un rang de freqüències bastant acotat, ja que va de 10 a 27 MHz, i una de les característiques del mòdul *HIL*, es que simula sobre la pròpia FPGA, però ho fa a una freqüència bastant inferior a la de funcionament normal, i aquí tenim una altra incompatibilitat amb el mòdul *HIL*, doncs el sensor no rep en l'execució del mòdul *HIL* el rellotge amb suficient velocitat com per funcionar directament. Es podria pensar doncs en la col·locació d'un rellotge extern per al sensor, com per exemple la utilització del mateix oscil·lador que fem servir per la FPGA per exemple, però això seria contradictori doncs el sistema no aniria síncron, doncs el sensor aniria a velocitat normal, és a dir, elevada, i la FPGA aniria a velocitat de simulació, baixa. A més a més cal dir que els exemples de mòduls *HIL* proporcionats per Altera que s'han vist, en el cas de tractament d'imatge el que feien era amb una imatge ja emmagatzemada al PC enviar-la a la FPGA, tractar-la i després enviar-la, i és obvi que

el nostre cas es ben diferent, ja que el que s'ha de fer és llegir directament del sensor d'imatge a la velocitat que marca aquest i és aquí la nostre font de problemes per fer el *HIL*, doncs el nostre disseny incorpora un Hardware extern a la FPGA i que a més a més és molt sensible al tema de "timings". Tot això que acabem de comentar sobre els problemes de control d'un Hardware extern ens van ser ratificats amb la prova de realitzar el mòdul *HIL* amb l'emulador de sensor, que com s'ha vist en el corresponent apartat ha funcionat, doncs per realitzar aquest disseny només hem necessitat de Hardware la FPGA, però s'ha de dir que el temps per a capturar una imatge al simular el bloc *HIL* és òbviament molt superior al d'una captura en funcionament normal.

Un cop s'han provat diverses propostes que es van rebre des del servei de suport d'Altera es va pensar que si el sistema no funcionava com es podia esperar amb la utilització del mòdul *HIL*, s'havia de comprovar que realment es capturava una imatge i es configurava el sensor d'imatge amb els paràmetres desitjats.

Una opció era la d'enviar la imatge via el port sèrie del PC, però això tenia l'inconvenient que s'havia d'implementar un enviament de les dades via sèrie a un adaptador de nivells de tensió per a complir l'estàndard RS-232, i això suposava afegir Hardware extern i un mòdul que envies les dades que s'hauria d'haver implementat en VHDL.

Una altra opció hagués sigut la de capturar els valors mitjançant el SignalTab i exportar-los a un fitxer d'Excel o de text, però això tenia l'inconvenient que el SignalTab utilitza la pròpia FPGA per emmagatzemar els valors i després enviar-los al PC per mitjà del port JTAG. A més a més el tamany d'aquesta memòria no es suficient com per capturar una imatge complerta, per tant s'hauria d'haver realitzat bastants "subcaptures" per aconseguir una captura complerta, amb la consegüent variació dels resultats, doncs no es fan mai dues captures iguals, si molt similars però mai idèntiques, i això podria haver estat una font d'errors.

Com a solució adoptada definitivament és va optar per utilitzar un analitzador lògic, doncs es disposa d'un a les dependències de Cephis. Aquest analitzador lògic es connecta per mitjà d'un port USB al nostre PC. Un cop s'ha instal·lat el dispositiu i el

software de captura de dades que incorpora, hem de fer la interconnexió de les sortides de dades de la FPGA cap a l'analitzador i assignar els ports que hem utilitzat per enregistrar els senyals al PC.

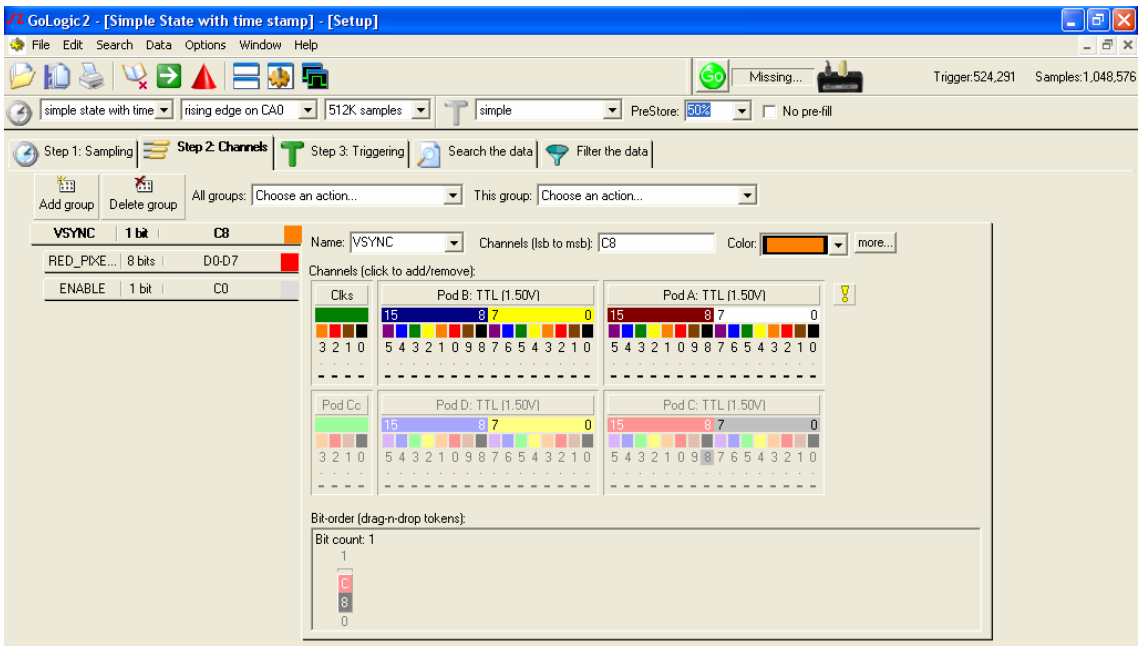


Figura 51.Vista general del programa de l'analitzador lògic

A la imatge anterior es pot veure l'aspecte del programa i l'assignació dels ports de captura amb els de dades. El primer que s'ha de dir d'aquest analitzador és que pot fer el sampling a freqüències predefinides pel software, o es pot utilitzar un rellotge extern per fer que la sincronització amb les dades sigui la adient. Aquesta ha sigut l'opció que s'ha fet servir. Com que hem fet que el sensor vagi contínuament realitzant captures d'imatge, al fer l'adquisició de les dades ens trobarem amb dades que pertanyen a una imatge incompleta doncs seria molta casualitat just començar a adquirir al començar la imatge. Vist això el que s'ha de fer és un cop ens arribin les dades acotar les dades a exportar, només agafarem les dades que es trobin entre pols i pols del senyal VSYNC, doncs recordem que aquest senyal fa un pols just abans de que comenci una nova imatge. Aquestes dades es capturaran com es mostra a la següent imatge:

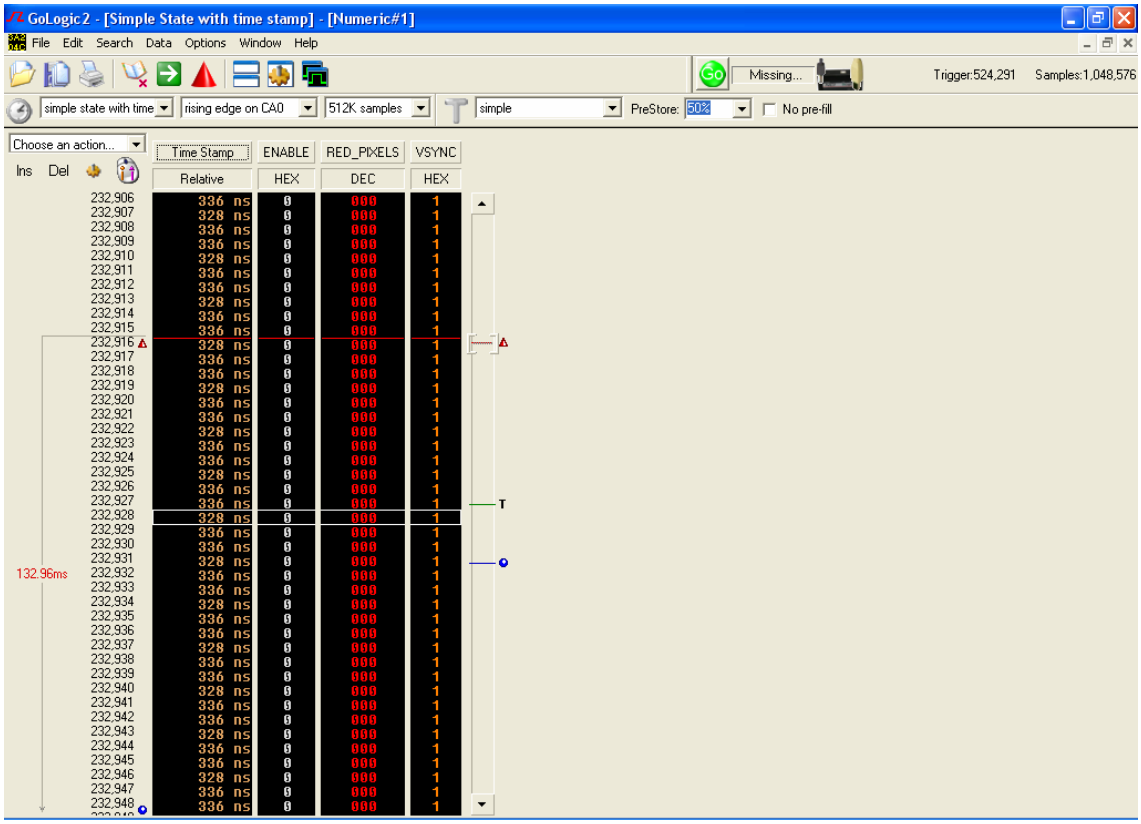
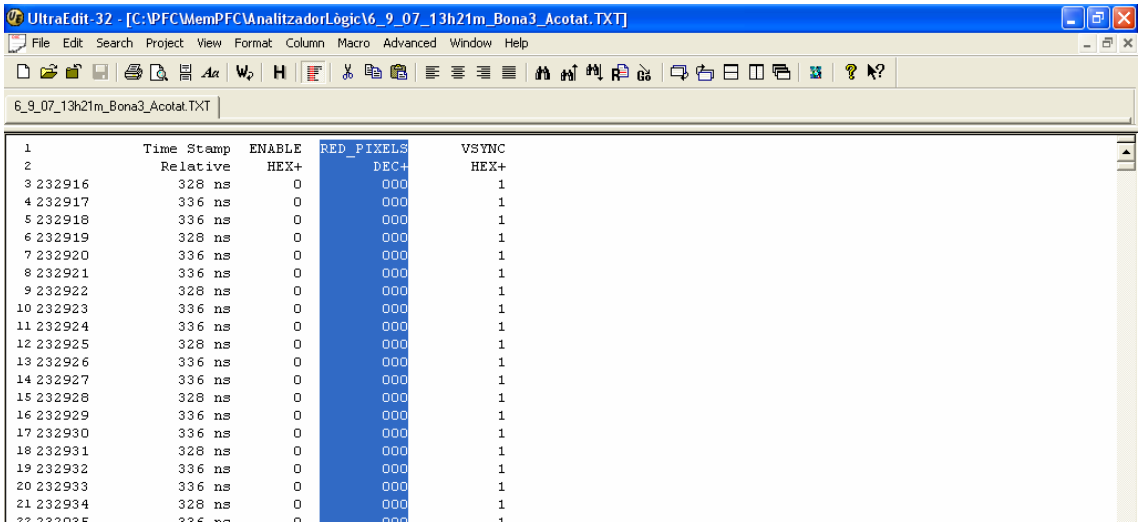


Figura 52.Vista de les dades registrades per l'analitzador lògic

Per exportar les dades tenim vàries opcions, per exemple ho es pot fer a dos tipus de fitxers, un de tipus Excel (.csv), o un de tipus text. Aquí si escollim l'opció d'exportar-ho cap a l'Excel veiem que com a molt es pot tenir 65536 files i l'exportació es fa en una sola columna, per tant donat al nostre volum de dades , que correspon a 76800 dades bones més les que no ho són, superem aquest límit de l'Excel. Per tant no ens queda altra opció que fer la exportació cap a un fitxer de text. Un cop hem escollit el tipus de fitxer ens resta per dir quin interval es vol exportar, que pot ser tot, només el que es veu a la pantalla o un interval delimitat per l'usuari amb un parell de marcadors, donat que el programa té fins a tres parelles de marcadors. Un cop creat el fitxer hem d'agafar la columna corresponent a cada port per després assignar-la a una variable de *Matlab* i així poder convertir les dades en una imatge. Per escollir només la columna necessitarem d'un programa d'edició de text que sigui potent. El software escollit ha sigut el mateix que s'ha utilitzat per la edició dels fitxers VHDL o de *Matlab* i és el UltraEdit de la casa IDM Computer Solutions, doncs ens permet agafar només el text que estigui acotat en unes coordenades determinades, la qual cosa ens anirà molt bé per crear la nostre variable per passar-li al *Matlab* i que aquest composi la imatge.



	Time Stamp Relative	ENABLE HEX+	RED_PIXELS DEC+	VSYNC HEX+
3	232916 328 ns	0	000	1
4	232917 336 ns	0	000	1
5	232918 336 ns	0	000	1
6	232919 328 ns	0	000	1
7	232920 336 ns	0	000	1
8	232921 336 ns	0	000	1
9	232922 328 ns	0	000	1
10	232923 336 ns	0	000	1
11	232924 336 ns	0	000	1
12	232925 328 ns	0	000	1
13	232926 336 ns	0	000	1
14	232927 336 ns	0	000	1
15	232928 328 ns	0	000	1
16	232929 336 ns	0	000	1
17	232930 336 ns	0	000	1
18	232931 328 ns	0	000	1
19	232932 336 ns	0	000	1
20	232933 336 ns	0	000	1
21	232934 328 ns	0	000	1

Figura 53.Vista de les dades exportades a un fitxer de text

En la imatge anterior es pot observar la selecció de les columnes desitjades, corresponent en aquest cas a la variable de l'analitzador lògic RED_PIXELS, i que l'enviarem cap a *Matlab*.

Farem el mateix amb el senyal d'enable de les dades de sortida. Un cop fet això executarem el mateix codi que teníem per la StopFunction del mòdul *HIL*, que recordem que el que fa es transformar la variable que és un array d'una dimensió a un de dues amb el tamany de la nostre imatge. També s'encarrega de formar aquesta matriu amb els valors vàlids, és a dir els que es corresponen amb l'activació del senyal d'enable. Un cop executat aquest programa per crear la imatge el *Matlab* ens mostra la següent imatge:

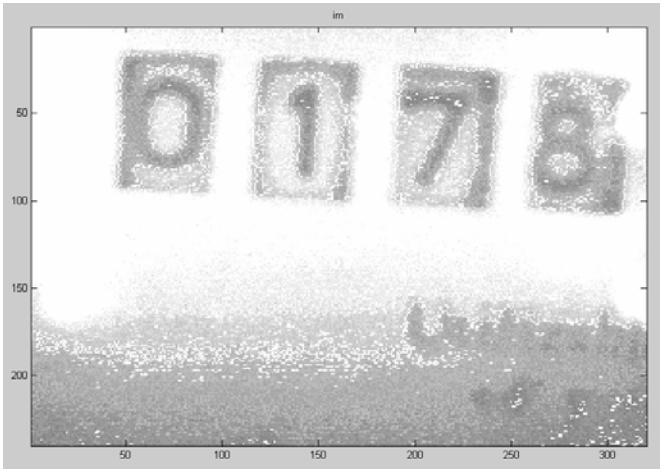


Figura 54.Imatge composada pel Matlab de les dades de l'analitzador lògic

Capítol 5.
Conclusions.

5.1. Conclusions.

Un cop s'ha realitzat tot el projecte es pot fer una sèrie de valoracions respecte la idea que es tenia abans de la realització del mateix.

La primera és que la toolbox *DSPBuilder* de *Matlab*/*Simulink* és una interfície de prototipat ràpid, sobretot per els kits de desenvolupament que es troben a les llibreries del software. A la vegada també és molt útil per fer tractament de dades amb els blocs propis de *Matlab*, donat que també ens permet importar codi hardware que ja tinguem fet per utilitzar-lo en el disseny final, però que té una sèrie de limitacions importants com són la no acceptació per part del *HDL Import* de pins bidireccionals, que els necessitem per el mòdul de comunicació I2C. Una altra limitació es que en el mòdul de Hardware In the Loop no es pot tenir més d'un senyal de rellotge tant d'entrada com de sortida i això junt amb el fet que la seva execució no és a temps real han fet que el projecte no s'ha pogut realitzar com en un principi es va pensar.

Cal dir però que al final s'ha trobat una manera per poder comprovar que el nostre disseny realment funciona, tot i que és una mica enrevessat donat que primer hem de capturar per mitja d'un analitzador lògic una llarga sèrie de dades, després seleccionar manualment l'interval que pertany a la nostre imatge i després amb les eines que *Matlab* disposa composar una imatge.

S'ha esmentar però que s'han trobat més problemes dels previstos inicialment, com acabem de veure. Altera Mysupport ajuda, però a vegades no solen trobar solucions pels temes que no tracten habitualment, així pels dubtes que han sorgit amb dissenys un xic més complexos dels habituals no solen tenir resposta o la resposta és que no es pot realitzar. De fet això es comenta pel fet que quan s'utilitzava la versió 5.1 de *DSPBuilder* i no es feia servir dels kits de desenvolupament que hi havia a les llibreries, l'assignació de Pins no es feia be, com ja s'ha comentat a l'apartat corresponent al *DSPBuilder*, i això era un error que no es va trobar a cap lloc que es parlés del mateix. A més a més es va trigar un temps fins a obtenir una resposta per part del servei de suport que ens expliqués el que estava passant i com es solucionava. Un cop es va canviar a la versió 6.1 es va comprovar com el problema estava pràcticament solucionat, a excepció de les entrades, que hi continua havent algunes que no les fa correctament. Per tant el que s'ha hagut de fer és retocar aquests petits errors amb l'ajuda de *QuartusII*.

En resum es pot dir que tot i aquestes dificultats cal extreure'n una valoració positiva del projecte en global, ja que les dificultats han ajudat a aprendre encara més sobre tot plegat, i a buscar camins per intentar trobar solució als diversos problemes sorgits, tot i que al final el projecte no s'ha pogut fer amb l'eina Hardware In the Loop.

5.2. Experiència personal i professional.

La realització d'aquest projecte m'ha suposat un coneixement d'altres eines per a la síntesis d'un codi hardware sobre una FPGA que no sigui amb el *QuartusII*. En aquest cas les eines eren el *DSPBuilder* sota l'entorn *Simulink* de *Matlab*.

Amb el *DSPBuilder* he après a realitzar dissenys d'una manera relativament ràpida, sobretot si s'utilitza un dels kits de desenvolupament que s'inclouen a les llibreries. També he pogut aprendre que el software no sempre ofereix els resultats desitjats, com per exemple en el problema de la versió 5.1 del programa on l'assignació de pins d'entrada sortida no es realitzava correctament, o com la limitació dels ports

bidireccionals, o inclòs el tema de la simulació Hardware In the Loop. Però tot això ens ha fet buscar altres vies per comprovar la validesa del disseny, com per exemple amb el SignalTab d'Altera o l'analitzador lògic, o inclòs un oscil·loscopi per veure que les dades sortien o els senyals de sincronisme.

També cal ressaltar que tot i que el *QuartusII* és una eina que per a mi era coneguda des d'abans de la realització del projecte he aprofundit una mica més en les opcions que ens ofereixen les eines d'aquest entorn.

En addició al que s'acaba d'esmentar s'ha de reconèixer que tot i que el *Matlab* no l'hem fet servir com s'havia previst en un principi, si que ha estat vital per la conversió de les dades que el numèriques a una imatge i per tant realitzar la comprovació visual del resultat final.

Per últim he pogut veure les dificultats d'haver de realitzar la configuració i la gestió de les dades del sensor, i que no és una cosa tan trivial com es podria pensar en un principi, però tot i així s'ha d'extreure una conclusió global bona, doncs amb les modificacions suggerides en el pròxim apartat segurament es podria dur a terme el projecte com inicialment s'havia plantejat.

5.3. Evolució futura.

Com el projecte no s'ha pogut realitzar com en un principi es pensava, està clar que la línia de futur apunta directament a trobar possibles solucions per a poder dur-lo a terme tal i com ens havíem proposat. Per fer això s'hauria de refer el hardware del sistema MiraKonta:

- El primer que faríem seria introduir un sistema que gestionés el sensor, incloent la seva configuració i rebuda de les dades, i les emmagatzemi, és a dir, hauria de

comportar-se com un buffer. Aquest buffer que gestiona el sensor podria ser perfectament una altra FPGA.

- El següent pas a realitzar és el de fer que el senyal de rellotge del sensor vingui d'un sistema que no sigui la FPGA que contingui el mòdul *HIL*, per exemple podria provenir de l'altra FPGA, la que ens fa la part de gestió del sensor, també podria provenir d'un oscil·lador extern o de qualsevol altre dispositiu que ens donés un senyal de rellotge que compleixi les especificacions del sensor d'imatge.

El fet d'incloure una altra FPGA ens donaria pas a un sistema que es comportaria globalment asíncron localment síncron, ja que les dades del sensor i la seva configuració es faria complint els timings que ens requereix el sensor d'imatge i un cop posades les dades en el buffer anar-les llegint al ritme que vagi marcant l'execució del mòdul *HIL*. Com a element de buffer es podria fer ús d'una Megafuntion d'Altera que crea una Dual Port RAM. El principal avantatge de fer servir una DPRAM és el fet que és un dispositiu que va molt be per enllaçar dos sistemes que van cadascun a la seva velocitat. D'aquesta manera el hardware del sistema quedaria de la següent manera.

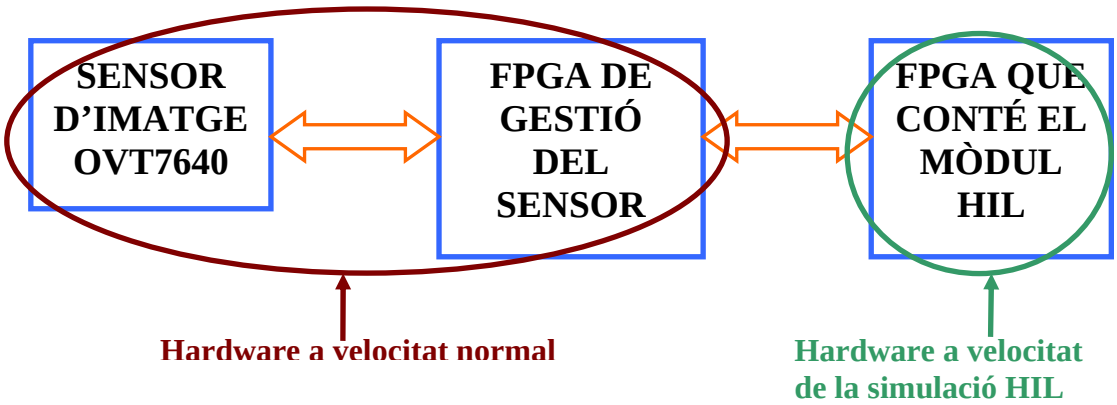


Figura 55.Diagrama de blocs de la possible solució per el mòdul *HIL*

Si fessin aquestes modificacions al hardware del sistema MiraKonta, podríem llavors realitzar la caracterització dels comptadors de cabal d'aigua. Inclòs d'aquesta manera elaborant un programa amb *Matlab* que anés ajustant els paràmetres de configuració per

a que la imatge resultant fos òptima. D'aquesta manera el *Matlab* ens seria una eina molt potent per fer una automatització del procés.

Referències i Bibliografia.

- ***ModelSim***. Software de simulació de codi HDL de la casa Mentor Graphics. Adreça URL:

  http://www.model.com/resources/resources_manuals.asp

- **Altera**. Fabricant de FPGA, CPLD i software per la síntesi. Adreça URL:

  www.altera.com

  - https://mysupport.altera.com
  - http://www.altera.com/support/ip/dsp/ips-dsp-builder.html(*DSPBuilder* User Guide)
  - http://www.altera.com/support/ip/dsp/ips-dsp-ip-cores.html (IP Cores User Guides)

- **Google**. Multicercador d'informació paramètrica. Adreça URL:

  http://www.google.es

- **Datasheet catalog.** Recerca de *datasheets* de components electrònics. Adreça URL:

  http://www.datasheetcatalog.com

- ***Matlab*.** Software de simulació i de càlcul numèric. Adreça URL:

  http://www.mathworks.com

- **VHDL Online.** Tutorial de VHDL Online. Adreça URL:

  www.vhdl-online.de/tutorial/

- **Omnivision**. Fabricant de sensors d'imatge. Adreça URL:

  http://www.ovt.com/

- **Wikipedia**. Enciclopedia lliure. Adreça Lliure

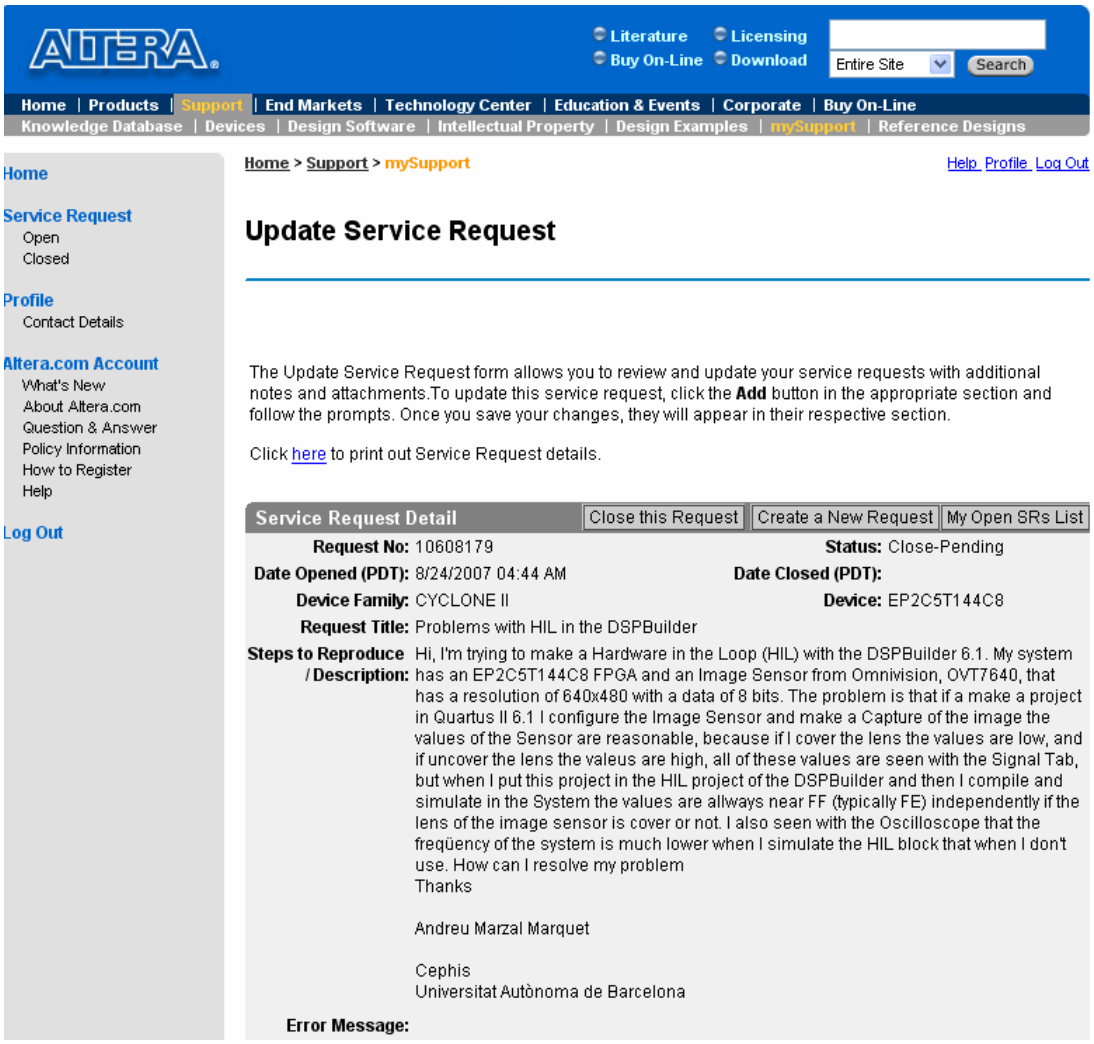
  http://es.wikipedia.org/wiki/Portada

- **VHDL FORUM**. Foro VHDL. Adreça URL:

  hhttp://www.tek-tips.com/threadminder.cfm?pid=284

Annexos.

Annex 1. Confirmació per part d'Altera la impossibilitat de la realització del mòdul *HIL* amb el disseny actual



ALTERA

Literature Licensing
Buy On-Line Download
Entire Site Search

Home | Products | Support | End Markets | Technology Center | Education & Events | Corporate | Buy On-Line
Knowledge Database | Devices | Design Software | Intellectual Property | Design Examples | mySupport | Reference Designs

Home
Service Request
Open
Closed
Profile
Contact Details
Altera.com Account
What's New
About Altera.com
Question & Answer
Policy Information
How to Register
Help
Log Out

Home > Support > mySupport — Help Profile Log Out

Update Service Request

The Update Service Request form allows you to review and update your service requests with additional notes and attachments.To update this service request, click the **Add** button in the appropriate section and follow the prompts. Once you save your changes, they will appear in their respective section.

Click here to print out Service Request details.

Service Request Detail — Close this Request | Create a New Request | My Open SRs List

Request No: 10608179 — **Status:** Close-Pending
Date Opened (PDT): 8/24/2007 04:44 AM — **Date Closed (PDT):**
Device Family: CYCLONE II — **Device:** EP2C5T144C8
Request Title: Problems with HIL in the DSPBuilder
Steps to Reproduce / Description: Hi, I'm trying to make a Hardware in the Loop (HIL) with the DSPBuilder 6.1. My system has an EP2C5T144C8 FPGA and an Image Sensor from Omnivision, OVT7640, that has a resolution of 640x480 with a data of 8 bits. The problem is that if a make a project in Quartus II 6.1 I configure the Image Sensor and make a Capture of the image the values of the Sensor are reasonable, because if I cover the lens the values are low, and if uncover the lens the valeus are high, all of these values are seen with the Signal Tab, but when I put this project in the HIL project of the DSPBuilder and then I compile and simulate in the System the values are allways near FF (typically FE) independently if the lens of the image sensor is cover or not. I also seen with the Oscilloscope that the freqüency of the system is much lower when I simulate the HIL block that when I don't use. How can I resolve my problem
Thanks

Andreu Marzal Marquet

Cephis
Universitat Autònoma de Barcelona

Error Message:

Updates	1 - 7 of 7	Add an Update | Previou | Ne
Date Created (PDT)	**Type**	**Note**
8/30/2007 02:24 AM	To Customer	Hi Andreu, About your application, HIL would really be a good approach. However, as you know, the multi-clock domain design cannot be supported in HIL. In your design, besides the clock used for FPGA, you have to output synclk to camera. So, it's hard to remove this clock signals. If you want to store the data during debugging, you might try other methods. If there is other output port on your target board such as serial port, you can output the data to PC via this port and then store them by some existing system tools. Best Regards, Kevin
8/29/2007 11:29 AM	From Customer	Hi Kevin, My design originaly had two clocks, but now I only use one "CLK", the other "OVT_PCLK" is just the same as "CLK" but X times slower (X can be programed to the Image Sensor), so I decide to make all the project with just one clock and a counter of X values, so the sistem is working good in quartus. Another thing is that I take fotographs no video. Well my problem is that I need the HIL block to recive the hole image and with the data through a program in matlab I compose the Image, and if the HIL module works I can change parametres of the Image sensor, such as Gain, Exposure, saturation.., and I have to make with DSPBuilder because this is objective of my project.
		Hi Andreu, I have checked the latest project you uploaded. I found that there were more than one clock signals in the project. However, HIL block cannot support multi clocks. You can find more details about the HIL requirement in the section of "HIL" in DSP Builder Reference Manual.

	8/28/2007 08:10 PM	To Customer	I have another suggestion. HIL is only used for hardware acceleration in Simulink. However, since you need to input the video into your target board and download your design to the board, you can debug your design just with QuartusII and your target board. That's, from your description and your application, you don't have to use HIL simulation. You can debug in your target board directly. Best Regards, Kevin
	8/28/2007 05:19 AM	To Customer	Hi Andreu, Thanks for uploading your project which is helpful to solve this problem. I have checked the project you uploaded. Are you using the "HDL Import" in DSP Builder to import the project of "TestAplicacionEsclavo"? However, I tested the project of "TestAplicacionEsclavo" and it cannot pass the compilation. The signal of "sGain" is assigning by different value in different process. So, please make sure your original project can pass compilation. If you want to test the project of "TestAplicacionEsclavo", you can import it directly to "HIL" block. And, before you download the project, please check if there is signal at the pins which is connected to your sensor. Best Regards, Kevin
	8/28/2007 02:29 AM	From Customer	First of all I have to apologize, because I made a mistake and I made an attachment of a wrong design, know I'm going to upload a correct project. I'm going to do what you comment about importing directly "TestAplicacionEsclavo" to HIL bloc. As I told you in other messages I connect an Oscilloscope to the Image Sensor and the signals are correct, but the diferenence between using the HIL or not, is that when I don't use the HIL the timing are OK, but when I use the HIL block the frequency of the sistem falls. An other problem I have is that in the datasheet of the DSPBuilder comments that when you Import an HDL code and this code has 2 clocks one is set like main clock an the other is shown as an Input, but when I do this no one is shown like an input.

		Thanks The new project is named 28_08_07
8/27/2007 05:33 AM	From Customer	Hi, I forgot to tell you that my signals/pins are exported. I don't have a global reset of the system, I only generate one in my VHDL code to initialize all the FPGA, but not to the sensor. The problem is that if I use an Oscilloscope or the Signal Tab in Quartus, no in DSPBuilder, I can see the signals OK, in the case of using an oscilloscope the frequency of all the system goes down using the HIL block, but works. The main problem is that without HIL the system seems to works more o less correctly, but with the HIL block the result of the simulation is at I comment in my first message I also add the project if you want to take a look
8/27/2007 02:19 AM	To Customer	Hi Andreu, Thanks for using mySupport. I have checked your issue. To solve your problem, please go through the following steps. 1. please help make sure that the pins of your FPGA device connecting to your sensor are exported by selecting "export" in the control panel of HIL; 2. Please help check if there is global reset signal used in your design. If so, please simply add a clock block to the original MDL and set "aclr" to "Active High", as opposite to its default value. Recompile both the orginal version and HIL revision. Then the simulation will work as expected. I will set this SR to "close-pending" for time being. When the latest version is released, I will let you know as soon as possible. Best Regards, Kevin

Attachments 1 - 2 of 2 Add an Attachment Previou Ne

Date Attached (PDT)	Attachment Name	Attachment Type
8/28/2007 02:34:01 AM	28_08_07	rar
8/27/2007 05:41:06 AM	23_08_07	rar