# Desenvolupament d'un processador SISA en VHDL

Joel Criado Ledesma

Mario Rodríguez Pérez

Professor: Josep Llorenç Cruz Díaz

Dept. Arquitectura de Computadors

Primavera 2017-18

# Contingut

# 1 Introducció

L'objectiu del treball ha estat la realització d'un System on Chip (SoC) fent servir una FPGA integrada en una placa de desenvolupament. Concretament, es tractava d'implementar en un llenguatge de descripció hardware (VHDL en el nostre cas) un processador i la resta de components que completen el SoC.
El processador a implementar ha estat el SISA amb suport per a Interrupcions, Excepcions, Mode Sistema, Memòria Virtual i sense Unitat FP. El SoC consta d'aquest processador, d'un controlador I/O, una memòria SRAM de 64KB, suport per a teclat PS/2 i per a VGA.
Aquest treball s'emmarca dins l'assignatura Projecte d'Enginyeria de Computadors (PEC) de la Facultat d'Informàtica de Barcelona (FIB), així que és lògic pensar que té un alt component formatiu. Concretament, amb aquest treball esperàvem aprendre a programar elements hardware mitjançant un llenguatge de descripció com VHDL, posar a prova i millorar els nostres coneixements de la matèria HW implementat tot el SoC, usar les eines i familiaritzar-nos amb les plaques de desenvolupament per FPGAs i millorar el treball en grup i la planificació i disseny d'un projecte.
Per a implementar tot això, hem dividit el treball en diverses etapes, començant amb la implementació d'un processador unicicle senzill fins a acabar amb un processador multicicle que suporta tots els elements que s'acaben de descriure. A continuació trobareu la descripció de totes aquestes etapes.

# 2 Entorn de Treball

Per a la realització d'aquest projecte s'ha consultat activament la documentació de cada etapa, situada a la pestanya "Proyecto AC" de la pàgina de PEC, la documentació dels processadors SISA-F i SISA-S, així com les transparències Resum VHDL Bàsic.
El projecte ha estat desenvolupat amb el software Quartus II Web Edition 13.0 amb l'extensió SignalTap II. També hem fet servir ModelSim-Altera Starter Edition 10.1 per a tasques de debug; tot el programari és del fabricant Altera.
El hardware seleccionat pel projecte ha estat la placa Terasic DE1 amb la FPGA Cyclone II EP2C20F484C7.

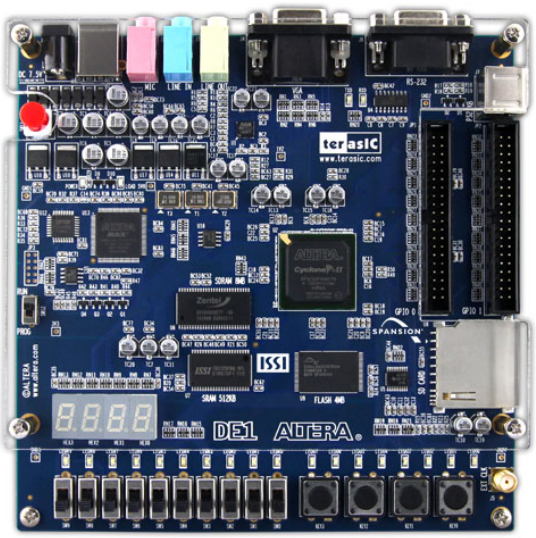


Figure 1: DE1

# 3 Etapes del Projecte

## 3.1 Etapa 1: Proccessador Base

Hem començat el treball implementant un processador molt senzill. Es tracta d'un processador unicicle capaç d'executar les instruccions MOVI, MOVHI i HALT de l'arquitectura SISA.
En aquesta etapa, les decisions preses han estat nul·les. Hem anat seguint la documentació pas a pas per realitzar el disseny, creant els diferents components tal com s'explicava. Així doncs, hem creat els mòduls control_l (genera els senyals de control), unidad_control (conté control_l i gestiona el PC), regfile (banc de registres, 8 registres de 16 bits), alu (realitza les operacions en funció de la lògica de control), datapath (connecta l'alu amb el banc de registres) i el proc (mòdul principal, connecta la unitat de control amb el datapath). Com a memòria d'instruccions hem fet servir una memòria ROM proporcionada pel professor.
L'esquema implementat és el següent:

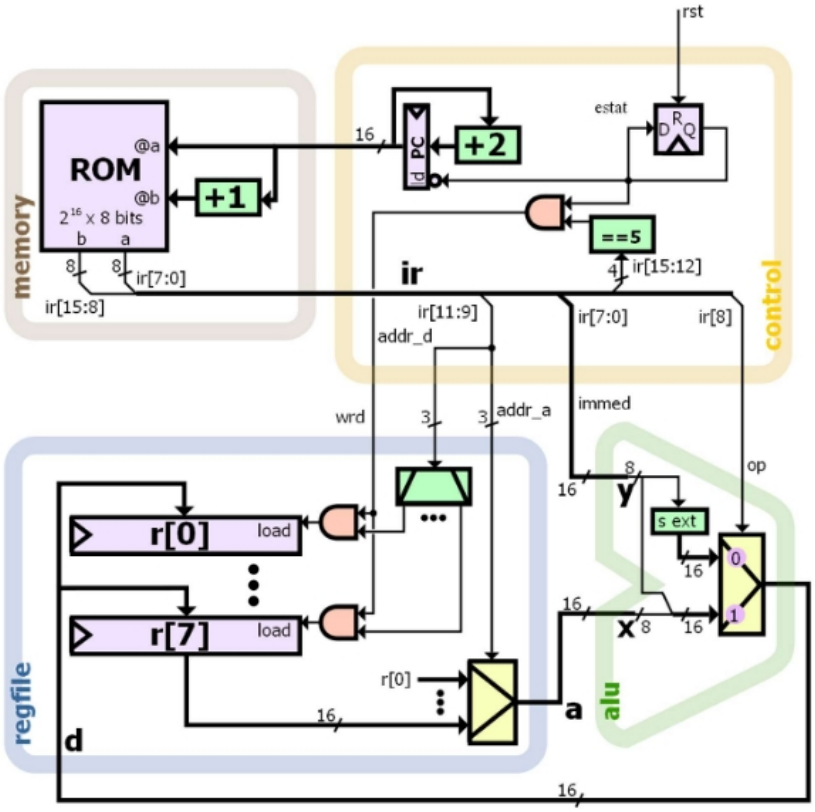


Figure 2: Processador Unicicle

## 3.2 Etapa 2.1: Processador multicicle

L'objectiu d'aquesta etapa era implementar els accessos a memòria, és a dir, les instruccions LD, LDB, ST i STB i per a facilitar la tasca s'ha dividit el processador en dos cicles (ara és multicicle): fetch d'instrucció i demw (decode, execution, memory i write).
Hem creat el nou mòdul, anomenat multi, que s'encarrega de gestionar en quin cicle es troba el processador i modifica els senyals de control en funció d'això.
Amb aquest canvi fet, hem afegit la resta de la lògica necessària per a fer els ld/st, així com el càlcul de direcció a la qual es vol accedir, que es troba dins l'alu. En aquestes instruccions, és necessari multiplicar l'inmediat x2 (tal com diu la documentació de l'arquitectura SISA) i en el nostre cas hem decidit fer-ho en el datapath, en funció del senyal immed_x2.

## 3.3 Etapa 2.2: Controlador de Memòria

En aquesta etapa havíem d'implementar el controlador de memòria i donat que aquest havia de fer write i read hem repartit la feina en dos subetapes, una per cada operació.
Per fer aquesta etapa hem fet servir de guia la documentació aportada per l'assignatura i el datasheet de la memòria que ve incorporada a la placa (Esmentada a l'apartat de l'entorn).
En la primera subetapa hem decidit agafar el primer mètode que oferia el fabricant per fer les lectures, aquest no deixa mai la memòria en baix consum i sempre fa lectures encara que no serveixi per res. Hem decidit fer-ho així perquè d'aquesta manera disminuïa la dificultat del codi a implementar de la subetapa, i donat que aquest curs no se centrava en el rendiment hem pensat que seria la millor opció.
A la segona subetapa, o la de write i read, el fabricant dona quatre opcions d'escriptura i hem escollit la segona opció, ja que és la que sembla més fàcil d'implementar.
Hem definit aquestes etapes per separar les dues funcionalitats i anar provant d'una en una. Una vegada comprovat que les lectures funcionan correctament hem incorporat les escriptures.
Amb el read i write escollit, hem hagut d'implementar els accessos a memòria amb 2 cicles. Donat que els reads i el primer cicle del write comparteixen molts senyals aquests dos comparteixen el primer cicle (e0).

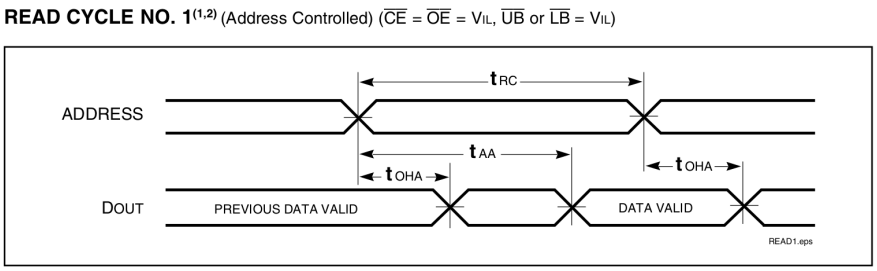


Figure 3: Read

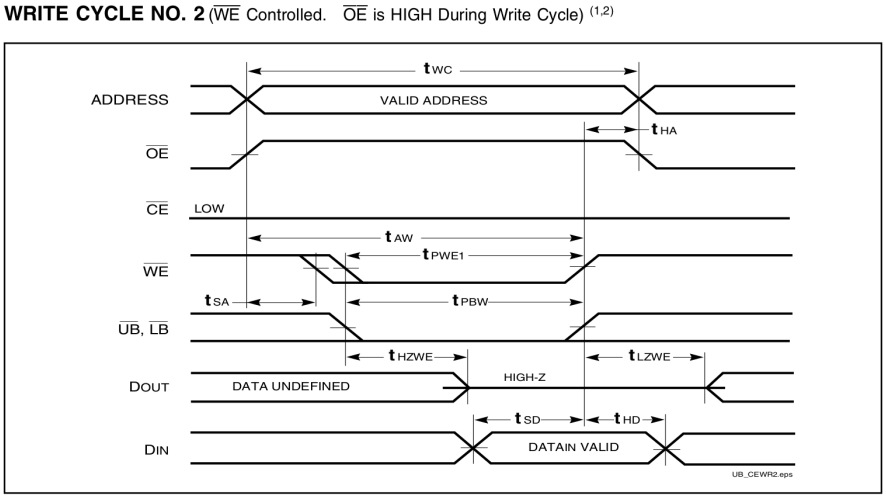


Figure 4: Write

## 3.4 Etapa 3: ALU (Arithmetic Logic Unit)

L'alu és l'element del processador encarregat de realitzar la majoria d'operacions que es duen a terme. Cap processador està complet sense una i fins aquest moment, el component anomenat "alu" no és representatiu d'una alu real.
En aquesta etapa es modifiquen els diferents components per donar suport a les instruccions ADD, ADDI, SUB, OR, XOR, AND, NOT, SHA, SHL, CMPLT, CMPLE, CMPEQ, CMPLTU, CMPLEU, MUL, MULH, MULHU, DIV i DIVU.
Per a realitzar això, hem canviat la mida del senyal de control "op" a 7 bits, que codificava unes poques operacions fins ara, fent que a l'alu arribin els bits necessaris per seleccionar l'operació a realitzar. S'ha tingut en compte que el mòdul ha de seguir calculant la direcció efectiva dels accessos a memòria, com fins ara.
Dins l'alu, hem decidit realitzar totes les operacions en paral·lel i seleccionar el senyal adequat al final, amb la intenció d'estalviar temps. Hem fet aquesta selecció en dues etapes per simplicitat; en la primera se selecciona la sortida d'un dels blocs (operacions aritmètiques, adreça de memòria, etc.) i en la segona seleccionem quin bloc és el correcte. La sortida z simplement és una OR negada de tots els bits de la sortida de l'alu.

## 3.5 Etapa 4: Instruccions de Salt

Partim aquesta etapa en tres subetapes, agrupant les instruccions per opcodes i semàntica semblant.
La primera subetapa està formada per les instruccions BZ i BNZ, ja que tenen un comportament semblant. Hem afegit tota la lògica a la ALU per generar el bit Z i hem complementat el selector de la ALU amb les noves instruccions. Després d'això hem provat els salts amb els tests facilitats per la documentació i algun que ens va deixar alguns companys. També per les següents subetapes hem engreixat el opcode que portàvem a la ALU donat que necessitàvem diferenciar bé totes les possibles instruccions, a més a més hem pensat que seria més fàcil per futures ampliacions si ja portem el maxim de bits que diferencien una instruccio ( 4 op code + 6 function ). També hem afegit un senyal de control "tknbr" que és el que s'encarrega de seleccionar el que s'ha d'assignar al pc. Aquest senyal a l'entrega es genera al datapath encara que no té molt de sentit donat que seria lògic que és generes al control_l.
La segona subetapa composta per tota la família de jumps, menys el jal i el calls. Aquests comparteixen comportament donat que incrementen el pc, el guarden i assignen el contingut del "registre a" al pc, per aquest motiu hem portat el pc des de la unitat de control fins al datapath.
La tercera subetapa és la del jal i calls, encara que del calls només hem provat el seu correcte funcionament dintre de ALU i el datapath. Aquest que no es pot provar bé ara mateix, ja que és de la part de "Mode Sistema".

## 3.6 Etapa 5: Entrada i Sortida

Amb aquesta ampliació del processador busquem poder integrar diversos dispositius externs en el SoC, com per exemple un ratolí o un teclat. Per a donar suport a això, és necessari crear un mòdul similar al banc de registres, però en aquest cas amb 256 registres de 16 bits, i implementar les instruccions IN i OUT.
El mòdul afegit (controladores_IO) en aquest punt està implementat de la mateixa manera que el banc de registres, amb escriptures síncrones i lectures constants. Per a poder fer servir els leds i visors hexadecimals, els hem mapejat directament a un dels registres d'aquest nou banc, així que escrivint al registre adequat es poden activar els elements (5 leds verds, 6 leds vermells, 10 visors hexa).
Pel que fa a la lògica de control, el canvi més significatiu ha estat per donar suport a immediats de 8 bits, ja que les instruccions IN/OUT són les úniques que els fan servir de tot el processador.

## 3.7 Etapa 6: Controlador de teclat PS/2 i vídeo VGA

Aquesta etapa va consistir a seguir la documentació de l'assignatura i adaptar el mòdul que ens va proveir l'assignatura, al nostre codi. El més complicat d'aquesta etapa va ser fer totes les connexions i generar bé la lògica de control. Per això també hem adaptat el controlador de memòria donat per redirigir els senyals al VGAController.
Hem decidit mapejar la memòria, com deia la Documentació, a la posició de memòria 0xA000 (0x5000 física).

## 3.8 Etapa 7.1: Interrupcions

Per fer les interrupcions hem decidit fer una controladora que s'encarregui de gestionar totes les interrupcions i que enviï un senyal al control per avisar que hi ha una interrupció esperant a ser atesa.
Hem incorporat aquesta controladora al Controladors_IO donat que tots els mòduls estan en aquest nivell, com el Timer, Pulsadors, Botons i Keyboard. Aquests mòduls envien els senyals a la controladora d'Interupccions i aquesta és l'encarregada de gestionar i apuntar-se quina de les interrupcions esta esperant. Hem posat la controladora a aquest nivell donat que ens simplifica la circuiteria a afegir als nivells superiors. A la controladora d'interrupcions li entra inta que és el senyal que avisa a la controladora quan hi ha una interrupció ja atesa i li surt intr, senyal que s'encarrega d'avisar al processador que hi ha una interrupció a tractar. També hem afegit un tipus de cicle mes al multi, el SYSTEM; aquest és l'encarregat de fer totes les modificacions al hardware per guardar-se el context en el qual ha esdevingut interrupció. A més a més hem creat una altra entitat per als registres de sistema, donat que s'ha de definir un comportament per el cicle de SYSTEM.
Propaguem el senyal de SYSTEM per tots els dispositius que pugin alterant el seu funcionament normal en aquest cicle i filtrem tots els permisos d'escriptura al multi.
Per avisar al banc de registres que és cicle de multi hem portat un senyal de control al system_regfile per fer totes les escriptures a registres privilegiats en un cicle.

## 3.9 Etapa 7.2: Excepcions

A aquesta etapa hem dubtat entre dues maneres d'implementar aquesta funcionalitat: -La primera consisteix a centralitzar la detecció d'excepcions, aquesta manera obliga posar la controladora en un lloc estratègic per on passin totes els senyals necessaris per la detecció de les excepcions, a més a més és necessari que li entrin fins i tot els valors que agafen els operands que entren a la ALU.
-La segona manera consistia a descentralitzar la detecció d'excepcions, fent que la controladora només detectes amb els senyals que li envien els mòduls les excepcions i generes tota la lògica necessària per executar la rutina si fos necessari. Aquest mètode ens obligava a enviar tots els senyals d'excepcions on es generaven, cosa que feia engreixar tota la circuiteria donat que moltes de les excepcions es generen o bé fora del proc o fora de la logica de control. Hem decidit que era més clar per nosaltres fer la segona opció, encara que engreixes una mica les capes superiors amb els senyals de detecció d'excepció. Aquesta manera facilita la comprensió al lector, ja que veu els senyals on es produeixen. Tot i això algunes exepcions com la de direccio il·legal, que es produeixen fora del proc, es generen en el mateix Exception Controller .
A més a més hem d'incorporar les interrupcions al codi de la controladora donat que aquesta haurà de passar un codi d'excepció al system_regfile.

## 3.10 Etapa 7.3: Mode privilegiat i Crides a Sistema

Aquesta etapa és una de les més importants del processador, per aquest motiu hem partit aquesta etapa en tres subtetapes.
-La primera etapa va consistir a fer tot el control per gestionar el fmode sistema i passar la paraula de control al control_l per subetapes posteriors. Aquesta lògica consistia a fer els canvis adients per passar d'un mode a l'altre i darrerament comprovar la seva correcta execució amb la placa.
-La segona etapa va ser la del calls, encara que la seva lògica ja estava feta hem retocat la part d'excepcions perquè ho detectes com una excepció més i saltes al mode sistema.
-L'última etapa consistia a fer tota la part d'excepcions relacionades amb el Mode Sistema, a part de retocar excepcions passades hem afegit encara més lògica per detectar les noves. Moltes d'aquestes

excepcions es generen al control_l donat que a l'etapa d'excepcions hem escollit descentralitzar part de la detecció d'excepcions de la seva controladora.
Encara que la documentació de l'assignatura ens recomanava posar les instruccions de IN,OUT, com a operacions no privilegiades, per facilitat els jocs de prova, hem escollit fer cas de l'especificació del SISA-F que sí que les marcava com a privilegiades. Hem deixat que el HALT es pogués executar com a usuari normal.

## 3.11 Etapa 8: TLB (Translation Lookaside Buffer)

El TLB és l'últim element que afegim al processador i el que el completa, donant suport a la Memòria Virtual i a la Paginació, un dels elements més importants per a poder córrer un SO en el teu processador. També s'afegeixen les instruccions WRPI, WRPD (modifiquen el TAG físic i els bits v i r del TLB corresponent), WRVI, WRVD (modifiquen el TAG virtual del TLB corresponent) i FLUSH (invalida el TLB indicat).

En el nostre cas, hem implementat dos TLBs, un per a instruccions i un altre per a dades. Hem decidit instanciar dos elements separats en el projecte per poder inicialitzar-los amb valors diferents, donant més versatilitat al processador. L'estructura d'un dels mòduls és la següent:

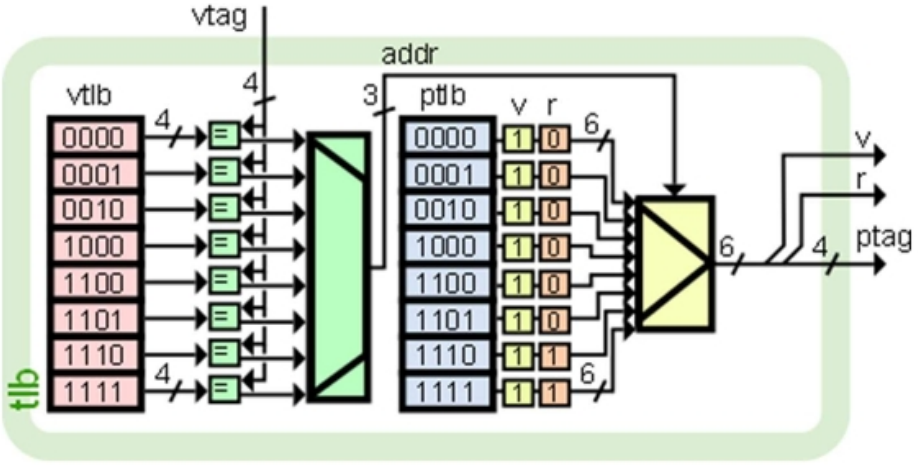


Figure 5: TLB

Com es pot apreciar en el dibuix, les comparacions es realitzen en paral·lel i si no hi ha miss, s'escull l'entrada física que ha generat el hit, donant prioritat a la primera ocurrència si n'hi hagués més d'una. Les pàgines són de 4KB, amb un total de 16 pàgines en el sistema.

També ha estat necessari afegir nova lògica de control per gestionar els nous mòduls i instruccions, així com modificar el controlador d'excepcions per donar cabuda a totes les excepcions del TLB. Com algunes d'elles depenen de l'estat actual del processador, per simplicitat s'ha decidit portar l'estat fins al controlador d'excepcions i fer la comparació allà en comptes de filtrar cada excepció en el mòdul "multi". Per acabar, hem decidit fer el pas a l'estat de sistema des del cicle DEMW només, així que hem hagut de mantenir el valor d'alguns senyals (col·locar registres), ja que algunes excepcions ocorren durant el cicle de FETCH.

# 4 Discussió

Amb el processador ja acabat, podem afirmar que ens ha semblat un treball enriquidor i estem satisfets (en una línia general) amb el treball fet.
Concretament, creiem que la part final del projecte ha estat enriquidora i interessant, ja que les interrupcions i les excepcions són una part molt important dins d'un processador actual i, poder implementar-les amb un cert grau de llibertat, ens ha permès aprofundir i familiaritzar-nos amb el seu disseny. El fet de poder gestionar diferents dispositius i esdeveniments mitjançant interrupcions és una eina molt potent. A banda d'això, també ens quedem amb tot el temps que hem dedicat a debugar el codi, hem pogut aprendre molt sobre un procés al qual, fins ara, havíem dedicat poc temps en el GEI.
D'altra banda, també volem dir que no estem gaire satisfets amb el nostre controlador de memòria. El vam dissenyar tenint al cap la idea keep it simple i això ens ha passat factura al llarg del projecte, havent de refer el controlador a la meitat i amb molt temps perdut per culpa d'això. Hem après que a vegades no tot pot ser tan simple.
Per seguir amb la feina amb aquesta versió del processador tenim diverses opcions, en funció de si volem més funcionalitats o millor rendiment.
L'opció més obvia és la de continuar amb el desenvolupament de la família SISA i ampliar el processador amb la unitat de Floating Point descrita en el processador SISA-F, dotant al processador d'un dels components bàsics d'unt processador convencional.
Una altra alternativa per avançar amb el projecte és aprofitar que el processador és multicicle i segmentar-lo en funció de les etapes actuals, o inclús crear-ne de noves a partir de l'etapa DEMW, que es podria separar en més d'una. Amb això, es podria executar una instrucció a cada cicle del processador (a diferència d'ara) i milloraria el rendiment substancialment.
També, podem notar que la nostra placa té diversos elements que encara no hem explorat, com poden ser la sortida de jack o els GPIOs, així que podríem afegir algun controlador per aquests mòduls en el nostre SoC.
Pel que fa a la memòria, si ens fixem bé estem fent servir només una petita part de la SRAM, així que podríem millorar el disseny substituint-la per la SDRAM de la qual disposa la placa amb la qual estem treballant.
Per acabar, seria interessant analitzar quines parts del processador triguen més i on està el camí crític, amb la intenció de detectar parts innecessàries i/o redundants per millorar el temps d'execució, l'àrea del chip i/o el consum energètic.

# 5 Conclusions

En el transcurs del projecte hem seguit unes pautes de disseny i una metodologia (explicat prèviament) per arribar a obtenir el nostre processador complet. Avui en dia, un processador ha de constar de diverses característiques com suport per I/O, Interrupcions i Excepcions i Memòria Virtual, dels quals el nostre processador disposa. A més a més, pel fet de treballar amb una placa, hem pogut dissenyar no només un processador, sinó un SoC, i afegir suport per a vídeo, teclat i aprofitar altres elements de la placa Terasic DE1.

Hem pogut veure la utilitat de la combinació dels llenguatges de descripció hardware (VHDL, Verilog) amb una FPGA. Poder fer les proves necessàries durant el desenvolupament del chip amb un software facilita molt tot el procès (col·locar les portes lògiques una a una ha de ser una feina bastant tediosa) i permet obtenir un munt de mètriques diferents que ajuden a decidir el següent pas per millorar el processador i a polir el producte final. Això, combinat amb les eines de debug (el nostre gran descobriment en aquest treball!), com SignalTap i ModelSim, agilitzen molt tot el procés i segur que acaben reduint costos.

| *Metrica* | *Valor* |
|---|---|
| *Frequència* | 12.24 MHz |
| *Tamany Chip* | 9.918 e.l |
| *Consum* | 80.38 mW |

Table 1: Metrics Table

Amb les eines que la versió gratuïta de Quartus posa a la nostra disposició hem pogut obtenir les mètriques de la Taula 1. Com es pot esperar, el consum energètic és bastant reduït (un i5 actual té uns 70-80W de consum aproximadament), ja que les funcionalitats del nostre processador són bastant limitades (per exemple, no té Cache) o reduïdes (processador de 16 bits, freqüència bastant baixa).