Memòria del projecte del processador

Projecte d'Enginyeria de Computadors

Marcel Arroyo Pinar
Eduard Benito Peris

1.Índex

2.Introducció

2.1 Propòsit

L'objectiu d'aquesta assignatura és que l'alumne aprengui a desenvolupar un prototip d'un computador o un SoC (System on Chip) en un xip programable sobre una placa base per crear un mini-ordinador. Es posaran en pràctica alguns dels conceptes adquirits en les assignatures anteriors sobre el disseny de la microarquitectura d'un processador, sobre el disseny i implementació de software de sistema, i sobre el disseny de sistemes digitals.

2.2 Importància del projecte per a la formació

En el món de l'Enginyeria Informàtica, concretament en l'Enginyeria de Computadors, és fonamental conèixer el funcionament d'un computador a nivell lògic. És per això que aquesta assignatura engloba tots els conceptes adquirits durant la especialitat per aplicar-los en la realització d'un disseny d'un processador.

Aquest projecte permet a l'alumne familiaritzar-se amb les principals eines de disseny del mercat durant la primera fase del projecte. Una vegada s'ha assolit aquest objectiu l'alumne haurà de ser capaç de començar a realitzar el disseny del processador de manera guiada pel professor. Finalment l'alumne haurà d'acabar l'implementació prenent les seves pròpies decisions, ja que en l'última part del projecte es dona total llibertat per a realització de les tasques.

2.3 Planificació

Es tracta d'una planificació incremental, és a dir, partint d'un processador amb un set d'instruccions molt reduït, s'anirà incrementant les funcionalitats d'aquest i afegint noves instruccions interpretables. També s'inclou una primera fase de "hands-on" per a familiaritzar-nos amb l'entorn de treball, les seves eines i la sintaxi del llenguatge a utilitzar al projecte (VHDL). Com a resum, les fases de projecte són les següents:

1. Aprenentatge de les eines de desenvolupament per als xips programables (FPGA) i pràctica de llenguatge descriptiu de hardware VHDL
2. Implementar petits components o dispositius al xip programable de la placa base
3. Implementar una primera versió simplificada del processador a una FPGA (sense memòria externa ni suport per a sistema operatiu o dispositius externs)
4. Implementar una versió completa del processador
5. Programar un sistema de boot (BIOS) per al Sistema Operatiu al processador
6. Evaluar el rendiment de diferents aplicacions sobre la plataforma en la que s'ha dissenyat

2.4 Resultats obtinguts

Durant les diferents etapes del projecte han sorgit diferents problemes que s'han hagut d'anar solucionant. En les primeres etapes del projecte, al estar més guiades, l'implementació d'aquestes i el seu correcte funcionament va ser més senzill. A diferència de les últimes on van començar tots els problemes, ja que els petits errors o els casos no contemplats van anar afectant al correcte funcionament. A mesura que s'anaven afegint etapes el debug es va anar complicant però finalment s'han pogut implementar totes les etapes amb un correcte funcionament, excepte la de la TLB. Amb aquesta última es van presentar problemes durant l'execució del NSnake, ja que durant l'execució del programa en un determinada longitud de la Snake el programa es quedava penjat per causa d'un accés a una pàgina no present a la TLB.

2.5 Capítols del projecte

Les fases es poden reestructurar en les diferents etapes en les que hem executat el projecte (anomenarem capítols a aquestes etapes):

1. Aprenentatge de les plaques de desenvolupament
2. Processador base
3. Processador multicicle
4. Controladora de memòria
5. Unitat Aritmeticològica
6. Instruccions de Salt
7. Entrada / Sortida bàsica
8. Controladora de teclat PS/2 i video VGA
9. Interrupcions
10. Excepcions
11. Mode privilegiat i crides a sistema
12. TLB (Translation Lookaside Buffer)

Les fases 1 i 2 corresponen al capítol d'*Aprenentatge de les plaques de desenvolupament*
La fase 3 correspon als capítols *Processador Base* i *Processador Multicicle*
La fase 4 correspon als capítols *Controladora de memòria, Unitat Aritmeticològica, Instruccions de Salt, Entrada/Sortida bàsica, Controladora de teclat PS/2 i video VGA, Interrupcions, Excepcions, Mode privilegiat i crides a sistema i TLB*.
La fase 5 correspon a la part conjunta de desenvolupament: Es tracta de implementar algun tipus de millora al disseny base del processador i habilitar-lo per a poder córrer un petit sistema operatiu multiusuari (amb capacitat per a executar vàries aplicacions concurrentment).
Finalment, la fase 6 constarà d'un set de proves que executarà el nostre processador i ens permetrà avaluar el rendiment d'aquest: Cicles necessaris per a realitzar un càlcul (i extrapolant els resultat podrem extreure el temps per instrucció o MIPS)

3.Entorn de desenvolupament i recursos necessaris

3.1 Hardware

Per a dur a terme el projecte, com ja hem dit a la introducció d'aquest document, utilitzarem un xip FPGA, en aquest cas de la companyia Altera. Concretament es tracta d'un Xip Cyclone II EP2C20F484C7 amb 18752 components lògics (elements combinacionals i registres lògics). La elecció d'aquest és simplement perquè és el que porta la placa de desenvolupament que hem utilitzat per a la realització del projecte. Es tracta del model DE1 del fabricant de plaques de desenvolupament Terasic Inc.

Aquesta placa, apart de tenir els pins de la placa disponibles per als nostres dissenys, porta diferents components instal·lats i conectats amb la FPGA per a poder fer ús d'ells de forma més o menys senzilla (la placa porta adjunta la documentació pròpia i la de la resta de components que té instal·lats). Tot i que no utilitzarem molts d'aquests components (com per exemple el lector de targetes SD, un xip de memòria FLASH o un altre de memòria SDRAM) alguns si que els utilitzarem per a interactuar amb el processador que estem dissenyant. Alguns d'aquests components són:

- SRAM (512kb) : Manufacturat pel fabricant de components electrònics ISSI, és la memòria que utilitzarem per a emmagatzemar totes les dades i tot el codi utilitzats per a comprovar el funcionament del nostre processador.
- Driver VGA : A través d'una controladora proporcionada durant el desenvolupament del projecte, gaudirem d'una sortida de vídeo i una porció de memòria del xip SRAM serà utilitzada com a memòria de vídeo. Més endavant s'explicarà com s'ha connectat.
- Connector PS/2: Implementarem una controladora de teclat que connectarem a aquest port per a tenir suport de teclat al processador mitjançant ports d'entrada sortida. Més endavant es concretarà com s'ha configurat per a que sigui funcional.
- Switcs / Polsadors: Utilitzarem aquests components de la placa principalment per a debugar el nostre disseny i per a comprovar el correcte funcionament d'aquest.
- Leds / 7-segment displays: Aquests seràn utilitzats igualment per a debugar.
- Altera MAX: Disposem d'un programador de FPGA i un connector blaster ubicat físicament a un port USB-B per a facilitar la programació d'aquesta desde les eines software.

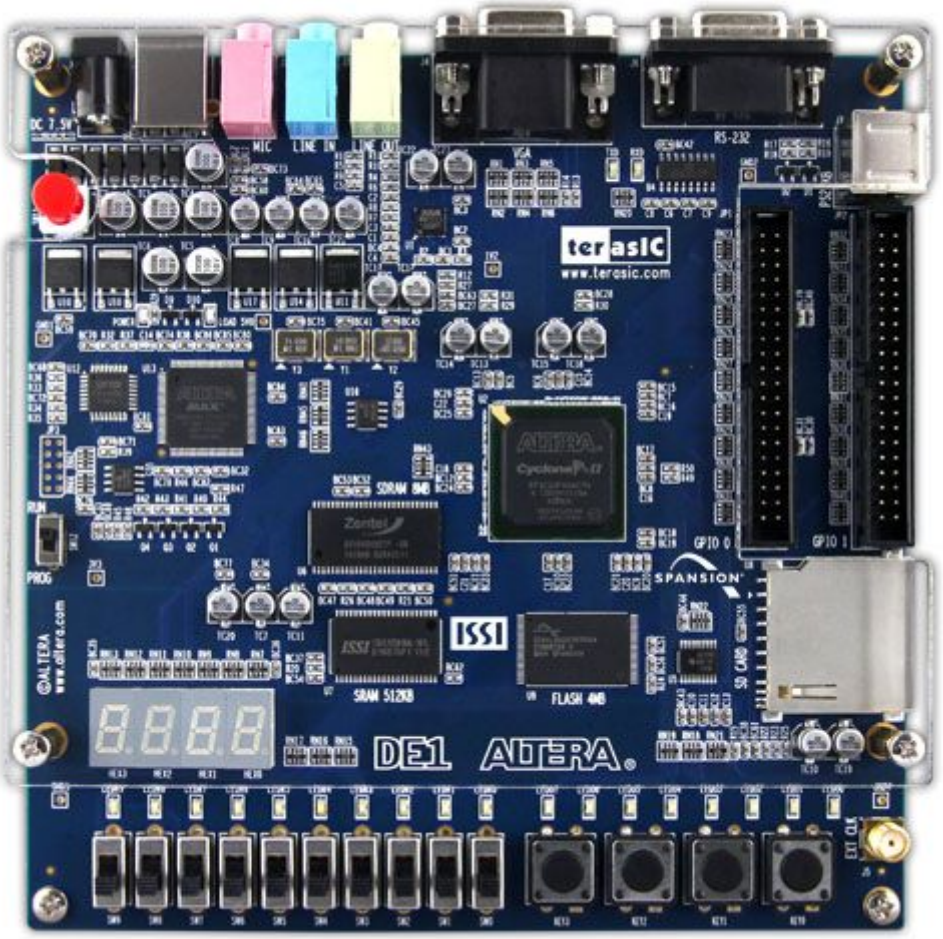

fig 3.1: *Placa de desenvolupament DE1 del fabricant Terasic.*

3.2 Software

- Quartus II 13.0sp1
  Es tracta d'una eina de software produïda per Altera per l'anàlisi i la síntesi de circuits en HDL. Tot i que permet la edició dels documents, el seu workflow en aquest sentit no està tan cuidat com a eines més senzilles com *Notepad*++, que per produir codi ens és molt més fiable. Quartus II l'utilitzarem durant tot el projecte per a compilar i validar els nostres dissenys, ademés d'usar-lo com a eina central de la "suite" d'Altera que inclou també el següent software:

  - Multisim
    Es tracta d'un simulador de circuits que accepta arxius de descripció HDL. Ens permet fer simulacions dels nostres dissenys per validar la seva correctesa d'una forma més ràpida (la compilació per a la simulació és molt mes ràpida que la compilació per a programar la FPGA ja que només hauria de validar el disseny i la connectivitat d'aquest).
    És una eina molt vàlida però no es pot garantir que el nostre disseny funcioni a la FPGA ja que, entre d'altres coses, s'ignora el temps de propagació de les senyals.

  - SignalTAP
    És una eina de debugatge. Un cop carregat el nostre disseny a la FPGA, amb aquesta eina podrem debugar el seu funcionament. Gràcies a la seva connectivitat JTAG, ens permet visualitzar els estats de les senyals internes del nostre disseny que volguem depurar. A diferència d'un debugger de software, quan marquem un break (sempre serà per canvi en l'estat d'una senyal) el sistema no s'aturarà, però podrem visualitzar les senyals demanades en l'instant del canvi a la senyal marcada.

  - DE1 Control Panel
    Es tracta d'una petita interfície dissenyada per Terasic que serveix per a comunicar-se amb la placa de desenvolupament. Un cop carregat el disseny proporcionat amb aquesta eina a la FPGA, podrem connectar-la amb aquesta interfície per més tard encendre i apagar els leds, dibuixar coses pels visors i carregar dades i codi als xips de memòria (en el cas d'aquest projecte només utilitzem la memòria SRAM)

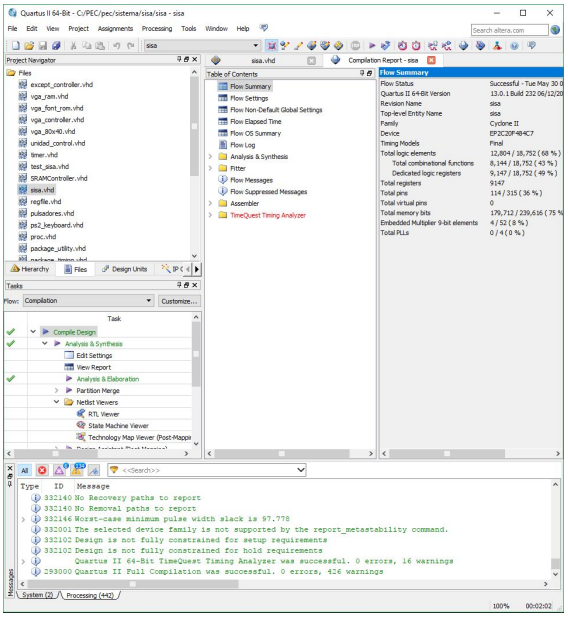

fig 3.2: *Quartus Software*

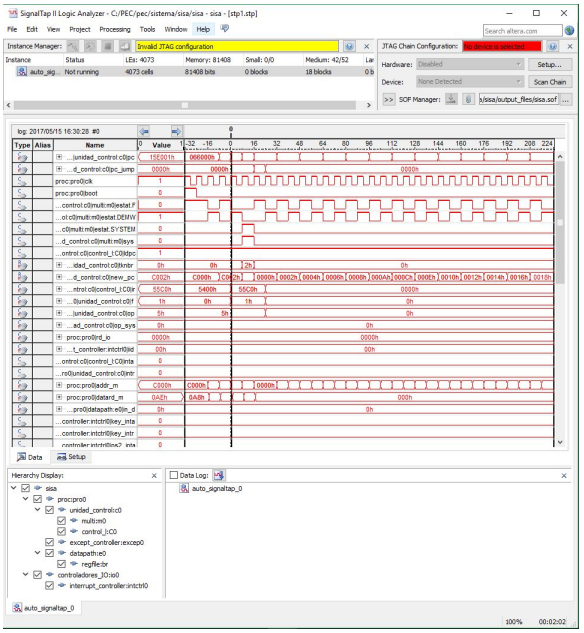

fig 3.3: *SignalTAP*

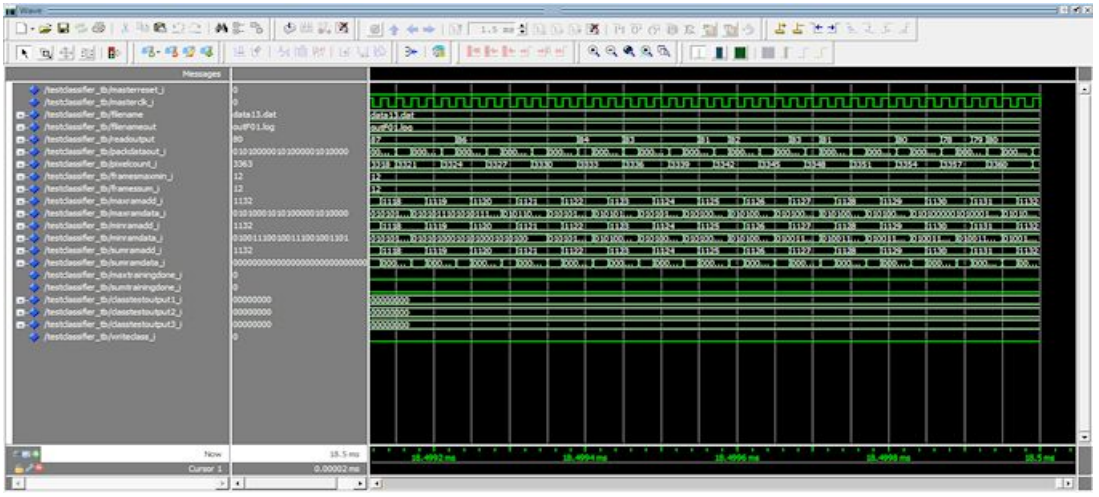

fig 3.4: *Modelsim*

4.Etapes

Durant el desenvolupament del projecte s'han anat incorporant diverses etapes en les quals s'han discutit decisions de disseny i s'han escollit diferents estratègies d'implementació. En aquest apartat es poden veure quines etapes han format part del projecte i quines decisions s'han pres.

4.1 Aprenentatge de les plaques de desenvolupament

Aquesta etapa és un primer contacte amb les eines de desenvolupament. Es tracta de realitzar tota una sèrie d'exercicis dissenyats per a entendre el funcionament del llenguatge descriptiu i de les eines disponibles. Es tracta d'una etapa totalment guiada en la que no s'han pres gaires decisions de disseny ja que es tractava d'habituar-nos a l'entorn de treball.

Tot i això ens vam trobar dificultats durant l'última fase d'aquesta etapa, en la qual es demanava implementar un intèrpret Morse. Al ser l'inici de curs no vam tenir en compte que una mateixa senyal no es podia modificar dos process diferents. La solució va ser declara dues senyals i modificar cadascuna en el seu corresponent process. D'aquesta manera es podia consultar el senyal desde el altre process sense haver de modificar-la.

4.2 Processador base

És la primera fase real en el disseny del processador. En aquesta etapa s'han definit els primers elements del processador, així com els seus components principals: una unitat de control, un camí de dades amb un banc de registres i una ALU bàsica.
Aquest sistema té un set d'instruccions amb 2 instruccions: MOV i MOVHI, per tant, la nostra ALU únicament haurà d'interpretar aquestes dues instruccions i moure els bits a les posicions corresponents (part alta o part baixa del registre).

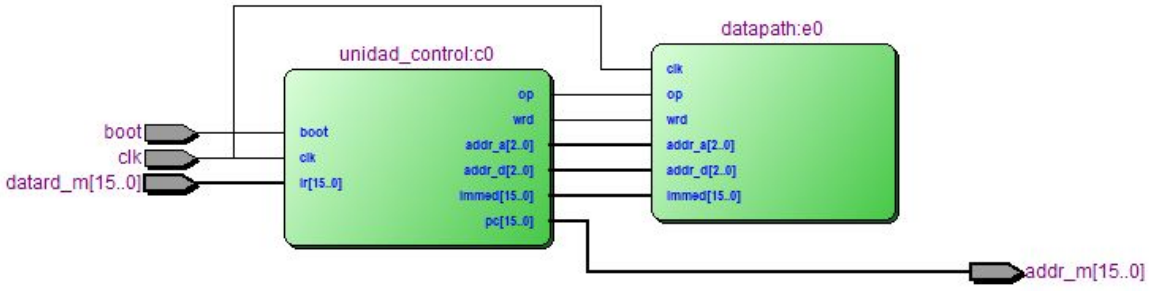

fig 4.1: *Estructura inicial del processador*

Com es pot comprovar a la figura 4.1, el sistema no té cap tipus de sortida: Ara mateix únicament es poden moure literals als registres de propòsit específic i per tant, per a comprovar el correcte funcionament, el simularem amb el ModelSim.

En aquesta etapa a la unitat de control està dotada d'un *Program Counter* i d'una senzilla lògica de control que únicament desvia els bits de la instrucció cap a les senyals de control sense aplicar cap mecanisme lògic (per ara no és necessari).

Les senyals que si que necessitem son *ldpc, wrd, addr_a, addr_d, immed*:

- **ldpc:** permís de càrrega del PC. Si es 0, el PC deixarà d'actualitzar-se (*fig. 4.2*).
- **wrd:** permís d'escriptura. S'activa quan s'ha d'escriure una dada a alguna posició del banc de registres.
- **addr_a:** adreça del banc de registres a llegir. La dada sortirà del BR i anirà a la ALU.
- **addr_d:** adreça de destí de les dades a escriure al banc de registres. La dada la recollirem pel port *d* del banc de registres.
- **immed:** valor literal a escriure al banc de registres. Si es tracta de la instrucció MOV la ALU seleccionarà els 16 bits de menor pes que li entren pel port *y* i extendrà el signe. Mentre que l'instrucció MOVHI posiciona les dades a la part alta.

```
process (clk, boot)
begin
    if boot='1' then
      next_instruction <= x"C000";
    else
      if rising_edge(clk) then
          if (ldpc='1') then
            next_instruction <= next_instruction + 2;
          end if;
      end if;
    end if;
end process;
```

fig. 4.2: *PC del processador unicicle*

El PC s'ha d'incrementar en 2 a cada tic de rellotge, ja que les instruccions son de 16 bits i l'adreçament a memòria es a nivell de byte. Per tant, cada instrucció ocuparà dos bytes.

El banc de registres, segons la especificació SISA, consta de 8 registres[1] de propòsit específic. Les lectures son asíncrones i les escriptures, per a garantir que pel port *d* la dada a escriure estigui llesta, les farem síncrones (*fig. 4.3*).

```
a <= banc_reg(conv_integer(addr_a));

escriu : process(clk)
begin
  if (wrd = '1') then
    if( rising_edge(clk) ) then
      banc_reg(conv_integer(addr_d)) <= d;
    end if ;
  end if;
end process ; -- escriu
```

fig. 4.3: *primera versió del banc de registres*

[1] http://docencia.ac.upc.edu/FIB/grau/PEC/Protegido/Documentacion/SISA-F-Sep-2007.pdf

4.3 Processador multicicle

Aquesta etapa implementa un sistema de màquina d'estats al processador. Aquesta s'anomenarà *multi* i en tot moment indicarà quina etapa s'està executant en el processador. Això farà que la lògica de control es compliqui una mica més.

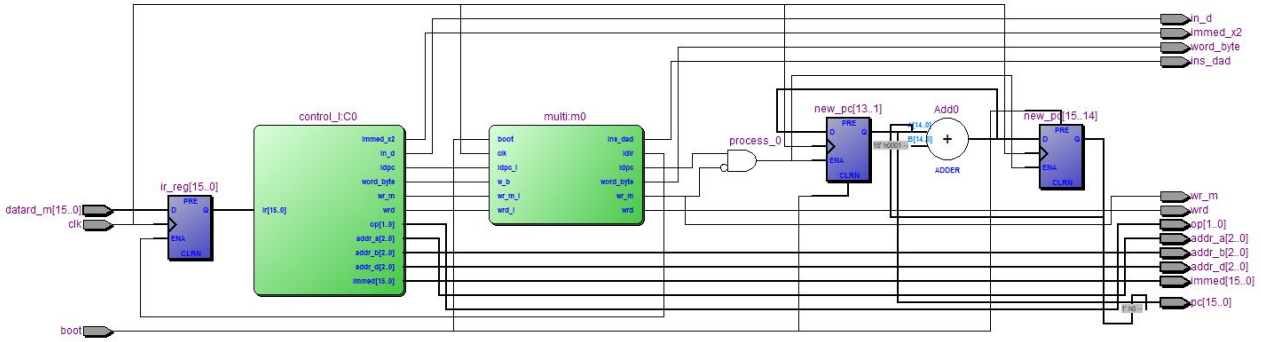

fig. 4.4: *unitat de control de la segona etapa del projecte*

Necessitem de 2 estats, ja que la memòria de dades i d'instruccions és la mateixa i podrien aparèixer riscos estructurals. Durant la etapa de fetch únicament recollirem la instrucció a decodificar de la memòria i posteriorment, a la etapa DEMW llegirem les dades de memòria o escriurem en ella (si es necessari).

El decodificador d'instruccions ara ja no és tan trivial com a la etapa anterior. Tot i així segueix sent una implementació senzilla ja que únicament hem de controlar la instrucció que hem carregat. La principal millora que hem realitzat ha estat la nova definició de la senyal d'instrucció op. En previsió de futures ampliacions hem considerat convenient crear un nou *TYPE* anomenat instrucció que ens facilitarà la comprensió de la implementació de la entitat (fig. 4.4, 4.5).

```
TYPE instruccio IS (ST,STB,LD,LDB,MOVI,HALT);
SIGNAL op_code: instruccio;
```

fig. 4.4: *definició del nou tipus "instrucció"*

```
wr_m      <= '1' when (op_code = ST or op_code = STB) else '0';
in_d      <= '1' when (op_code = LD or op_code = LDB) else '0';
immed_x2  <= '1' when (op_code = LD or op_code = ST) else '0';
word_byte <= '1' when (op_code = LDB or op_code = STB) else '0';
```

fig. 4.5: *el codi resulta molt més comprensible decodificat*

S'han generat noves senyals de control per a l'ampliació de ports del banc de registres i la escriptura a memòria. Aquestes senyals son:

- **addr_b:** adreça del port b de lectura del banc de registres. La dada que surt pel port *b* la farem arribar al port d'escriptura de la memòria.
- **wr_m:** senyal de permís d'escriptura a memòria (per a les instruccions ST, STB)
- **in_d:** selecciona la procedencia de la dada a escriure: pot venir de la ALU o de la memòria
- **immed_x2:** indica si s'ha de fer un desplaçament a l'immediat com en les instruccions LD i ST
- **word_byte:** indica a la memòria si es realitza un accés a byte. Aquesta senyal la genera el mòdul multi.

Totes les adreços (de memória, ports de lectura o d'escriptura del banc de registres) s'extreuen directament de la codificicació de la instrucció i s'envien cap als dispositius que les requereixin. Cal recordar que encara no treballem amb memòria real i per tant únicament es pot comprovar el funcionament al ModelSIM amb una implementació de memòria que se'ns proporciona durant el projecte

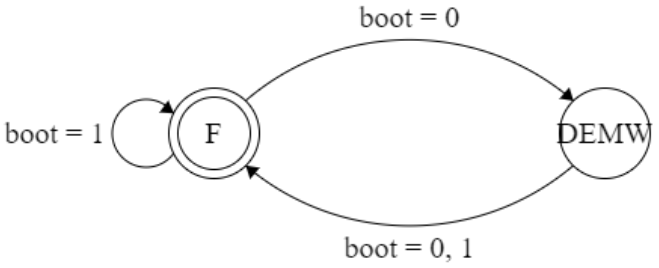

fig. 4.6: *Diagrama d'estats del processador multicicle*

En la figura 4.6 podem veure la màquina d'estats que forma part del mòdul multi comentat anteriorment. Ademés inhibeix les senyals que provenen del decodificador a l'etapa de fetch i controla la senyal *ldpc*.

Al banc de registres se li ha afegit un segon port de lectura amb la mateixa implementació que el que ja teníem (lectures asíncrones)

4.4 Controladora de memòria

En aquesta etapa es dissenyarà una controladora de memòria per al nostre projecte que es connectarà als drivers de la memòria SRAM de la placa de desenvolupament, ja que fins aquest moment s'estava treballant amb un emulador de la memòria. D'aquesta manera es podrà començar a treballar amb la FPGA. L'objectiu d'aquesta etapa és poder afegir les instruccions corresponents a memòria (LD, ST, STB, LDB) i mantenir el correcte funcionament del processador després d'aquesta millora.

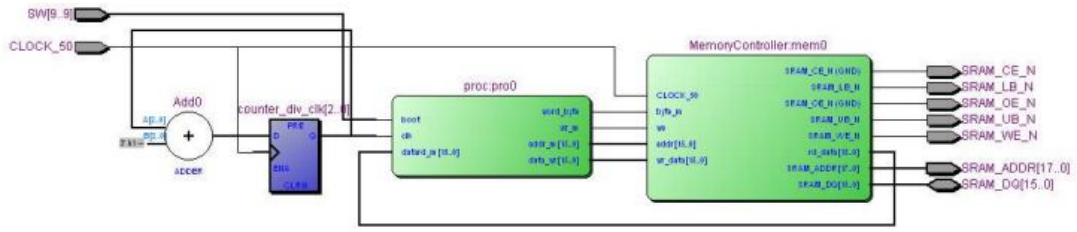

fig. 4.7: *Esquema de blocs de la entitat SISA*

Com es pot veure en la figura 4.8 en l'esquema de blocs del SISA s'ha intercanviat l'emulador de memòria per el controlador de memòria que treballarà directament amb la FPGA. El controlador serà l'encarregat d'una correcta lectura i escriptura de les dades que interactuen amb la SRAM de la placa. És per això que haurà de saber gestionar les següents senyals:

- **clk:** Aquest rellotge no és el mateix amb el qual funciona el procesador, és més ràpid, ja que d'aquesta manera ens permet realitzar operacions de memòria en 1 cicle de processador.
- **addr:** Correspon a l'adreça de memòria generada pel processador a la que es vol accedir.
- **wr_data:** Correspon a la dada que es vol escriure a memòria.
- **rd_data:** Correspon a la dada llegida de memòria de la posició addr.
- **we:** És la senyal encarregada d'informar al controlador si s'està produint una lectura o una escriptura.
- **byte_m:** És la senyal encarregada d'informar al controlador si s'està produint un accés a memòria a nivell de byte o word.
- **SRAM_ADDR:** Correspon a l'adreça que va dirigida a la memòria SRAM. Aquesta no és la mateix que ens arriba per l'entrada addr provinent del processador, ja que el sistema de direccionament canvia. És per això que en aquest mòdul s'ha manipulat l'adreça d'entrada per a un correcte direccionament.

- **SRAM_DQ**: És un bus bidireccional (d'entrada i sortida) pel qual van les dades cap i des del xip de memòria. Com que és un bus bidireccional implica que diversos components posaran dades en ell en diferents moments. S'ha de garantir en el disseny que els dos components mai posin un valor al bus de forma simultània. Per això, quan un component no vulgui escriure al bus haurà de posar les seves sortides en alta impedància (High-Z).
- **SRAM_UB_N :** Senyal de control de la SRAM. S'encarregà d'informar a la memòria si s'està produint la lectura o una escriptura de la part alta.
- **SRAM_LB_N:** Senyal de control de la SRAM. S'encarregà d'informar a la memòria si s'està produint la lectura o una escriptura de la part baixa.
- **SRAM_CE_N:** Senyal de control de la SRAM. S'encarregà d'activar la memòria per tant sempre a de prendre el valor 0.
- **SRAM_OE_N:** Senyal de control de la SRAM. S'encarregà d'activar la memòria per tant sempre a de prendre el valor 0.
- **SRAM_WE_N:** Senyal de control de la SRAM. S'encarrega d'informar a la memòria si s'està produint una lectura o una escriptura.

Per tal d'una correcte gestió de les senyals de la SRAM s'han definit diferents estats per facilitar la seva comprensió. Encara que la implementació s'hagués pogut definir amb 2 estats, hem preferit fer-ho amb 3 perquè sigui més entenedor. Aquest diagrama d'estats ve controlat pel rellotge clk_50 (50 Mhz) mentre que a la CPU li arriba un rellotge 4 vegades mes lent (clk_counter(2)). Gràcies a aquest disseny hem pogut realitzar un controlador de memòria que serveix les dades sense temps de penalització, és a dir, podem servir dades provinents de memòria en tan sols 2 tics de rellotge (El que triga una instrucció).

Per tant com podem veure a la figura 4.8 quan el processador està a la etapa de FETCH, la memòria està a l'estat INIC. Al següent cicle, depenent de la instrucció que haurem llegit de memòria, anirem a l'estat ESCRIP o LECT. Al cicle següent, ja haurem servit la instrucció i per tant retornarem a l'estat inicial INIC.

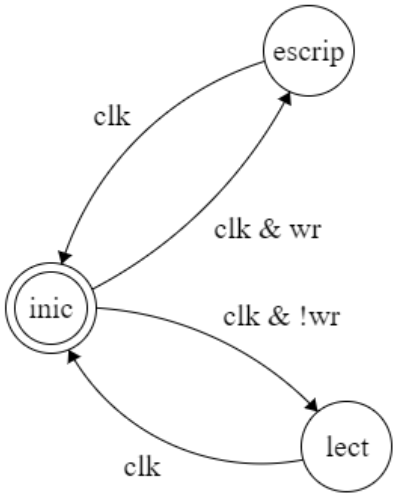

fig. 4.8: *Diagrama d'estats de la controladora de memòria*

Com s'ha dit anteriorment en aquesta nova etapa del projecte ja es començarà a treballar amb la placa i per tant es podran utilitzar noves eines per al debug del nostre disseny com el Signal Tap II.

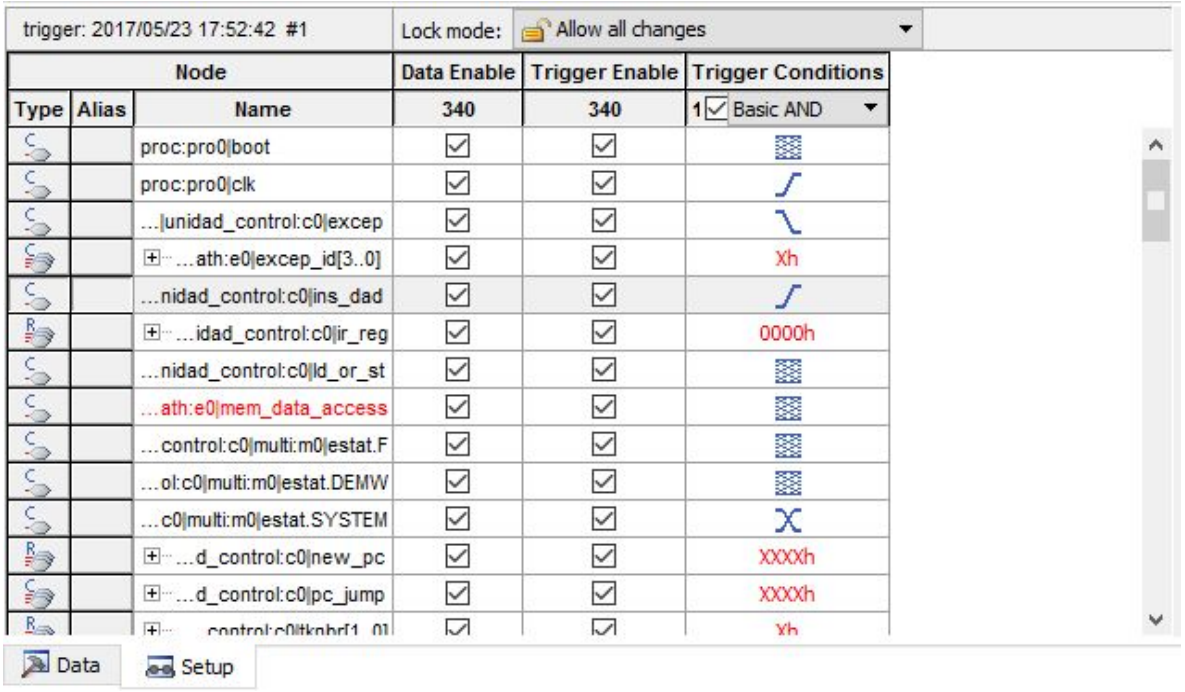

fig. 4.9: *Finestra de configuració de les senyals al Signal Tap II*

Aquesta eina ens permet seleccionar quines senyals volem estudiar durant l'execució del nostre programa a la placa. Gràcies aquesta tenim diferents mecanismes per estudiar el comportament del programa:

- **Don't care:** És el primer símbol que es pot observar en la senyal *proc0:boot.* Aquest ens indica que la senyal seleccionada no és rellevant per al nostre testeig. Per tant els futurs canvis que aquesta presenti durant l'execució seran ignorats.
- **Rising edge:** És el segon símbol que es pot observar en la senyal *proc0:clk.* Aquest ens indica que en el moment que es produeixi un flanc ascendent en aquesta senyal, l'execució del nostre programa es paralitzarà, per tal de poder veure l'estat actual de les senyals.
- **Falling edge:** És el tercer símbol que es pot observar en la senyal *unidad_control:c0:excep.* Aquest ens indica que en el moment que es produeixi un flanc descendent en aquesta senyal, l'execució del nostre programa es paralitzarà, per tal de poder veure l'estat actual de les senyals.
- **Either edge:** És el símbol format pels dos anteriors com es pot veure en la senyal *multi:M0:estat.SYSTEM.* Aquest ens indica que en el moment que es produeixi un

flanc ascendent o descendent en aquesta senyal, l'execució del nostre programa es paralitzarà, per tal de poder veure l'estat actual de les senyals.

- **Assignació d'un valor determinat (XXXXh):** Si es vol controlar quan un registre pren un valor determinat i parar-ne l'execució per estudiar les altres senyals, es pot efectuar aquesta operació inserint la dada desitjada en la senyal que ens interessi estudiar. Com s'ha fet en la senyal *c0:ir_reg*

4.5 Unitat Aritmeticològica

En aquesta etapa s'implementaran totes les operacions de càlcul ja que fins ara la nostra ALU únicament realitza sumes i concatenacions.

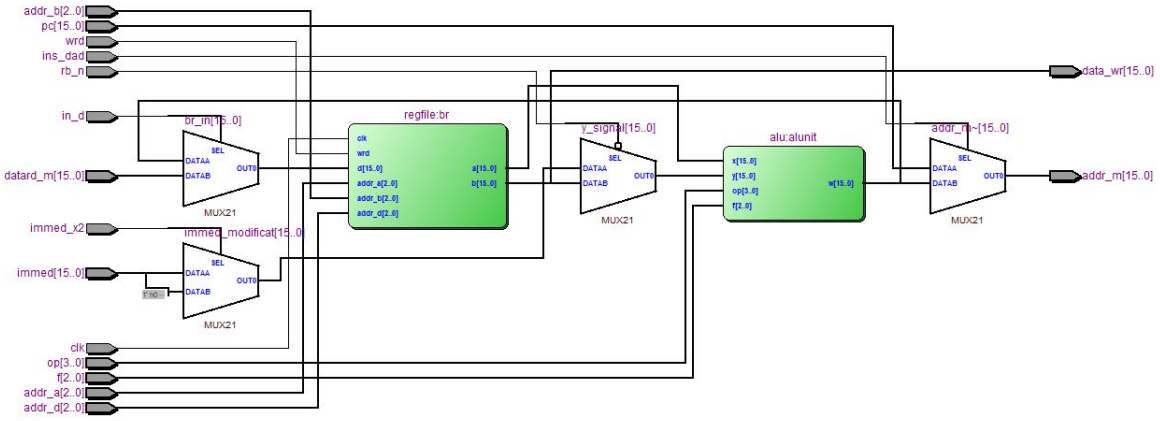

fig 4.10: *Camí de dades per a la etapa de ALU*

A la unitat de control hem afegit els nous codis d'operació per a la ALU i també s'han afegit les noves operacions a la lògica de control(fig. 4.11). Al datapath s'ha afegit un multiplexor que permet connectar el port b del banc de registres a la entrada y de la ALU. Aquesta selecció es fa mitjançant la senyal *rb_n* que generarem al decodificador.

```
TYPE instruccio IS (LA,CMP,ADDI,ST,STB,LD,LDB,MOVI,EXT,HALT);
SIGNAL op_code: instruccio;
```

fig 4.11: *Nous estats afegits a op_code*

La implementació de la ALU encara que semblés trivial al plantejar-la, ens ha suposat problemes pels conflictes entre llibreries en les operacions amb extensió de signe (multiplicacions i divisions). Els castings entre diferents tipus dificultaven bastant la comprensió del codi i no trobàvem una única llibreria per a realitzar totes les operacions amb STD_LOGIC_VECTOR, ja que *numeric* treballa amb integers i per a possibles futures ampliacions hem decidit utilitzar altres tipus que no siguin STD_LOGIC, el mínim possible. Hem optat per utilitzar les llibreries necessàries per a cada operació o família d'operacions individualment. Això ho hem aconseguit creant una entitat per a cada tipus d'operació (MULT, MULTU, DIV, DIVU), totes al mateix document (alu.vhd).

Tot i així hem creat una funció per a utilitzar els operadors de comparació de la llibreria arith, ja que no hem trobat un cast directe de boolean a std_logic i consultant la bibliografía hem trobat aquesta funcionalitat[2]

4.6 Instruccions de Salt

L'objectiu d'aquesta etapa és implementar instruccions de salt per tal de permetre la ruptura del seqüènciament implícit. En aquesta fase per això no s'implementarà la instrucció de crida al sistema CALLS, ja que es necessita tenir implementat el mode sistema primer al processador. Per tant ens centrarem en les instruccions de salt condicional(BZ, BNZ) i les de trencament de seqüència(JMP, JZ, JNZ i JAL).

Per a les instruccions de salt condicional necessitem el valor del camp *"b"* de la sortida del banc de registre per tal d'avaluar la condició de salt. Per tal d'efectuar aquesta comprovació hem dissenyat un modul en el datapath que ens comprova si el valor de la sortida del banc de registre es 0 o no (Senyal z).

```
WITH b_signal SELECT
    z <= '1' WHEN x"0000",
         '0' WHEN others;
```

fig 4.12: *Multiplexor que genera el senyal z*

Per a les instruccions de trencament de seqüència necessitem conèixer l'adreça de destí (ubicada a un registre). Aquesta dada la farem arribar a la unitat de control mitjançant un curtcircuit el qual agafarà la dada de la sortida "b" del banc de registres. A partir d'ara, el següent PC estarà controlat per el senyal tknbr com es pot veure en la figura 4.13 el qual es genera a la unitat de control a partir de la instrucció emmagatzemada al *ir*. El process de clock ara pot actualitzar-se amb una adreça no seqüencial.

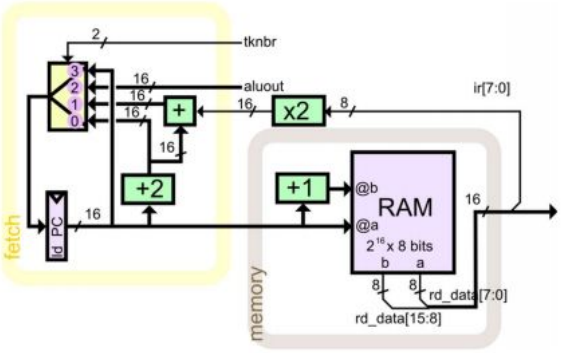

fig 4.13: *Multiplexor encarregat de seleccionar següent PC*

Al provar la nova implementació de la unitat de control al ModelSIm, ens vam adonar que el PC saltava a adreces que no hauria d'anar, ja que la senyal *tknbr* s'actualitzava a la etapa

[2] The VHDL Handbook, David R. Coelho ISBN: 978-1-4612-8902-9

de fetch i mostrava resultats imprevisibles. La solució que hem trobat ha estat controlar tant la senyal de *tknbr* com el contingut d'*ir_reg* amb la senyal de clock mitjançant un process.

4.7 Entrada / Sortida bàsica

L'objectiu d'aquesta etapa es permetre la comunicació amb els dispositius d'entrada i sortida. Per tal de fer-ho possible s'han generat un controlador de dispositius que en permetrà encendre els LEDs, utilitzar l'agrupació de 4 visors de 7-segments per permetre la visualització de valors numèrics, llegir el valor de 8 switchs i controlar 4 polsadors que té la placa.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0		
0 1 1 1	Rd	0	n n n n n n n n	Input	IN	
	Rb	1		Output	OUT	

fig 4.14: *Codificació de les instruccions IN/OUT*

Com en aquesta etapa no es disposa d'interrupcions l'ús d'aquests dispositius s'haurà de realitzar mitjançant les operacions IN i OUT com es pot veure a la figura 4.14. Aquestes s'encarregaran d'escriure i llegir una estructura formada per 256 registres de 16 bits situada en un component separat del processador on cadascun d'aquests estarà assignat a un dispositiu i/o. El disseny d'aquest component tindrà la següent estructura:

fig 4.15: *Entitat del component controladores_IO*

- **boot:** Senyal encarregada d'inicialitzar els senyals d'aquest component com per exemple iniciailitzar l'estructura de registres a 0.
- **addr_io:** S'encarrega d'informar al controlador I/O amb quin dispositiu s'està treballant. Aquesta senyal serà l'iterador que ens permetrà seleccionar el registre(dispositiu) que volem accedir.

- **wr_io:** Bus encarregat de transferir al controlador I/O l'informació que es vol escriure en algun dels registres dels dispositius.
- **rd_io:** Bus encarregat de transferir dades del controlador I/O cap el processador.
- **wr_out:** Senyal que ens indica si l'instrucció és de sortida.
- **rd_in:** Senyal que ens indica si l'instrucció és d'entrada.
- **led_verdes:** Bus encarregat de transferir dades cap els ports dels leds verds.
- **led_rojos:** Bus encarregat de transferir dades cap els ports dels leds vermells.
- **switch:** Bus encarregat de transmetre les dades recollides dels interruptors.
- **keys:** Bus encarregat de transmetre les dades recollides dels polsadors.
- **HEX0:** Bus encarregat de transmetre les dades generades pel processador cap el visor0 de 7-segments.
- **HEX1:** Bus encarregat de transmetre les dades generades pel processador cap el visor1 de 7-segments.
- **HEX2:** Bus encarregat de transmetre les dades generades pel processador cap el visor2 de 7-segments.
- **HEX3:** Bus encarregat de transmetre les dades generades pel processador cap el visor3 de 7-segments.

Per tal de permetre la comunicació entre els dispositius I/O i el processador els senyals *wr_io, rd_io, wr_out, rd_in i addr_io* els portarem cap aquest últim per generar les senyals corresponents amb el *ir* de la instrucció. A més a més per seleccionar que la entrada del banc de registre sigui una dada llegida d'un dispositiu necessitarem afegir la senyal rd_io a la entrada *d* del BR. Aquesta nova entrada no suposarà cap problema per a al senyal de control que gestiona aquest multiplexor ja que al disposar de 2 bits no farà falta ampliar-la.

4.8 Controladora de teclat PS/2 i video VGA

Aquesta etapa esta dedicada a la implementació de les controladores de teclat i de pantalla al nostre projecte. Com que es tracta de perifèrics, els connectarem a través dels ports de lectura i escriptura que acabem d'implementar en la etapa anterior, i tot el seu comportament es controlarà a través d'aquests. Com que encara no tenim un mecanisme d'interrupcions implementat, la interacció amb aquests perifèrics s'haurà de fer per consulta (*polling*).

Amb el document explicatiu de la etapa, se'ns proporcionen diverses controladores de teclat i pantalla, amb petites diferències entre elles. Durant el plantejament de la etapa, vam decidir utilitzar les versions senzilles de les controladores.

Com ja hem dit, la idea es conectar els inputs i outputs d'aquestes controladores als ports del *IO_controller*.

L'accés al teclat es fa a través dels ports 15 i 16 d' *IO_controller*, el 15 contindrà el codi ASCII de l'últim caràcter premut, mentre que al port 16 conté l'avís de nou caràcter polsat. Aquest avís es desactivarà fent una escriptura qualsevol al port 16 *(fig. 4.16)*.

```
if (wr_out = '1') then
   --ack keyboard
   if (addr_io = x"10") then
      io_ports(16) <= x"0000";
      clear_char <= '1';
```

fig. 4.16: *Sistema d'ack de lectura de tecles*

Un dels problemes que ens vam trobar va ser precisament per aquest avís, ja que només escrivíem al port i no activàvem el senyal *clear_char*. Això ho vam trobar ja que als jocs de prova rebíem els chars corresponents a les tecles, però no ens contava la quantitat de repeticions d'aquella tecla.

Finalment quan hem comprovat que les noves funcionalitats es comportaven correctament, hem provat a canviar la controladora de pantalla per la segona versió que se'ns ha proporcionat (la que te implementada la funcionalitat de cursor) i hem observat que els glitches que ens apareixien amb la primera versió han desaparegut, tenint tota la imatge molt millor definició i sense marges borrosos. Degut a aquesta millora, decidim canviar la controladora VGA i utilitzar la segona versió fins al final del projecte.

4.9 Interrupcions

L'objectiu d'aquesta etapa és incorporar mecanisme d'interrupcions. Per a això s'ha afegit un nou cicle a la màquina d'estats com es pot veure a la figura 4.17 que activarà el mecanisme d'atenció a excepcions. El programa saltarà a una adreça preestablerta, que correspondrà a la RSG on es faran les operacions necessàries establertes pel SO per servir la interrupció corresponent.

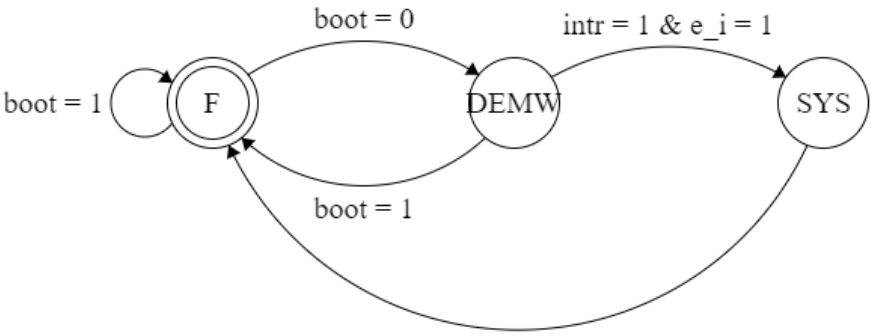

fig. 4.17: *Nova màquina d'estats amb l'estat SYSTEM*

El mecanisme d'interrupcions, encara que s'ha implementat en versions prèvies a la implementació de les excepcions funciona sobre aquestes. en aquesta etapa s'ha creat un mòdul anomenat interrupt_controller, que es troba dintre del *controladores_I/O,* que captura totes les interrupcions que es generen en els diferents dispositius (*switch_intr, timer_intr, ps2_intr i key_intr*) per més tard comunicar-li al processador aquestes esdeveniments. És per això que el mòdul interrup_controller té com a funció identificar quina interrupció s'ha produït i generar un id d'interrupció (iid) com podem veure a la figura 4.18.

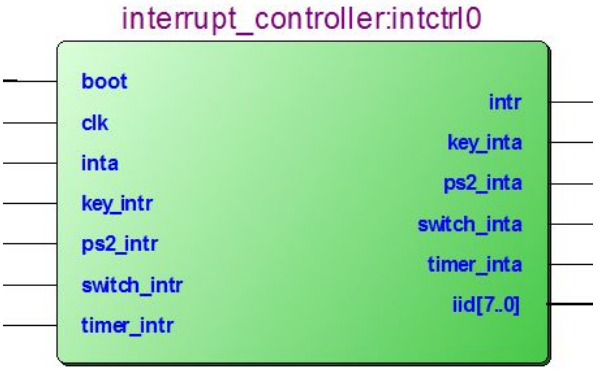

fig. 4.18: *Entitat del component interrupt_controller*

Paral·lelament el processador extreu una senyal de petició d'atenció (intr) que va a parar al mòdul d'excepcions per a que aquest generi l'id d'excepció 15 (interrupcions). Quan el processador entra al cicle SYSTEM, fa saltar el PC a la RSG on es consulta l'id d'excepció i s'adona que es tracta d'una interrupció. Una vegada s'ha identificat la interrupció saltarà a la rutina específica per atendre aquella interrupció. L'ack (inta) que retornem a l'element que ha generat la interrupció es farà a través de la instrucció GETIID (per a que això funcioni, aquesta instrucció haurà d'estar a dins de la RSG i activar la senyal inta), que mourà l'id de la interrupció al registre destí codificat a la instrucció.

Durant l'implementació de les interupcions s'ha volgut mantenir el correcte funcionament de totes les fucnionalitats anteriors. És per això que per tal de mantenir la espera activa *polling,* s'ha elaborat una porta or entre el senyal *clear_char* corresponent al *polling* i el *ps2_inta* corresponent al *inta* de la interrupció.

4.10 Excepcions

L'objectiu d'aquesta etapa es incorporar el mecanisme d'excepcions el qual és molt similar al implementat amb les interrupcions. En aquest cas quan es produeixi una excepció es procedirà al mateix al sistema. Primer es saltarà a la RSG on s'identificarà quina excepció ha saltat(números del 0 al 15, on aquest últim serà una excepció de tipus interrupció) per més tard saltar a la rutina pròpia d'aquella excepció per tal de servir-la. A continuació es mostren les diferents excepcions incorporades (s'inclouen també les de la següent etapa, excepcions de TLB):

- **Excepció 0:** Instrucció il·legal. Aquesta excepció es produeix quan el decodificador del processador es troba amb un codi de instrucció que no té implementat.
- **Excepció 1:** Alineament senar. Es produeix quan es fa un fetch d'una instrucció que té una direcció de memòria senar o es produeixi un load o store de word amb una direcció senar.
- **Excepció 2:** Overflow en una operació de coma flotant.
- **Excepció 3:** Divisió per zero en coma flotant. Aquesta excepció es dona quan la operació que ha de realitzar la ALU és una divisió de coma flotant i el segon operador és un zero.
- **Excepció 4:** Divisió per cero. Aquesta excepció es dona quan l'operació que ha de realitzar la ALU és una divisió d'enters o naturals i el segon operador és un cero.
- **Excepció 6:** Miss a la TLB d'instruccions. S'activa aquesta excepció quan es fa un fetch d'una direcció, la pàgina de la qual no es troba al TLB i per tant no es pot traduir la direcció física.
- **Excepció 7:** Miss a la TLB de dades. Aquesta excepció es produeix quan s'intenta fer un load o store en una direcció, la pàgina de la qual no es troba al TLB de dades i no es pot traduir la direcció física.
- **Excepció 8:** Pàgina invàlida al TLB d'instruccions. Es produeix quan s'intenta fer un fetch d'una direcció que es troba en una pàgina amb el bit d'invàlida al TLB.

- **Excepció 9:** Página invàlida al TLB de dades. Igual que l'anterior però quan es realitza a una página referenciada amb un load o store.
- **Excepció 10:** Pàgina protegida al TLB d'instruccions. Es fa un fetch en mod eusuari d'una pàgina que pertany al sistema operatiu (té el bit de protecció activat) i, per tant, només ell pot executar el codi d'aquesta pàgina.
- **Excepció 11:** Pàgina protegida al TLB de dades. De mode similar a l'anterior, el programa d'usuari està executant-se en mode no privilegiat i intenta llegir (load) o escriure (store) en una pàgina que pertany al sistema operatiu (té el bit de protecció activat).
- **Excepció 12:** Pàgina de només lectura. Aquesta excepció s'activa quan una instrucció de store fa un accés a una pàgina que té el bit de només lectura activat al TLB de dades.
- **Excepció 13:** Excepció de protecció. s'activa quan s'està executant una instrucció d'entrada/sortida o de sistema i en canvi el processador està en mode usuari.
- **Excepció 14:** Crida a sistema. Aquesta no és pròpiament una excepció sinó que es produeix quan es realitza el CALLS.
- **Excepció 15**: Interrupció. Aquesta excepció no és tampoc una excepció sinó que s'activa quan es produeix una interrupció hardware , tal i com ja hem implementat en la etapa anterior.

fig. 4.19: *Entitat del component except_controller*

El mòdul d'excepcions està implementat a la entitat excep_controller com es pot veure a la figura 4.19 i consta d'una senyal d'entrada provinent de cada mòdul on es pot detectar una de les situacions d'excepció llistades anteriorment. Aquestes senyals generen un id intern d'excepció que a la vegada modifica la senyal excep (és la senyal que avisa al processador que ha d'anar al cicle de system per atendre la excepció). Quan s'activa el mode system, depenent de quin id d'excepció surti del mòdul excep_controller (com consultar un accés mal alineat a memòria, per exemple), el banc de registres realitzarà unes decisions determinades depenent del tipus d'excepció que s'ha generat. A més a més s'ha afegit una senyal *enabled_int* al *except_controlle*r per inhibir les interupcions en cas que es faci una instrucció DI.

4.11 Mode privilegiat i crides a sistema

En aquesta etapa els canvis han sigut mínims, hem afegit la instrucció CALLS que es tractarà com una excepció ja que ens activarà l'estat de SYSTEM per anar a la RSG. Haurem de proporcionar un registre on guardar la i*d* per a atendre la crida a sistema i des de la RSG saltarem a la rutina específica (igual que atendre una excepció o una interrupció però l'identificador s'emmagatzema prèviament en un registre rX).

Un cop s'ha entrat en el cicle SYSTEM s'activarà el mode sistema (bit S7<0> = 1) i el retorn a mode usuari es farà a través de la instrucció RETI. A partir d'ara les instruccions protegides únicament es podran executar en mode sistema (principalment totes les operacions amb op_code = 1111), si intentem executar-les amb el mode sistema desactivat, es produirà una excepció amb id = 13. També hem habilitat zones protegides de memòria que únicament es podran accedir des de aquest mode sistema. Finalment aquesta protecció també s'extén als registres de sistema (sX), ja que les instruccions RDS i WRS passen a ser instruccions protegides i per tant únicament es podrà accedir a aquests en mode sistema.

A la controladora d'excepcions hem afegit les noves que es podrán produïr:

- Excepció 12: Accés a memòria protegida. S'intenta accedir a regions de memòria protegida desde el mode usuari.
- Excepció 13: Instrucció protegida. S'intenta executar una instrucció protegida desde el mode usuari.
- Excepció 14: Crida a sistema. Té lloc quan s'executa la instrucció CALLS.

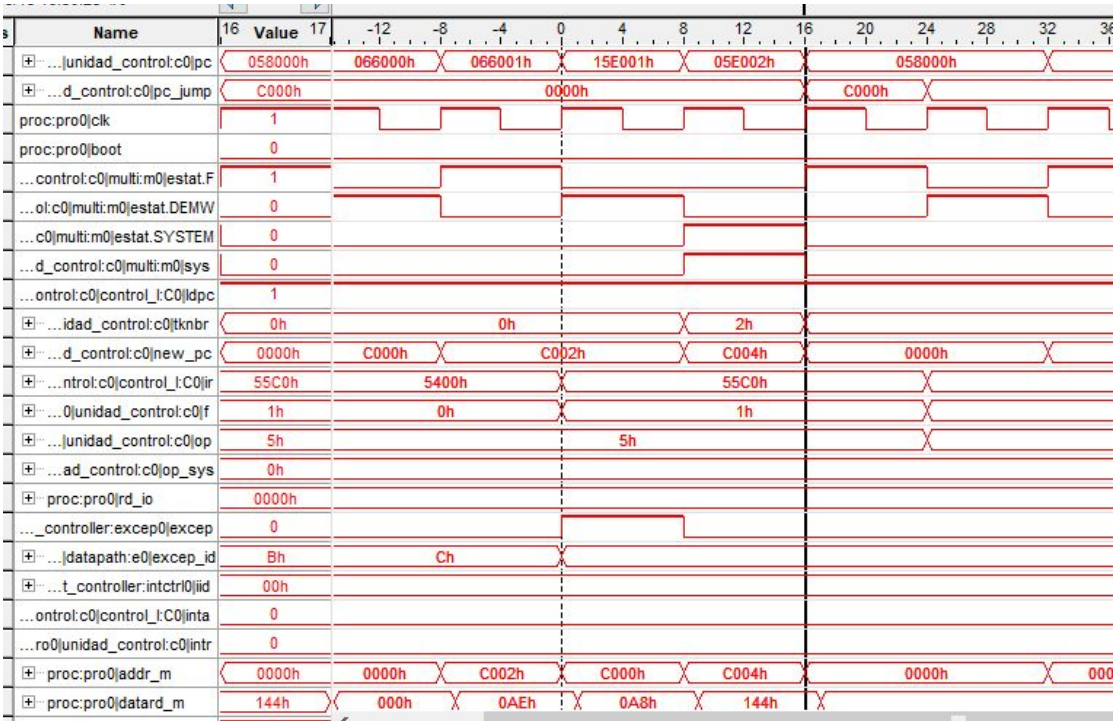

fig. 4.20: *Primeres proves del mode sistema. Accés a memória protegida*

També hem creat els nous senyals d'excepció per a capturar les noves excepcions. Per a la excepció 11 comparem la adreça sol·licitada a memòria durant la etapa DEMW, per a la 13 el senyal es genera al decodificador d'instruccions al igual que la 14.

4.12 TLB (Translation Lookaside Buffer)

En la etapa final implementarem dues TLB al nostre sistema. Aquestes s'hauran d'afegir al mig del bus d'adreces de memòria i faran la traducció d'adreces virtuals a físiques. Quan una traducció no sigui trobada a la TLB, es produirà una excepció per a decidir que s'ha de fer. Les entrades del buffer s'escriuran amb unes instruccions específiques que estaran protegides pel mode sistema (WRPI, WRVI, WRPD, WRVD).

La implementació de les TLB és similar a l'exemple de la documentació, però les escriptures als registres es fan des de un mateix port. Per a indicar quin tipus d'adreça estem escrivint als registres (virtual o física) hem implementat 2 senyals de permís d'escriptura (v_wr i p_wr).

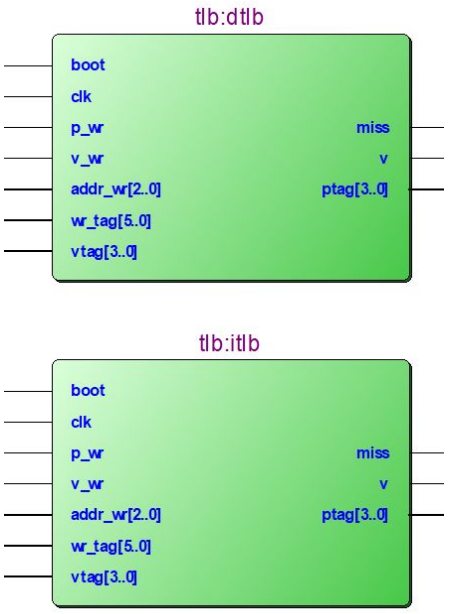

fig. 4.20: *Entitats corresponents a les dues TLB al camí de dades*

A l'hora de testejar els jocs de proves anteriors, hem hagut d'inicialitzar les tlb amb les pàgines necessàries ja que els programes de prova com l'nSnake no tenen programada una rutina d'atenció a les excepcions, així que si una tlb no trobés la traducció, el mecanisme de captura d'aquestes ens portaria a una adreça imprevisible. (Com que inicialitzem tots els registres a 0 en realitat si que podem predir on anirà, que es l'adreça h0000). Un cop inicialitzades les pàgines correctament ens adonem que arribat un moment de l'execució es produïa un miss a la tlb i es produïa el salt al cicle SYSTEM. Vem decidir refer la TLB i aquest problema ens va tornar a aparèixer. Tot i així, els nostres jocs de prova els va passar satisfactoriament.

5.Conclusions

Una de les coses que més valorem d'aquesta assignatura és la autonomia que es dona a cada grup per a implementar el processador. Encara que es guiïn les etapes i s'ajudi en el plantejament, el comportament dels components s'ha d'implementar en base a tots els coneixements que hem anat adquirint durant la carrera.

És un exercici molt interessant de consolidació de coneixements i aclariments dels possibles errors comessos, a més d'una molt bona introducció al llenguatge descriptiu VHDL. Ens permet apropar-nos una mica més a les tasques que ens trobarem al món laboral i amb els terminis ajustats, cosa que ens obliga a decidir ràpidament la estratègia per a implementar els diferents components del processador. Tot i així, el recolzament docent que es rep en moments problemàtics puntuals és necessari per arribar a finalitzar el projecte, ja que a vegades com a novells cometem errors en la planificació que ens compliquen el projecte innecessàriament.

En quant a tecnologies, hem conegut les FPGAs, una tecnologia que permet apropar el disseny de dispositius específics a les mans de gairebé tothom i permeten funcionalitats que fa pocs anys eren gairebé impensables en el dia a dia. Per als estudiants de la nostra especialitat és una oportunitat molt interessant ja que sembla que en un futur, per la computació d'altes prestacions i el disseny de màquines de propòsit específic, aquesta tecnologia pot ser molt útil.

En definitiva, al ser una assignatura de projecte, en quan a adquisició de nous coneixements no és de les assignatures més complexes ja que com hem dit abans es tracta d'aplicar tot l'après durant els estudis. Però ens orienta en un treball en equip i en com gestionar aquest.

6.Possibles treballs futurs

Implementació d'àudio

Una possible millora del sistema que s'està implementant es suport d'àudio adaptant alguna de les controladores disponibles als exemples proporcionats amb la placa de desenvolupament. Com es tracta d'un dispositiu extern, es tractarà com a tal a traves de la controladora de periférics i la interacció necessaria serà portada a terme a traves dels ports d'aquesta.

Implementació d'un SO complet amb emagatzematge persistent (disc dur, targeta sd…)

L'altra amplicació del projecte que s'està portant a terme es la funcionalitat de canvi de context (fer un sistema multiusuari) com a última part de l'assignatura. La idea es crear un quantum i mitjançant un algoritme de round-robin que el processador pugui anar creant i destruint les piles de cada procés i que aquests ocupin durant un temps la cpu.

Si aquesta funcionalitat s'implementa, teóricament es podría instal·lar un petit sistema operatiu al sistema i que aquest controli les capacitats multiusuari.

Implementació de coma flotant

Una altra possible ampliació sería afegir tot el set d'operacions en coma flotant al processador. Per això, apart d'ampliar el nostre banc de registres amb els registres per a coma flotant, s'hauria de crear una nova alu per a totes aquestes operacions, ademés de modificar la lógica de control per a que l'estat de tot el sistema continui sent coherent i poder executar la resta d'operacions sense problemes.