Estudi de viabilitat de l'actualització del HW d'una impressora de llarg format

TREBALL DE FINAL DE GRAU

Autor – Enric Bover Centelles

Director – Benoit Petre

Ponent – Daniel Jimenez Gonzalez

Especialitat – Enginyeria de computadors

Resum

Avui en dia es busca optimitzar el màxim possible els recursos hardware. Actualment en alguns dels projectes d'HP hi ha un malbaratament de l'ús de les memòries que incorporen les FPGA que utilitzen ja que no hi ha cap procés que ajudi a l'usuari a optimitzar-ne l'ús d'aquests recursos. Aquest projecte estudia la viabilitat de crear un disseny HW que pugui ser incorporat en un projecte existent i que fàcilment pugui millorar l'ús de les memòries.

Resumen

Hoy en día se busca optimizar el máximo posible los recursos hardware. Actualmente en algunos de los proyectos de HP hay un desperdicio del uso de las memorias que incorporan las FPGA que utilizan ya que no hay ningún proceso que ayude al usuario a optimizar el uso de estos recursos. Este proyecto estudia la viabilidad de crear un diseño HW que pueda ser incorporado en un proyecto existente y que fácilmente pueda mejorar el uso de las memorias.

Agraïments

Agrair tant al meu director, Benoit Petre, com el meu ponent, Daniel Jimenez per tot el suport que m'han donat durant el desenvolupament d'aquest projecte. Han sigut molt atents i m'han ajudat sempre que han pogut.

Per altra banda agrair també al professorat de GEP com a les persones de Recursos Humans d'HP per ajudar-me tant en les etapes més inicials del projecte com amb els tràmits per poder arrancar el projecte.

Índex

1 Context

1.1 Introducció

Avui en dia tota empresa busca ser competitiva en el mercat, i dins del món de les impressores HP Inc. no és una excepció. Per poder tenir un lloc en el mercat cal oferir un servei o producte a un preu atractiu per al consumidor, les empreses que aconsegueixin suplir les necessitats dels consumidors al millor preu destacaran per sobre de les altres.

En el cas concret d'HP, cada cop que es desenvolupa un nou producte hi participen un gran nombre de grups d'enginyers de diferents especialitats: elèctrics, mecànics, programadors, etc. i tots aquests han de complir una sèrie de requeriments tant en l'àmbit general com l'àmbit de grup.

Aquests requeriments defineixen i acoten com ha de ser el producte en desenvolupament. En el cas d'una impressora defineixen aspectes com la quantitat de pàgines per minut que pot imprimir, mesures del producte, com serà el panell de control, etc. i un dels requeriments que sempre és present en tot projecte és el pressupost. S'hi posen molts esforços en tots els departaments involucrats per aconseguir reduir costos al màxim.

1.2 Context del projecte

Actualment a HP una de les parts més importants de la impressora és la que anomenem *PrintEngine*. El PrintEngine s'encarrega de gestionar tots els recursos hardware directament relacionats amb la impressió: decideix quan moure els motors que desplacen el paper, quan activar i desactivar els *nozzles* per on surt el tint, etc.. Tradueix les pàgines definides en un fitxer a una impressió real.

En un dels projectes d'HP, anomenat MAIA+, es vol utilitzar una arquitectura de PrintEngine molt similar a la d'un projecte anterior anomenat MAIA, però malauradament, es necessita més potència de CPU en la FPGA. És una de les peces claus que formen el PrintEngine del projecte MAIA, una FPGA Zynq-7000, i cal estudiar si aquesta pot ser substituïda per un model d'FPGA ZU5CG (Zynq UltraScale+ 5). S'ha escollit aquest nou model perquè compleix els requeriments proposats per l'empresa.

1.2.1 FPGA

FPGA ve de l'acrònim *Field Programmable Gate Arrays*, que es pot traduir per matriu de portes programable in situ, són uns dispositius semiconductors basats en una matriu de blocs lògics configurables (CLB) connectats mitjançant interconnexions programables. El gran potencial de les FPGA rau en el fet que pots reprogramar-les per aplicacions diferents i t'estalvia la feina d'haver de desenvolupar i optimitzar un circuit integrat específic que només podria executar una sola aplicació.

Un dels recursos presents en les FPGA són les memòries. En el projecte MAIA+ en una primera instància s'havia pensat utilitzar una FPGA on només s'utilitzessin uns tipus de memòria anomenada BlockRAM, però com s'han de complir els requeriments de costos, es va optar per utilitzar la FPGA ZU5CG. Aquesta, a part de tenir també BlockRAM, es caracteritza principalment per la incorporació d'un nou tipus de memòria: les UltraRAM.

1.2.2 BlockRAM

Les característiques principals de les BlockRAM[1] (a partir d'ara abreviades com a BRAM) són les següents:

- Capacitat màxima de 36Kb (Kilobits).
- Possibilitat de configurar cada BRAM com a dues BRAM independents de 18 Kb.
- Ports d'entrada i sortida com *True Dual Port* (TDP, dos ports on cada un pot llegir o escriure) o *Simple Dual Port* (SDP, dos ports on un és d'escriptura i l'altre de lectura).
- Amplada de port configurable.
- Pot suportar 2 freqüències de rellotge diferents, útil per quan es divideix la memòria en dos blocs independents de 18Kb.

1.2.3 UltraRAM

Les característiques principals de les UltraRAMs[1] (a partir d'ara abreviades com a URAM) són:

- Capacitat màxima de 288Kb.
- Els ports es comporten com un superset de SDP.
- Amplada de port no configurable.
- Només suporta una entrada de rellotge

1.3 El problema

En una primera instància es pensava que aquesta nova tecnologia d'URAM no suposaria un problema, però s'ha vist que això no és així. Unes de les problemàtiques que comporta són les següents:

1. Compatibilitat d'URAM amb BRAM.
2. Quantitat de recursos de memòria oferts per la nova FPGA vs els que es necessiten.
    a. Quantitat total de memòria.
    b. Granularitat de la memòria.

És crucial en una primera instància determinar si la feina que fan les BRAM pot ser desenvolupada per una URAM, ja que per aquest estudi depenem d'aquestes memòries. Sabem per proves anteriors realitzades a HP que les URAM poden fer el mateix paper que una BRAM, però no s'ha intentat veure si és viable un canvi tan gran en un projecte sencer com en el de MAIA+.

Per altra banda, existeix el problema de la memòria total disponible i de com està distribuïda. Se sap que el nou model ZU5CG té una quantitat de memòria suficient pel projecte MAIA+, però té una granularitat que dificulta la portabilitat. No és el mateix tenir 8 BRAMs de 36Kb (288Kb en total) que una sola URAM de 288Kb. En el primer cas tenim molta més granuralitat i podem fer més accessos a memòria simultanis (suposant que en ambdós casos utilitzem la mateixa freqüència de rellotge) respecte al segon.

Es van fer proves de síntesis del projecte MAIA+ sobre una ZU6CG (la versió 6 de la família ultraScale) per veure com es distribuïen les memòries i tenir una imatge més clara dels recursos necessaris, com es veu a la Taula 1, aquesta té destinats 25,1 Mb (Megabits) de memòria a BRAM, i com a resultat final de la síntesi es va obtenir que s'utilitzen 276 BRAM de 36Kb i 253 BRAM de 18Kb, això fa un total de 14,49 Mb de memòria de BRAM consumida.

En canvi la ZU5CG té 5,1 Mb de BRAM i 18,0 Mb de URAM, en total te 23.1 Mb de memòria disponible, memòria suficient, però quan es va fer una síntesi del projecte el resultat no era l'esperat, va fallar i en el resultat sortia que s'utilitzaven les 144 BRAMs disponibles i solament 12 URAM, això fa un total de 8,64 Mb consumits, quan el resultat esperat hauria de ser un valor pròxim a 14,49 Mb.

Model	Nº BRAM36	Nº URAM	Nº BRAM36 consumides	URAM consumides	Memòria consumida (Mb)	Memòria total (Mb)
ZU6CG	714	-	402,5	-	14,49	25,1
ZU5CG	144	64	144	12	8,64	23,1

Taula 1: Quantitat de recursos de memòria presents en els dos models FPGA[2]

Amb els resultats obtinguts d'aquestes síntesis es va determinar que calia optimitzar l'ús de recursos de memòria.

Per aquests motius cal fer un estudi de viabilitat sobre com afrontar aquests inconvenients i decidir si aquesta elecció per l'empresa serviria per fer la portabilitat.

1.4 Actors implicats

Tot seguit s'anomenen els actors implicats en aquest projecte:

- Desenvolupador: en aquest projecte només consta un sol desenvolupador, l'Enric Bover Centelles. Ell serà l'encarregat d'estudiar la viabilitat del canvi proposat, fer les proves necessàries i adients per determinar-ho, proposar canvis de l'arquitectura si són necessaris i documentar-ho en el seu Treball de Fi de Grau d'Enginyeria Informàtica.
- Director: és l'encarregat del projecte dins de l'empresa. S'encarregarà d'aconsellar, guiar i ensenyar al desenvolupador com realitzar les seves tasques correctament dins del context i polítiques de l'empresa. També mantindrà contacte amb el ponent.
- Ponent: assessorà al director del TFG per garantir que el treball segueix les normatives corresponents establertes per la universitat i avaluarà les diferents fites del projecte.
- Beneficiaris: l'empresa podrà determinar si és factible aquest canvi, en cas afirmatiu podrà tirar endavant amb el projecte sense haver de canviar els requeriments. I tant si l'estudi determina que és viable com si no, l'empresa tindrà una documentació ben elaborada sobre el comportament de les URAM en el seus dissenys. A més com en aquest projecte s'intentaran optimitzar els recursos de

memòries podrà aplicar els mètodes presentats en aquest treball en futurs projectes o en d'altres ja existents.

2 Justificació de la solució

Com s'ha dit en el context, l'objectiu principal d'aquest projecte consisteix a determinar si és viable l'ús de la ZU5CG en el projecte MAIA+. En ser un projecte en desenvolupament propi de l'empresa on es treballa amb una tecnologia que no s'ha utilitzat fins ara, implica que encara no hi hagi solucions existents per aquest problema. De totes les solucions possibles que es vagin trobant durant el desenvolupament d'aquest Treball de Fi de Grau s'haurà d'escollir i justificar quina d'aquestes és la més adient.

Amb un estudi teòric previ realitzat pel desenvolupador, ja es va determinar que la granularitat de les memòries afectava directament en la viabilitat del canvi i es va determinar que calia optimitzar l'ús de les memòries.

En la Figura 1 es pot observar de forma conceptual i resumida com s'instancien les memòries actualment. Cada cop que es necessita una memòria, es fa una instància de memòria i aquesta és assignada a una BRAM o una URAM físicament.

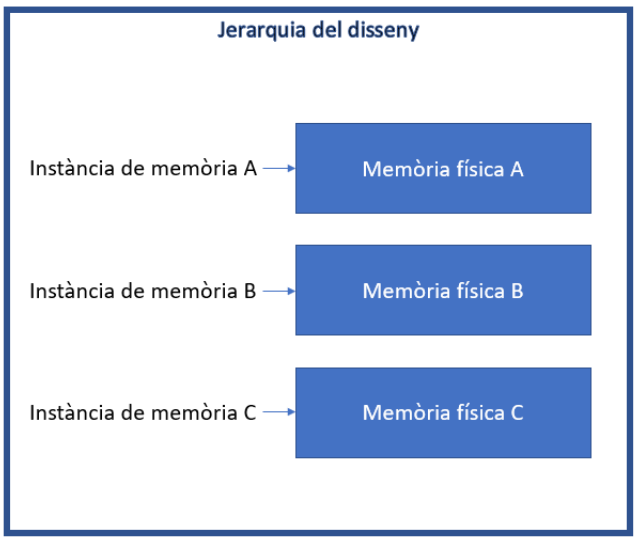

Figura 1: Estructura jeràrquica actual. Elaboració pròpia.

Una de les solucions presentades i més atractiva que va proposar el director consistia en generar un nou mòdul capaç d'optimitzar l'ús de les memòries del projecte. Si ens hi fixem en la Figura 2, s'hi representa de forma simplificada el que s'ha proposat com a solució per optimitzar els recursos. El que fa aquest nou mòdul consisteix en agafar la informació de les instàncies de memòria que hi ha en el projecte i agrupar-les de tal manera que es puguin aprofitar millor.

Aquesta solució ve arran de la següent problemàtica identificada en l'estudi teòric previ: suposem que en un punt de l'arquitectura tenim instanciada en el codi una memòria A que consumeix 6 Kb i en un altre punt de l'arquitectura tenim una altra memòria B que consumeix 4 Kb. A causa de com està definida l'arquitectura en el codi font i al

funcionament del sintetitzador, el resultat seria assignar tant a la memòria A com a la memòria B una BRAM de 18 Kb (es gastarien un total de 36Kb). Com es pot apreciar, s'està desaprofitant més del 50% de la memòria total en aquest cas (10 Kb utilitzats de 36 Kb disponibles).

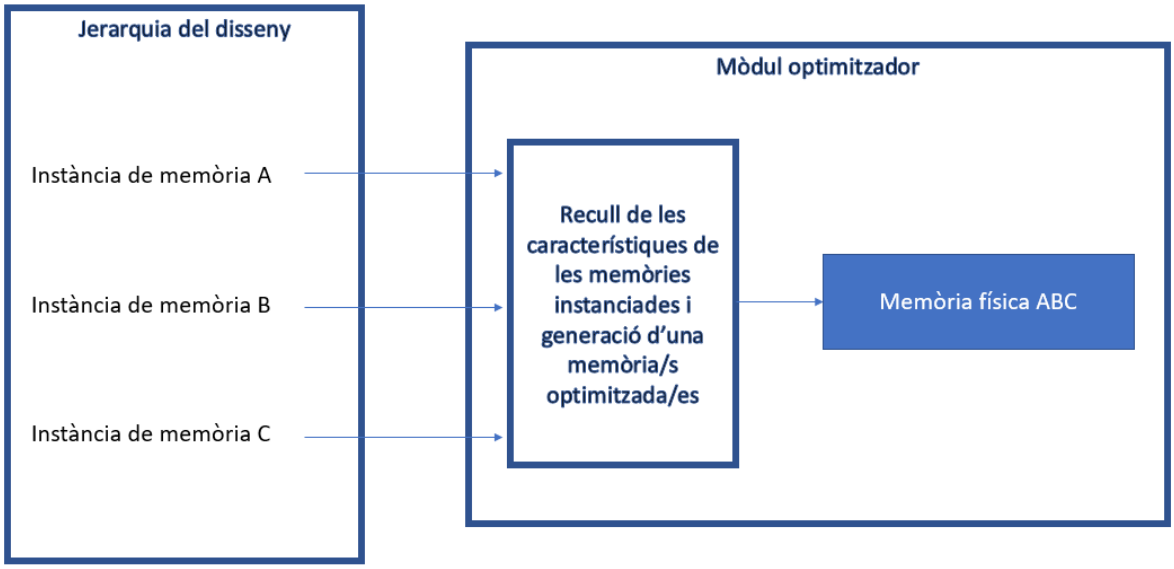

Figura 2: Concepte del nou mòdul a afegir en la jerarquia. Elaboració pròpia.

L'optimització la qual s'encarregarà de fer el nou mòdul consisteix a ajuntar tant la memòria A com la B en una de sola de 18 Kb, d'aquesta manera solament estaríem utilitzant 18 Kb i una sola BRAM (10 Kb utilitzats de 18 Kb disponibles) en comptes de 36 Kb i dues BRAM.

Aquesta solució comporta altres problemes, ja que si s'utilitza la mateixa freqüència d'accés a memòria en ambdós casos, en el primer cas podríem fer dos accessos a memòria en un temps T quan en el segon cas només en podríem fer un de sol amb el mateix temps T.

Un cop desenvolupat aquest mòdul seria convenient afegir-lo en l'arquitectura del projecte MAIA+ i veure com afecta la quantitat de recursos utilitzats i ens donaria una imatge més clara sobre la viabilitat del canvi.

S'ha escollit aquesta solució perquè el mòdul generat no només podria ser aplicat en el marc del projecte que es tracta en aquest Treball de Fi de Grau, sinó que també es podria transportar la mateixa idea a altres projectes de l'empresa.

3 Abast

En aquest apartat es parlarà dels objectius principals i secundaris del projecte, quins requisits determinaran si l'estudi valida o no la viabilitat, els requisits funcionals o no funcionals i quins possibles obstacles o riscos es podran trobar al llarg del desenvolupament d'aquest treball.

3.1 Objectius

L'objectiu principal d'aquest projecte consisteix a validar l'ús de les FPGA de la família UltraScale+ en el projecte MAIA+. A més a més en aquest treball es proposa una optimització de l'ús de les memòries que no només afecta l'àmbit d'aquest projecte, sinó que també podria afectar a altres projectes de l'empresa.

A continuació es mostren els objectius que serviran per emmarcar fins on arribarà aquest projecte i el seu compliment permetran determinar si s'ha assolit una resolució del problema.

- **Estudi teòric de l'arquitectura**
  - Identificar quants recursos de memòria tenim disponibles, quants s'utilitzen i com són utilitzats.
  - Proposta de diferents solucions d'optimització d'aquests recursos.
  - Identificar el millor cas possible d'optimització per cada una de les propostes trobades.
- **Programació del nou mòdul d'optimització de memòries**
  - Determinar requeriments d'aquest mòdul.
  - Dissenyar el mòdul en funció dels requeriments.
  - Testejar el mòdul en diferents casos d'ús i validar-lo (per a memòries del tipus TDP amb un mateix *clock* per cada port).
- **(Objectiu extra) Verificar el funcionament del mòdul en altres contextos**
  - Utilitzar el mòdul ja funcional en altres projectes existents, verificar la seva versatilitat, extreure resultats obtinguts i una valoració d'aquests.

3.2 Requisits funcionals i no funcionals

En el cas que es pugui dur a terme un nou mòdul capaç d'optimitzar l'ús de memòries, aquest ha de complir una sèrie de requisits funcionals i no funcionals:

- **Requisits funcionals:**
  - El nou mòdul ha de simular el funcionament esperat de les memòries que es volen optimitzar.
  - Les memòries beneficiades d'aquest mòdul només podran ser del tipus TDP amb un mateix *clock* en cada port, en cas d'èxit s'intentarà optimitzar per la resta de versions.
  - Capacitat d'optimitzar vàries memòries BRAM en BRAM o en URAM.

    - Capacitat d'unificar memòries de diferents nivells jeràrquics.
- **Requeriments no funcionals:**
    - Portabilitat i usabilitat assequible: que la interfície del nou mòdul que optimitzarà les memòries no difereixi molt del mòdul actual que instancia memòries. D'aquesta manera evitarem haver de fer un gran nombre de canvis en el codi font.
    - Documentació rigorosa del nou mòdul, de tal manera que pugui ser fàcil de mantenir.

3.3 Possibles obstacles i riscos

Existeixen agents que poden posar en perill el desenvolupament d'aquest projecte:

- **Verificació precoç:** podria donar-se el cas que ràpidament es verifiqués si és viable o no l'actualització del HW, en aquest cas el desenvolupador hauria d'intentar ajustar al màxim les optimitzacions i verificar que realment es manté la funcionalitat
- **Inexperiència:** en aquest projecte el desenvolupador té el desconeixement de la infraestructura que utilitza l'empresa per desenvolupar els projectes i falta de coneixement del llenguatge de programació utilitzat (System Verilog). Es podria donar el cas que el projecte s'allargués més del compte a causa del temps d'aprenentatge requerit per poder desenvolupar correctament el treball.
- **Teletreball:** en ser les pràctiques telemàtiques, en cas que existís algun fallo de connectivitat podria alentir el desenvolupament.

4 Metodologia i rigor

Com s'ha explicat anteriorment aquest treball té diferents fases i aquestes segueixen un ordre, és a dir, que no es pot passar a la fase següent fins que no s'ha determinat com a acabada la fase actual. Així doncs, aquesta estructuració del treball s'ajusta molt a la metodologia anomenada *Waterfall*.

La metodologia *Waterfall*[3] consisteix en desenvolupar un projecte de forma seqüencial. Aquesta consta de vàries etapes: 1) Identificació del problema, 2) Anàlisis de les possibles solucions, 3) Disseny de la solució, 4) Elaboració de la solució i 5) Comprovació de la solució. Per començar una etapa cal haver determinat com acabades les etapes anteriors.

Per validar tota la feina realitzada el desenvolupador, el ponent i el director es reuniran 1 cop per setmana i es determinarà si es pot passar o no a la següent etapa.

Eines utilitzades:

- **Git:** git[4] és un programari de sistema de control de versions pensat en l'eficiència i confiabilitat de manteniment de versions d'aplicacions amb una enorme quantitat de fitxers de codi font.
- **NoMachine:** NoMachine et permet connectar-te remotament a qualsevol altra màquina. A l'empresa utilitzen aquesta connexió per connectar-se a ordinadors més potents capaços de generar simulacions i proves de síntesis més ràpidament.
- ***Test bench*:** El desenvolupador s'encarregarà de programar una sèrie de *test benchs* que permetran verificar si el programari desenvolupat té el comportament esperat.

5 Planificació

En aquest apartat es fa la planificació temporal del TFG que consta dels següents apartats: 1) Descripció de les tasques, 2) Estimacions i Gantt i 3) Plans alternatius i obstacles.

5.1 Descripció de les tasques

En aquest apartat es descriuen curosament totes les tasques que s'han previst per realitzar al llarg d'aquest projecte, aquestes es realitzaran durant el període vigent del conveni de cooperació educativa en el qual participa el desenvolupador, des del dia 1 d'abril de 2020 fins al 5 de febrer de 2021, però es preveu que les tasques podran complir-se abans de la data final del conveni. Cal tenir en compte que el desenvolupador treballa a mitja jornada amb dos dies de vacances de lliure elecció al mes.

Els recursos utilitzats per a cada tasca els hem separat en humans i de material, a continuació en la Taula 2 es mostra una llista de tots els recursos previstos pel projecte, a cada un d'ells l'hi hem assignat una abreviació per identificar-los amb més facilitat.

RECURSOS HUMANS	**ABREVIACIÓ**
Desenvolupador	D
Director	DI
Ponent	P
Professor GEP	PG
RECURSOS MATERIALS	
Ordinador (amb els perifèrics necessaris)	O
Internet	I
Microsoft Word	MW
Microsoft Excel	ME
Microsoft PowerPoint	MP
NX Client	NX
GanttProject	G
Zoom	Z
Infraestructura de treball de l'empresa	IT

Taula 2: Llistat de recursos previstos amb la seva abreviació assignada. Elaboració pròpia.

En la Taula 3 que hi ha a continuació es mostra un breu resum de totes les tasques previstes, en els punts 5.1.1 i 5.1.2 es fa una descripció més detallada.

TASCA	**RESUM**	**DEPENDÈNCIES**	**RECURSOS**	**TEMPS (H)**
TGP1	Contextualització i abast	-	D, DI, P, PG, O, I, MW	25
TGP2	Planificació temporal	TGP1	D, DI, P, PG, O, I, MW, G	25
TGP3	Pressupost i sostenibilitat	TGP2	D, DI, P, PG, O, I, ME	25
TGP4	Emmarcar projecte	TGP3	D, O, I, MW, ME, G	20
TGP5	Reunions setmanals	-	D, DI, P, I, O, Z	25
TGP6	Documentació pràctica del projecte	-	D, O, I, MW, ME	70

TASCA	RESUM	DEPENDÈNCIES	RECURSOS	TEMPS (H)
TD1	Familiarització de l'entorn de treball	-	D, O, I, IT, NX	15
TD2	Recerca de documentació	-	D, O, I	20
TD3	Informe de l'arquitectura actual	TD1, TD2	D, O, I, MW, ME, NX, IT	16
TD4	Determinar quanta memòria s'està malbaratant	TD3	D, O, MW, ME	12
TD5	Analitzar comportament del sintetitzador	TD4	D, O, I, NX, IT	12
TD6	Recerca de solucions	TD5	D, O, I, MP	40
TD7	Definir casos d'ús dels mòduls	TD6	D, DI, P, O, I, NX, IT	25
TD8	Validació dels mòduls	TD7	D, O	40
TD9	Disseny dels mòduls	TD8	D, DI, P, O, I, NX, IT	60
TD10	Test dels mòduls	TD9	D, O, I, NX, IT	60
TD11	Documentació mòduls	TD10	D, O, I, NX, IT	30
TD12	Incorporació dels nous mòduls	TD10	D, O, I, NX, IT	60
TD13	Proves de síntesi amb els nous mòduls incorporats	TD12	D, O, I, NX, IT	60
TD14	Informe final de resultats obtinguts	TD13	D, O, I, MW, ME, MP	20
TOTAL				660

Taula 3: Resum de totes les tasques previstes. Elaboració pròpia.

5.1.1 Gestió de projecte

- **TGP1 – Contextualització i abast** (25 hores): Definir context i l'abast del projecte. Recursos: D, DI, P, PG, O, I, MW.
- **TGP2 – Planificació temporal** (25 hores): Planificació temporal del projecte i definició de les tasques a realitzar. Depèn de TGP1. Recursos: D, DI, P, PG, O, I, MW, G.
- **TGP3 – Pressupost i sostenibilitat** (25 hores): Anàlisis de pressupost i sostenibilitat del projecte. Depèn de TGP2. Recursos: D, DI, P, PG, O, I, ME.
- **TGP4 – Emmarcar el projecte** (20 hores): Concatenació del resultat de les 3 tasques anteriors incorporant el feedback rebut per part del director, ponent i tutor de GEP. Depèn de TGP3. Recursos: D, O, I, MW, ME, G.
- **TGP5 – Reunions setmanals** (25 hores): Cada setmana es destinarà entre 30 minuts i 1 hora a reunir-se el desenvolupador amb el ponent o el director o els dos al mateix temps per tal de fer un seguiment de la feina realitzada. Recursos: D, DI, P, I, O, Z.
- **TGP6 – Documentació pràctica del projecte** (70 hores): A mesura que es vagi avançant en la part més pràctica del projecte, paral·lelament s'ha de documentar els resultats obtinguts. Recursos: D, O, I, MW, ME.

5.1.2 Desenvolupament del projecte

- **TD1 – Familiarització de l'entorn de treball** (15 hores): El desenvolupador cal que es familiaritzi amb la infraestructura de treball de l'empresa. Recursos: D, O, I, IT, NX.
- **TD2 – Recerca de documentació** (20 hores): El desenvolupador ha de fer un recull d'informació relacionada amb el funcionament de les FPGA implicades en el projecte, les memòries BRAM i URAM; de com programar correctament amb System Verilog i documentar-se sobre l'ús de Vivado. Recursos: D, O, I.
- **TD3 – Informe de l'arquitectura actual** (16 hores): El desenvolupador ha de generar un informe que reculli l'ús actual de recursos en la FPGA per tal de tenir informació del punt de partida. Depèn de TD1 i TD2. Recursos: D, O, I, MW, ME, NX, IT.
- **TD4 – Determinar quanta memòria s'està malbaratant** (12 hores): El desenvolupador ha de determinar quina quantitat de memòria s'està malbaratant en l'arquitectura, on s'està malbaratant. Depèn de TD3. Recursos: D, O, MW, ME.
- **TD5 – Analitzar comportament del sintetitzador** (12 hores): El desenvolupador ha de determinar quins paràmetres utilitza el sintetitzador per assignar els recursos i com es pot ajudar-lo perquè i optimitzi millor els recursos disponibles. Depèn de TD4. Recursos: D, O, I, NX, IT.
- **TD6 – Recerca de solucions** (40 hores): El desenvolupador ha de pensar i valorar quines opcions existeixen per optimitzar l'ús de memòries i fer-ne un disseny teòric. Depèn de TD5. Recursos: D, O, I, MP.
- **TD7 – Definir els casos d'ús del mòdul** (25 hores): El desenvolupador ha d'acotar quins casos d'ús tindrà el mòdul. Depèn de TD5. Recursos: D, DI, P, O, I, NX, IT.
- **TD8 – Validació dels mòduls** (40 hores): El desenvolupador ha de demostrar mitjançant diagrames temporals que el/s mòdul/s que ha dissenyat podrien formar part de la solució. Depèn de TD6, TD7. Recursos: D, O
- **TD9 – Disseny dels mòduls** (60 hores): El desenvolupador ha de dissenyar els mòduls necessaris per poder dur a terme l'optimització de l'ús de recursos de memòries, aquests es basaran en els dissenys teòrics obtinguts de la tasca TD6 i seguirà les recomanacions fetes pel ponent i el director. Depèn de TD8. Recursos: D, DI, P, O, I, NX, IT.
- **TD10 – Test dels mòduls validats** (60 hores): El desenvolupador ha de dissenyar una sèrie de testos que permetin verificar que el funcionament dels mòduls és l'esperat en els diferents casos d'ús. Depèn de TD9. Recursos: D, O, I, NX, IT.
- **TD11 – Documentació dels mòduls** (30 hores): El desenvolupador ha de documentar la feina realitzada a l'empresa. Depèn de TD10. Recursos: D, O, I, NX, IT.
- **TD12 – Incorporació del nou mòdul** (60 hores): El desenvolupador ha d'incorporar el mòdul/s al projecte MAIA+ utilitzant cada un dels dissenys desenvolupats. Depèn de TD10. Recursos: D, O, I, NX, IT.

- **TD13 – Proves de síntesi amb els nous mòduls incorporats** (60 hores): Aquesta tasca només comptabilitza aproximadament el temps que es tardarà en fer cada un de les síntesis un cop feta la incorporació del nou mòdul. Depèn de TD12. Recursos: D, O, I, NX, IT.
- **TD14 – Informe final de resultats obtinguts** (20 hores): El desenvolupador ha de generar un informe que reculli els rendiments d'optimització que es poden generar utilitzant els nous mòduls i valorar els resultats obtinguts. Depèn de TD13. Recursos: D, O, I, MW, ME, MP.

5.2 Diagrama de Gantt

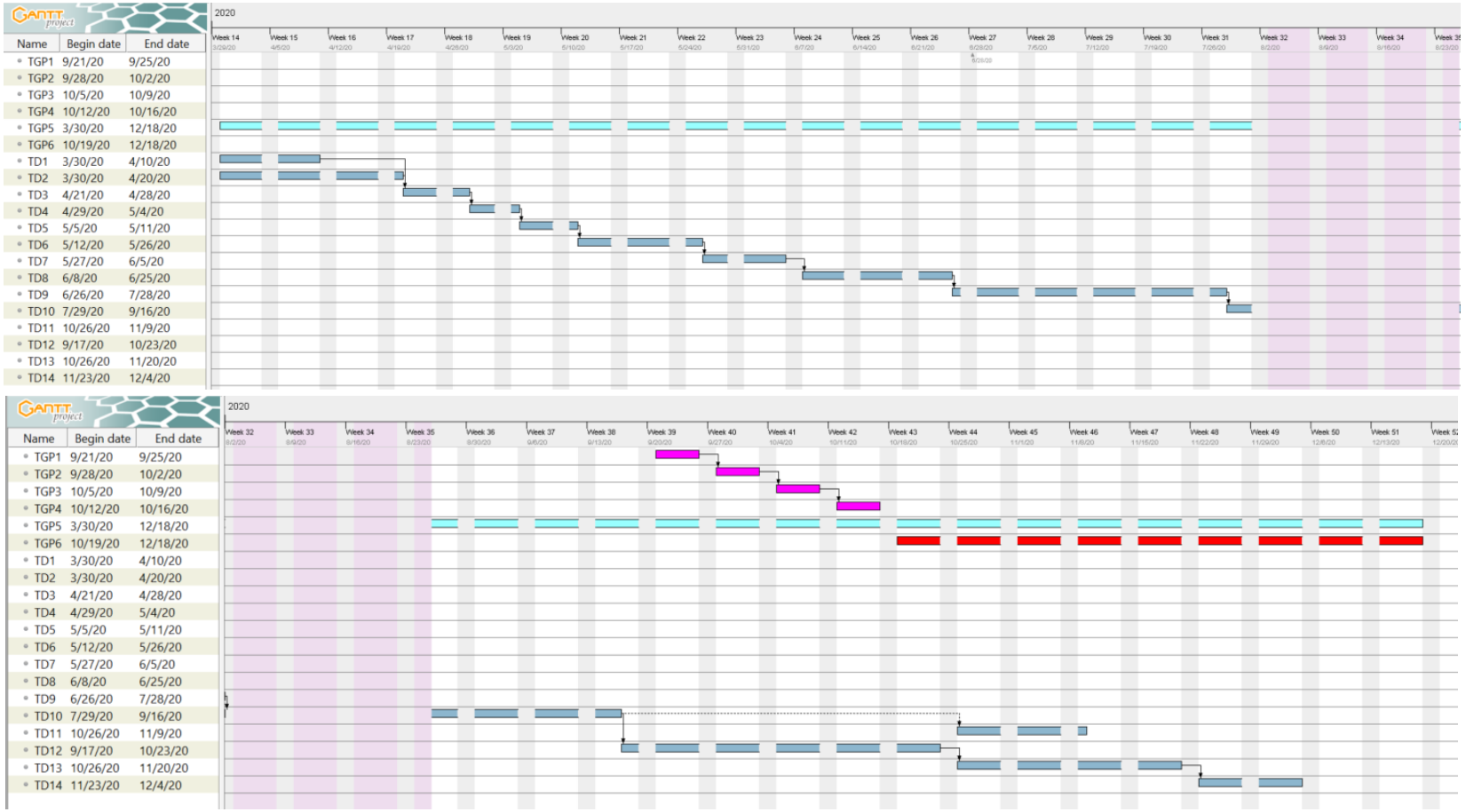

Figura 3: Diagrama de Gantt. Elaboració pròpia.

En la Figura 3 es mostra el diagrama de Gantt. S'ha fet una estimació sobre les dates en les quals es realitzen les tasques. En color blau cian tenim les reunions setmanals, en color lila fluorescent tenim les tasques assignades a GEP, en color vermell són les tasques relacionades amb la documentació de la part pràctica del treball i en color blau grisós, tenim les tasques relacionades amb el desenvolupament del projecte. Les zones marcades en color rosat són els dies de vacances.

5.3 Gestió del risc: Plans alternatius i obstacles

A continuació s'enumeren els riscos que poden sorgir durant el desenvolupament del projecte, quin impacte tindrien i quins plans alternatius es plantegen.

5.3.1 Verificació precoç

Existeix la possibilitat que l'objectiu del projecte pugui determinar-se en una de les etapes inicials i per tant que el projecte final sigui massa concís. Les probabilitats que això passi són bastant elevades, s'estima que les probabilitats que passi ronden el 50%. Amb l'estudi previ que va fer el desenvolupador de l'arquitectura ja se'n podia extreure informació suficient per a determinar si l'empresa qualificava el projecte com a viable o inviable.

Per a solucionar aquesta problemàtica es va decidir generar el mòdul unificador de l'ús de les memòries, ja que aquest té molta sortida per altres projectes i permet quantificar a quin nivell de viabilitat es troba realment el canvi del HW.

5.3.2 Inexperiència amb les tecnologies

En aquest projecte s'usa una infraestructura, un llenguatge de programació i unes tecnologies amb les quals el desenvolupador no està familiaritzat. Les probabilitats que el desenvolupador trobi problemes en aquest àmbit són del 100%. En aquest cas no es pot preveure amb facilitat una solució, per tant el seu impacte no és mesurable.

El que s'ha pensat per evitar aquest tipus de contratemps ha sigut afegir més hores a les tasques descrites, d'aquesta manera el marge de maniobra és més gran i permet més flexibilitat.

5.3.3 Teletreball

A causa de la situació actual de pandèmia tot el projecte s'està desenvolupant telemàticament, aquesta situació comporta certs riscs, com per exemple problemes de connectivitat. Les probabilitats que hi hagi problemes de connectivitat són molt baixes, aproximadament rondant per sota del 5%.

En el cas que es donin problemes de connectivitat el desenvolupador té la possibilitat de connectar-se des d'altres punts de la xarxa. En cas que fos problema de la xarxa de l'empresa aleshores no existeix una solució possible, ja que s'escapa del rang d'acció del desenvolupador, però es té constància que els pocs cops que l'empresa ha tingut caigudes a la xarxa, aquestes s'han solucionat en menys de 24 hores.

6 Pressupost

En aquest apartat es calcula quin és el pressupost d'aquest projecte. Per fer el següent càlcul s'han considerat els costos humans i materials que hi ha implicats a cada una de les tasques definides en la planificació.

6.1 Costos dels recursos humans

En aquest apartat s'identifiquen els costos per hora de cada un dels rols implicats en aquest projecte. En la Taula 4 es mostra el cost per hores de cada un dels recursos humans implicats. Pel sou brut del director s'ha fet una aproximació del que cobra una persona del seu rang en l'empresa. Pel del desenvolupador s'ha determinat el sou brut en funció del conveni de cooperació educativa que té associat. En el cas tant del ponent com del professor de GEP s'ha agafat com a cost base el de professor/a titular d'universitat present en el document "Taules retributives del personal docent i investigador" de l'any 2019, un total de 2700,54 € restant-li les deduccions mensuals fan un total de 2639,65 €, suposant que treballa 160 hores mensuals. El cost de la seguretat social és un 30% del sou brut.

ROL	SOU BRUT	SOU BRUT + SEGURETAT SOCIAL
Desenvolupador	9,00 €/h[5]	11,70 €/h
Director	31,25 €/h	40,63 €/h
Ponent	16,50 €/h[6]	21,45 €/h
Professor de GEP	16,50 €/h	21,45 €/h

Taula 4: Càlcul de costos dels recursos humans. Elaboració pròpia.

En la Taula 5 es mostra un càlcul aproximat de totes les hores dedicades per cada un dels rols per cada una de les tasques descrites en la planificació i el cost que suposa en el projecte.

TASCA	DESENVOLUPADOR (HORES)	DIRECTOR (HORES)	PONENT (HORES)	PROFESSOR GEP (HORES)	COST (€)
TGP1	25	1	1	1	376,03 €
TGP2	25	1	1	1	376,03 €
TGP3	25	1	1	1	376,03 €
TGP4	20	-	-	-	234,00 €
TGP5	25	25	25	-	1.844,50 €
TGP6	70	-	-	-	819,00 €
TD1	15	-	-	-	175,50 €
TD2	20	-	-	-	234,00 €
TD3	16	-	-	-	187,20 €
TD4	12	-	-	-	140,40 €
TD5	12	-	-	-	140,40 €
TD6	40	3	3	-	654,24 €
TD7	25	-	-	-	292,50 €
TD8	40	-	-	-	468,00 €
TD9	60	-	-	-	702,00 €

TASCA	DESENVOLUPADOR (HORES)	DIRECTOR (HORES)	PONENT (HORES)	PROFESSOR GEP (HORES)	COST (€)
TD10	60	-	-	-	702,00 €
TD11	30	-	-	-	351,00 €
TD12	60	-	-	-	702,00 €
TD13	60	-	-	-	702,00 €
TD14	20	-	-	-	234,00 €
TOTAL	660	31	31	3	9.710,83 €

Taula 5: Costos dels recursos humans. Elaboració pròpia.

6.1 Costos dels recursos materials

En aquest apartat es numeren els diferents costos materials del projecte, tant costos directes com indirectes.

6.1.1 Costos directes

Hardware

El hardware es calcula que s'amortitza fins al cap de 4 anys de vida útil. En la Taula 6 es mostra quin equip hardware es necessita pel projecte, quin és el seu cost total i quina la seva amortització que es farà durant els 9 mesos que s'ha previst que s'usarà.

HARDWARE	COST TOTAL	AMORTITZACIÓ
Portàtil HP EliteBook 840 G5	1.820,38 €[7]	341,32 €
Monitor LG 22M45HQ-B 22" LED	112,00 €[8]	21,00 €
Ratolí Logitech M90	5,99 €	1,12 €
Suscripció Servidors NoMachine	2.961,69 €/any[9]	2.221,27 €
TOTAL		2.584,71 €

Taula 6: Costos del hardware utilitzat. Elaboració pròpia.

Software

En la Taula 7 es mostra el software i les llicencies necessàries per aquest projecte.

SOFTWARE	COST MENSUAL	COST TOTAL
Llicencia Microsoft 365	10,50 €[10]	94,50 €
Github Enterprise	17,78 €[11]	160,02 €
TOTAL		254,52 €

Taula 7: Costos de les llicencies

6.1.2 Costos indirectes

En els costos indirectes tenim en compte el consum d'energia dels dispositius i del cost de la connectivitat a internet. En la Taula 8 es mostren els costos de consum elèctric de l'ordinador i del monitor. S'ha agafat com a cost mitjà de l'electricitat són uns 0,15 €/kWh i s'ha determinat que les hores d'ús dels recursos són iguals a les hores de treball del desenvolupador, és a dir, 502 hores.

MATERIAL	CONSUM PER HORES	CONSUM TOTAL	COST TOTAL
Portàtil HP EliteBook 840 G5	150W	75,300 kWh	14,85 €
Monitor LG 22M45HQ-B 22" LED	21W	10,542kWh	2,08 €
TOTAL			16,93 €

Taula 8: Costos del consum elèctric. Elaboració pròpia.

En la Taula 9 tenim el cost total que suposa la connectivitat a internet durant els 9 mesos.

FONT	CONSUM MENSUAL	COST TOTAL
Internet	38,00 €	342,00 €

Taula 9: Costos de la connectivitat a internet. Elaboració pròpia.

6.2 Cost total

COSTOS HUMANS	COSTOS DIRECTES	COSTOS INDIRECTES	COST TOTAL
9.710,83 €	2.839,23 €	358,93 €	12.908,99 €

Taula 10: Costos totals sense contingència. Elaboració pròpia.

A causa de la naturalesa del projecte els únics imprevistos són temporals i aquests ja s'han tingut en compte en el càlcul de la contingència que hi ha a continuació. S'ha decidit afegir una contingència del 15%, aquesta afectaria les hores de treball del desenvolupador, director, ponent, llicències software i al consum dels costos indirectes. A la Taula 11 es poden veure els càlculs de contingència arrodonits a l'alça i a la Taula 12 es mostren els resultats finals dels costos.

RECURS	HORES EXTRES	CONTINGÈNCIA
Humans		
Desenvolupador	99	1.158,30 €
Director	5	203,15 €
Ponent	5	107,25 €
Software		
Llicencia Microsoft 365	99 (4 mensualitats)	42,00 €
Github Enterprise	99 (4 mensualitats)	71,12 €
Indirectes1284,64		
Portàtil HP EliteBook 840 G5	99	2,23 €
Monitor LG 22M45HQ-B 22" LED	99	0,32 €
Internet	99 (4 mensualitats)	152,00 €
TOTAL		1.736,37 €

Taula 11: Càlculs de contingència. Elaboració pròpia.

COST TOTAL	CONTINGÈNCIA	COST TOTAL + CONTINGÈNCIA
12.908,99 €	1.736,37 €	14.645,36 €

Taula 12: Cost final amb la contingència. Elaboració pròpia.

6.3 Control de gestió

A causa a la naturalesa del projecte no existeix un risc el suficientment gran com perquè generi una desviació més gran que l'esperada. Es preveu que s'allarguin les tasques

esmentades en la planificació i el seu impacte econòmic ja es té en compte dins de les contingències, però no s'espera que es necessitin més recursos materials dels ja esmentats.

Per poder comptabilitzar aquestes desviacions el desenvolupador té un diari on apunta quines tasques ha fet cada dues setmanes, quins avenços ha realitzat i quines conclusions ha extret. Si es compara aquest diari amb el diagrama de Gantt que es mostra a la planificació es pot extreure informació suficient que determini quina ha estat la desviació real del pressupost.

Per fer aquest càlcul de la desviació es plantegen les següents fórmules. Aquestes s'han d'aplicar cada cop que s'acaba una de les tasques.

- Desviació dels costos humans**:** Costos humans estimats – Costos reals
- Desviació dels costos de Software: Cost software estimat – Cost software real
- Desviació dels costos indirectes: Cost indirecte estimat – Cost indirecte real

7 Informe de sostenibilitat

En aquest apartat s'avalua el nivell de sostenibilitat del projecte en tres àmbits: l'econòmic, l'ambiental i el social. Amb la informació donada podem extreure conclusions dels dos primers àmbits fàcilment, però en el cas del social és una reflexió més personal.

En l'àmbit econòmic aquest projecte només busca determinar si es pot utilitzar un hardware nou en un projecte de l'empresa. En el cas que sigui favorable sabem que no s'hauran de modificar els requisits generals de pressupost del projecte i per tant no s'encarirà la seva posada en producció, en cas contrari el més probable serà que es busqui una altra peça hardware que pugui desenvolupar la funció i a la vegada encarirà el projecte.

En l'àmbit ambiental, des d'un primer moment s'ha enfocat aquest estudi de forma genèrica, d'aquesta manera les conclusions que s'extreguin d'aquesta investigació podran ser aplicables directament en altres projectes de l'empresa, el que implica que no serà necessari realitzar un estudi similar i per tant no es tornaran a donar els costos tant econòmics com ambientals d'aquest projecte. També cal tenir en compte que no s'estan usant recursos nous, és a dir, que s'està usant una infraestructura ja muntada per l'empresa, el que implica que no s'ha hagut d'invertir directament en aquest projecte i per tant la petjada ecològica no té tant d'impacte.

En l'àmbit social aquest projecte m'enriquirà molt com a enginyer, ja que adquiriré habilitats tècniques, metodologies de treball i habilitats socials. Aquest projecte és una oportunitat tant professional com personal pel meu futur, ja que per primer cop puc participar en un projecte d'enginyeria dins del món real. Aquest treball no només m'enriquirà a mi com a persona, sinó que amb els resultats obtinguts podré facilitar la feina d'investigació dels altres treballadors en el desenvolupament de nous projectes en el futur.

7.1 Projecte posat en producció

7.1.1 Dimensió ambiental

Com es pot veure en els recursos necessaris del projecte i tal com es diu a la introducció d'aquest apartat, tots els recursos materials són reutilitzats, no ha calgut comprar material nou per al seu desenvolupament i el consum elèctric estimat no és molt elevat, per tant considero que l'impacte ambiental del projecte és l'esperat.

7.1.2 Dimensió econòmica

La dimensió econòmica del projecte està quantificada en l'apartat de pressupostos, on es detalla curosament quins costos hi ha associat a cada tasca, quin material es necessita i quin és el seu consum. També es tenen en compte contingències.

7.1.3 Dimensió social

El projecte aporta un nou coneixement a l'empresa, el que implica que els seus resultats tindran una repercussió directa en els treballadors directament implicats en l'àmbit d'aquest projecte. Per altra banda el desenvolupador haurà desenvolupat unes aptituds determinants pel seu futur.

7.2 Vida útil

Com que no hi ha cap solució existent no podem comparar la solució proposada amb cap altre en cap de les tres dimensions. El que si podem afirmar és la necessitat d'aquest projecte dins l'àmbit de l'empresa, ja que les conclusions extretes i el treball realitzat ajudaran en els futurs projectes i podran tenir en compte les noves tecnologies estudiades i saber quines implicacions tenen.

8 Estat de l'art

Per motius de protecció de dades no es pot mostrar informació que pugui revelar com està implementada l'arquitectura d'un projecte concret dins de l'empresa, per tant només es pot expressar les dades extretes sobre l'ús de hardware qualitativament.

Els primers passos que el desenvolupador va fer abans de començar a dissenyar el mòdul unificador va ser estudiar l'ús de les memòries d'un projecte dins de l'empresa d'HP. En aquest cas en concret s'estava estudiant canviar l'FPGA del projecte basada en memòries BRAM a una basat en memòries BRAM i URAM.

Les proves de síntesis que es van realitzar van determinar que tal com estava dissenyada l'arquitectura del projecte, aquesta no tenia en compte l'opció d'inferir URAM, per tant els recursos d'URAM eren infrautilitzats. Per altra banda també es va detectar que en molts casos en què s'utilitzava una memòria BRAM, només s'utilitzava una part de la memòria, en molts casos s'utilitzava menys del 50%, en comptes d'utilitzar-la tota.

La proposta que es va fer fora la de dissenyar un mòdul capaç d'unificar múltiples memòries en una de sola, d'aquesta manera s'aprofitaria millor la quantitat de memòria disponible tant per BRAM com per URAM. El més interessant d'aquesta proposta és que no només serviria per a aquest projecte en concret, sinó que fàcilment es podria portar a altres projectes.

En el cas del projecte en qüestió es va descartar l'opció de canviar l'FPGA, ja que amb l'anàlisi de la síntesi es va determinar que la quantitat de recursos disponibles és molt justa per la quantitat de modificacions que fan falta per dur a terme el canvi, i en el cas que es pogués incorporar el mòdul unificador en els diferents punts d'interès, res assegurava que es pogués realitzar la síntesi.

En els següents apartats es descriu quin ha estat el procés de disseny del mòdul unificador, com s'ha implementat, com s'ha verificat el seu funcionament, quins resultats s'han obtingut en casos reals i es conclou si aquesta modificació és viable.

9 Disseny del mòdul unificador

En aquest apartat s'exposa tot el procés de disseny del mòdul optimitzador, que a partir d'ara es dirà unificador, i es justifiquen totes les decisions preses pel seu desenvolupament. Tot aquest desenvolupament s'ha basat principalment en els requeriments descrits en l'apartat 3.2.

9.1 Introducció al disseny

El que es vol aconseguir amb aquest mòdul és reduir la necessitat de 2 o més memòries a una de sola, unificant-les de forma lògica, de tal manera que s'aprofitarà millor l'espai de memòria a canvi d'altres recursos.

El disseny original està format per una arquitectura paral·lela amb múltiples memòries, en canvi, el disseny proposat té una arquitectura seqüencial amb una sola memòria. És evident que el nou disseny, si vol aconseguir el mateix nombre de peticions per cicle que l'anterior, haurà de treballar amb una freqüència de rellotge més elevada mentre que la resta de l'arquitectura fora del mòdul unificador seguirà treballant a la freqüència de rellotge original. A partir d'ara, quan es parli de rellotge ràpid s'estarà referint al rellotge utilitzat en el mòdul unificador i, quan es parli de rellotge lent, es referirà al rellotge de la resta de l'arquitectura.

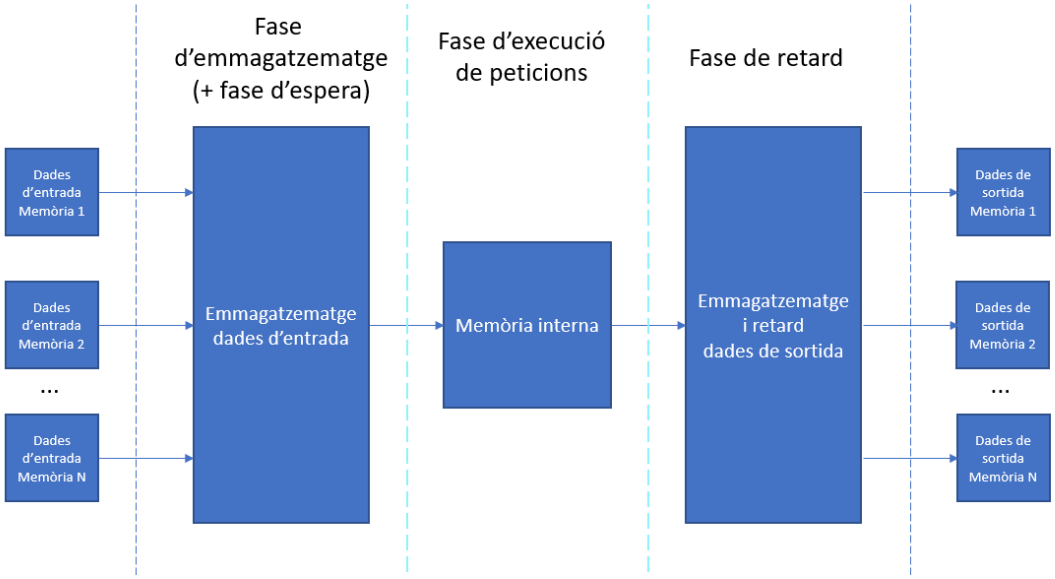

Figura 4: Fases del mòdul unificador. Elaboració pròpia.

A la Figura 4 es mostren les 4 fases del mòdul unificador. Primer hi haurà una fase d'emmagatzematge de les peticions, a continuació hi ha una fase d'execució i per últim hi ha una fase de retard de les dades de sortida.

Durant la fase d'emmagatzematge es guarda les peticions que arriben. Aquesta fase només dura un cicle de rellotge ràpid independentment de la quantitat de memòries que

es vulguin unificar. En el cas que una petició no sigui la primera a ser executada, aquesta haurà d'esperar-se i es dirà que la petició es troba en fase d'espera.

En la fase d'execució de peticions s'executen les peticions guardades en la fase d'emmagatzematge. Per cada petició guardada es necessita 1 cicle de rellotge ràpid per ser executada.

Per últim, la fase de retard guarda les dades de sortida de la memòria interna el temps suficient perquè siguin expulsades en el moment esperat. La quantitat de cicles que una dada està en aquesta fase dependrà del nombre de peticions i de l'ordre en què s'han executat les peticions. Aquesta fase només és útil quan la petició de la fase d'execució ha sigut una lectura, en cas d'escriptura no és rellevant la informació que surt de la memòria interna.

9.2 Diagrama temporal BRAM

És important primer estudiar el diagrama temporal d'una memòria BRAM per poder adaptar la capa de gestió de peticions al seu comportament i poder simular el mateix comportament que tindria amb l'arquitectura paral·lela. Una BRAM pot tenir diferents diagrames temporals en funció de la seva configuració:

- WRITE_FIRST: en cas de lectura la dada que es guarda en memòria es mostra al cicle següent a la sortida de la memòria. En cas d'escriptura es mostra la dada escrita a la sortida de la memòria al cicle següent.
- READ_FIRST: sempre es mostra a la sortida la dada guardada en l'adreça de memòria de la petició.
- NO_CHANGE: la sortida de la memòria es manté amb el valor de l'última petició de lectura.

En el cas del projecte cal assegurar que les peticions d'escriptura s'executen en el mateix cicle que es reben i que el resultat d'una petició de lectura està disponible a la sortida de la memòria al cicle següent després de rebre la petició.

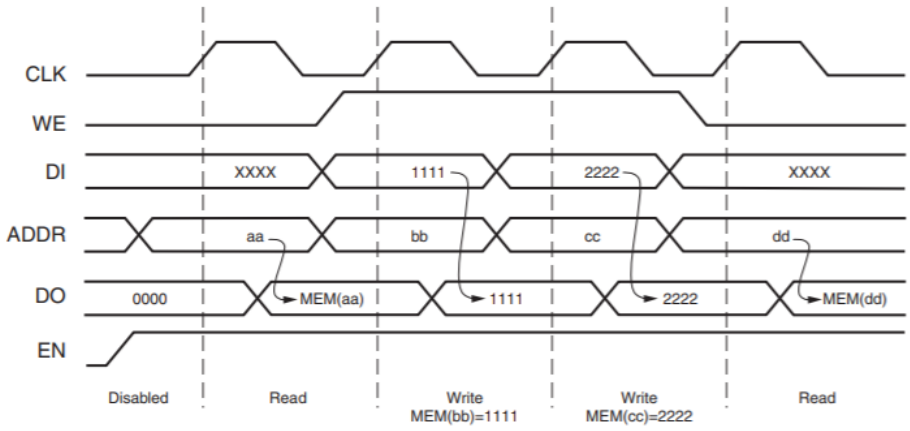

Figura 5: Exemple de diagrama temporal d'una memòria BRAM en configuració WRITE_FIRST. Font: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf

En la Figura 5 es mostra un exemple de diagrama temporal de BRAM en configuració WRITE_FIRST, en la Figura 6 es mostra un diagrama temporal de BRAM en configuració READ_FIRST i per últim, a la Figura 7, hi ha el diagrama temporal en configuració NO_CHANGE, aquests permet entendre millor el flux de dades.

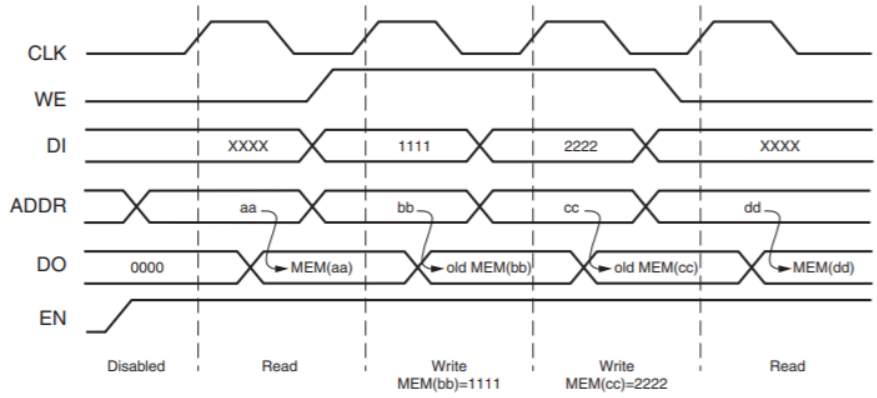

Figura 6: Exemple de diagrama temporal d'una memòria BRAM en configuració READ_FIRST. Font: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf

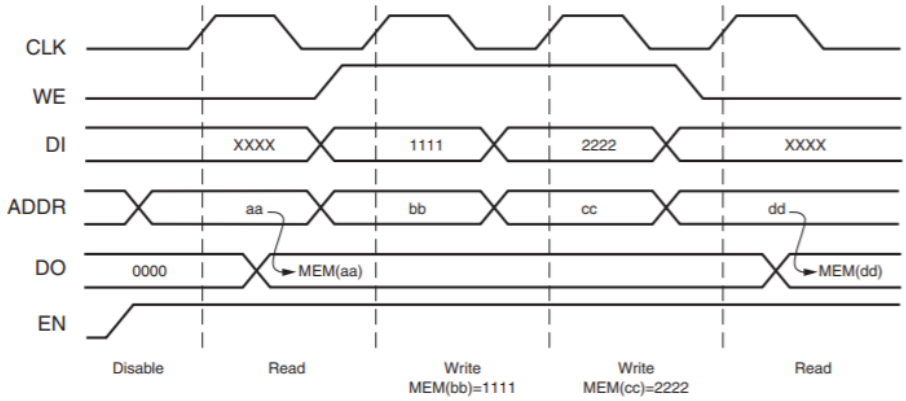

Figura 7: Exemple de diagrama temporal d'una memòria BRAM en configuració NO_CHANGE. Font: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf

Llegenda dels senyals de les Figures 5, 6 i 7.

- CLK: Freqüència del rellotge.
- WE: Bit per seleccionar lectura o escriptura.
- DI: Dada d'entrada.
- ADDR: Adreça de memòria de la petició.
- DO: Dada de sortida.
- EN: Bit per habilitar o deshabilitar el processament de peticions.

En les 3 Figures es mostra un exemple en què arriba una petició de lectura seguida de dues peticions d'escriptura i per última una altra petició de lectura.

9.3 Diagrama temporal de la memòria interna del mòdul unificador

En aquest apartat i en la resta s'estudia el cas en què s'unifiquen 3 memòries, però el mòdul té com a objectiu descriure un comportament genèric i per tant també ser capaç d'unificar i adaptar-se a les diferents quantitats de memòries que l'usuari l'indiqui d'unificar, aquests aspectes es comenten en l'apartat d'implementació.

A les figures 8, 9 i 10 es mostren diagrames temporals que corresponen a exemples del comportament de 3 memòries BRAM diferents (A, B i C) amb una mida de 256 paraules de 4 bits. L'objectiu és veure com s'unifiquen aquestes 3 memòries. S'utilitzaran aquests diagrames com a base per explicar el disseny.

En els diagrames temporals de les memòries A, B i C les peticiones arriben sempre just després d'un flanc ascendent i no comencen a ser processades fins al flanc ascendent següent. Remarcar que en el cas de lectura la dada d'entrada no és rellevant i s'ha marcat com a valor indeterminat, el mateix passa amb la dada de sortida en les peticions d'escriptura.

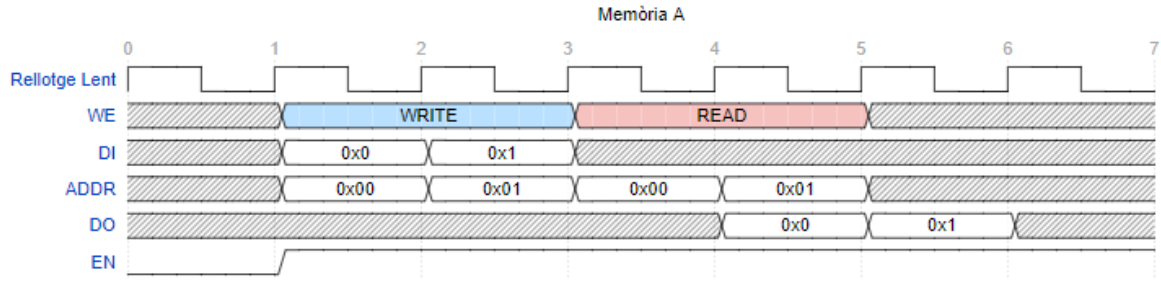

Figura 8: Peticions a la memòria A. Elaboració pròpia.

Adreça	Flanc 1	Flanc 2	Flanc 3	Flanc 4	Flanc 5
0x00	X	X	0x0	0x0	0x0
0x01	X	X	X	0x1	0x1

Taula 13: Estat de la memòria A. Elaboració pròpia.

En la Taula 13 es mostra l'estat de la memòria A des del flanc ascendent 1 fins el 5, aquest està basat en les peticions de la Figura 8. En el flanc 2 es comença a processar la primera petició d'escriptura a l'adreça 0x00 amb el valor 0x0. És entre el flanc ascendent 2 i el 3 que es guarda la dada. En el flanc ascendent 3 es pot assegurar que s'ha guardat la dada de la primera escriptura i es comença a processar la segona petició d'escriptura a l'adreça 0x01 amb el valor 0x1. Entre el flanc 3 i 4 s'actualitza la adreça de memòria.

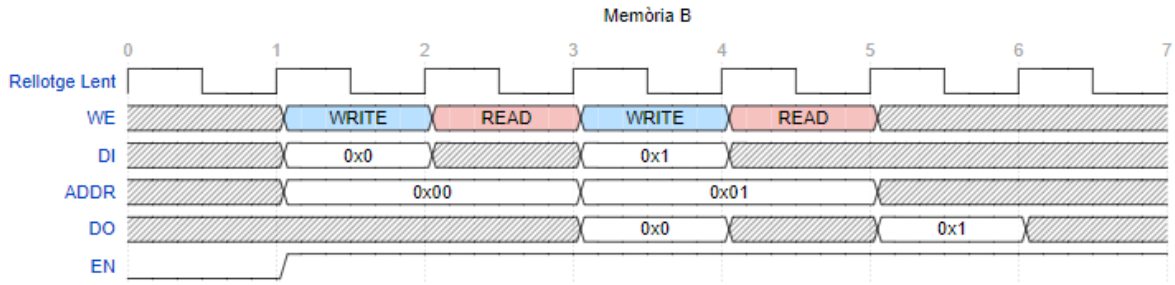

Figura 9: Peticions a la memòria B. Elaboració pròpia.

Adreça	Flanc 1	Flanc 2	Flanc 3	Flanc 4	Flanc 5
0x00	X	X	0x0	0x0	0x0
0x01	X	X	X	X	0x1

Taula 14: Estat de la memòria B. Elaboració pròpia

En la Taula 14 es mostra l'estat de la memòria B des del flanc ascendent 1 fins el 5, aquest està basat en les peticions de la Figura 9. Les peticions d'escriptura a l'adreça 0x00 i 0x01 es comencen a processar en el flanc ascendent 2 i 4 respectivament i, com es pot veure a la taula juntament amb el diagrama temporal de la Figura 9, aquestes ja es poden llegir just al flanc ascendent següent, en aquest cas el 3 i el 5.

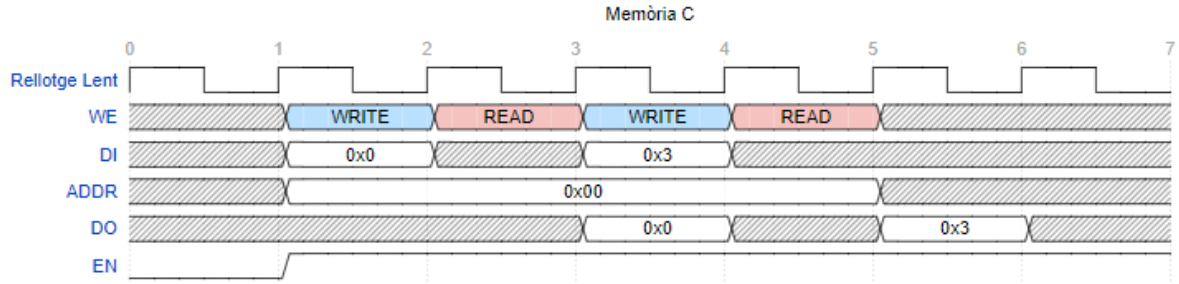

Figura 10: Peticions a la memòria C. Elaboració pròpia.

Adreça	Flanc 1	Flanc 2	Flanc 3	Flanc 4	Flanc 5
0x00	X	X	0x0	0x0	0x3

Taula 15: Estat de la memòria C. Elaboració pròpia.

En la Taula 15 es mostra l'estat de la memòria C des del flanc ascendent 1 fins el 5, aquest està basat en les peticions de la Figura 10. Les peticions d'escriptura arriben al flanc ascendent 2 i 4 i com es pot veure en el diagrama la dada ja es pot assegurar que està guardada en memòria just al flanc ascendent següent, en aquest cas en el flanc 3 i 5 respectivament.

A la Figura 11 es mostren tots els diagrames temporals de les Figures 8, 9 i 10 i a més a més s'ha afegit el diagrama temporal de la memòria interna del mòdul unificador. S'han enumerat els flancs ascendents del rellotge ràpid. Els senyals de les memòries A, B i C

tenen el sufix "_MEM_A", "_MEM_B" i "_MEM_C" respectivament. En el WE de la memòria D cada color: blau, verd i vermell, representa una petició de la memòria A, B i C, respectivament i s'han seqüencialitzat. WR1A indica que és la primera petició d'escriptura de la memòria A, en canvi RD2B indica la segona petició de lectura de la memòria B.

En la Figura 11 es mostren 5 punts d'interès:

1- Requadre vermell: Les 3 primeres peticions es realitzen als flancs ascendents 3, 4 i 5, però realment es podrien començar a processar cada una d'elles un cicle abans (flanc ascendent 2, 3 i 4 respectivament) però, com s'ha esmentat anteriorment, existeix una fase d'emmagatzematge en el disseny que dura un cicle i que s'encarrega de guardar totes les peticions en registres.
2- Requadre vermell: Si es vol enviar les peticions guardades de manera seqüencial implica que es necessita un senyal de control que s'encarregui de seqüencialitzar les peticions guardades. La petició executada passarà a fase d'execució mentre que la resta romandran en fase d'espera.
3- Requadre vermell: El senyal ADDR + OFFSET és el senyal de l'adreça de la memòria interna on s'executa la petició. Com es pot apreciar, aquesta no coincideix sempre amb l'adreça de l'arquitectura paral·lela. Això és degut al fet que s'ha assignat un espai de memòria de la memòria interna per cada una de les memòries unificades.
4- Requadre vermell: En el que dura 1 cicle de rellotge lent en tenim 3 de rellotge ràpid. En aquest cas és així perquè abans que les dades de l'entrada s'actualitzin es necessita un cicle per guardar les peticions on simultàniament s'executarà l'última petició del cicle de rellotge lent anterior i, un segon i tercer cicle per executar la 1a i 2a petició del cicle de rellotge lent actual. Si hi hagués 4 memòries unificades es necessitaria 4 cicles de rellotge ràpid per cada 1 de rellotge lent, el creixement de la freqüència del rellotge ràpid és lineal.
5- Requadre blau: Les dades de sortida de les peticions de lectura RD2A, RD2B i RD2C estan disponibles al flanc ascendent 13, 14 i 15 del rellotge ràpid respectivament, però tal com s'indica, s'espera que aquestes estiguin totes disponibles al flanc ascendent 16 del rellotge ràpid. D'aquí el perquè de la necessitat d'afegir una fase de retard de les dades de sortida. A més a més en aquesta fase s'haurà de dissenyar una arquitectura que sigui capaç de convertir la sortida seqüencial de la memòria interna en una sortida paral·lela.

Figura 11: Diagrama temporal de la memòria interna del mòdul unificador i de les 3 memòries que es volen unificar. Elaboració pròpia.

9.4 Gestió de les dades d'entrada i sortida

Com hem vist en l'apartat anterior fa falta afegir una capa de hardware que gestioni la informació d'entrada i sortida a la memòria interna. En aquest apartat es mostra com s'ha dissenyat aquesta capa.

9.4.1 Arquitectura d'entrada

A la Figura 12 es mostra com és l'arquitectura a l'entrada, es guarda tota la informació de cada una de les peticions. Aquesta és la part de l'arquitectura utilitzada durant la fase d'emmagatzematge.

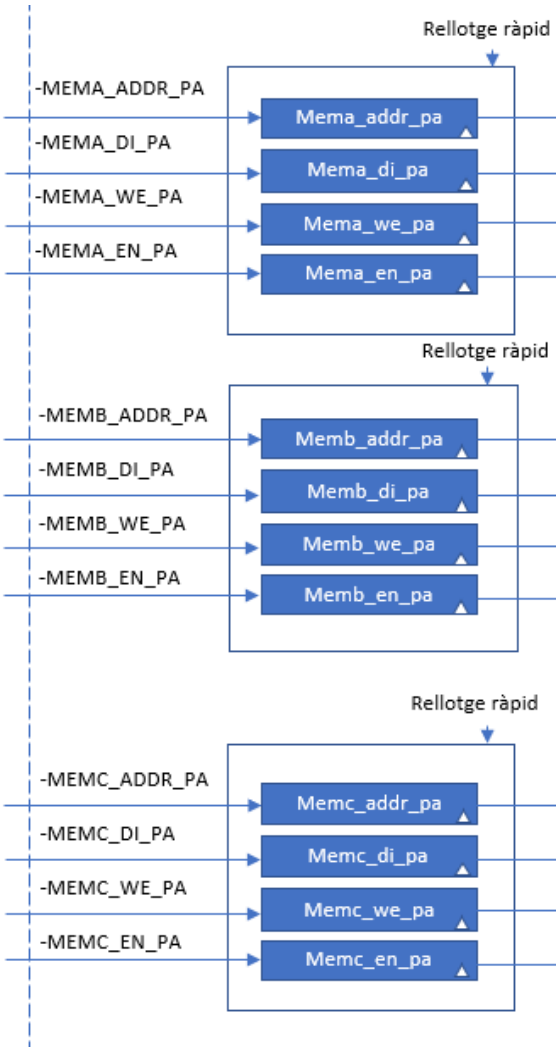

Figura 12: Registres a l'entrada del mòdul unificador. Elaboració pròpia.

Com es pot veure en la Figura 13 s'ha afegit el comptador anomenat Counter. Aquest pot prendre els valors entre 0 i el nombre total de memòries a unificar - 1, en aquest cas el rang va des del 0 fins al 2 i ajuda a determinar quina petició s'envia. Dins de la regió marcada amb una el·lipse vermella es veu com Counter s'encarrega de seleccionar quina petició s'envia a la memòria interna mitjançant el multiplexor Mux 1. Aquest senyal indica en quin cicle de rellotge ràpid ens trobem dins del cicle de rellotge lent.

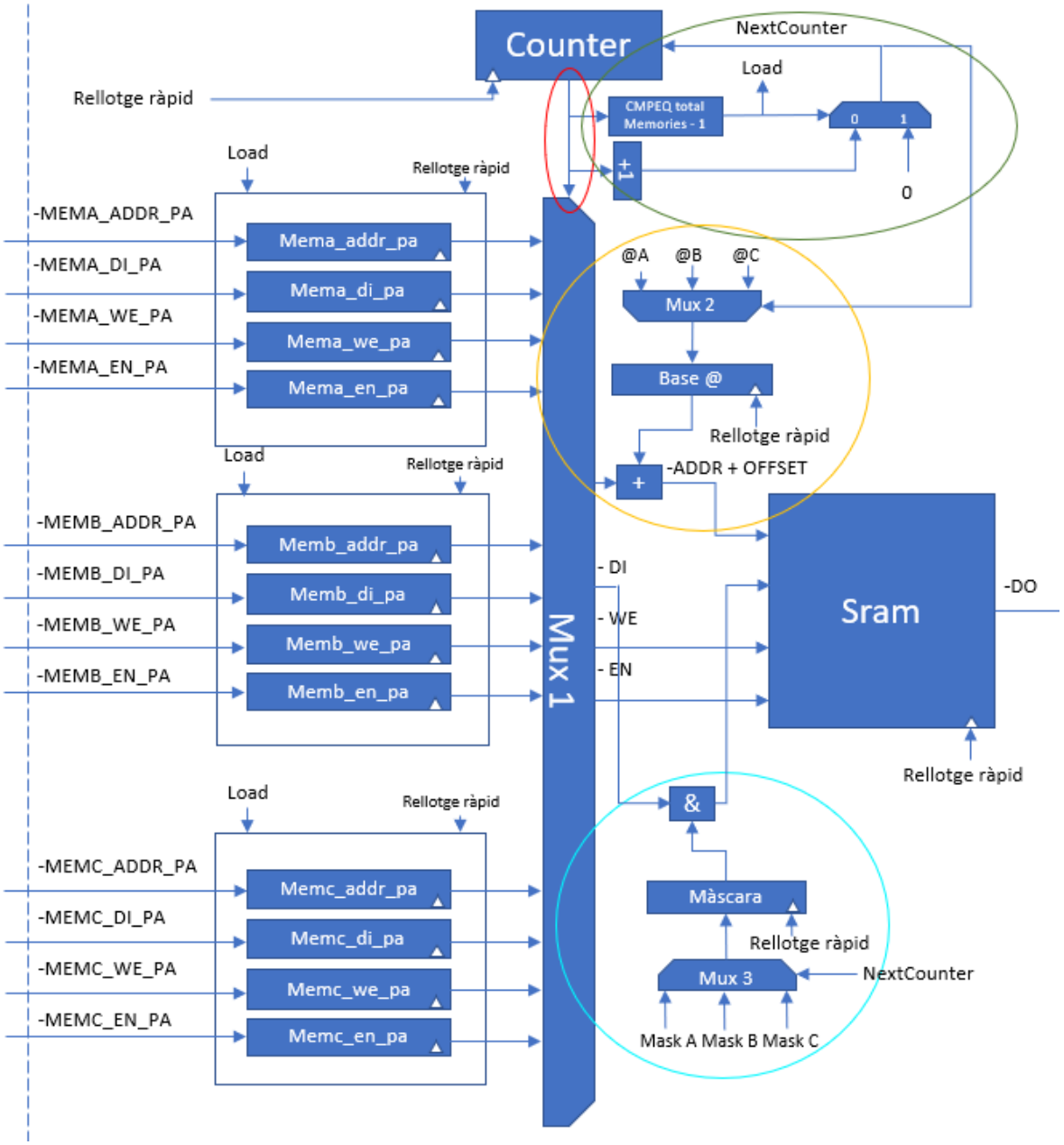

Figura 13: Registres d'entrada més gestió de peticions amb senyal de control. Elaboració pròpia.

En la regió marcada per l'el·lipse verda de la Figura 13 es veu el càlcul de NextCounter: primer s'agafa el valor de Counter, per una banda se li suma 1 i per una altra es compara si és igual al nombre de memòries – 1, en cas de ser igual NextCounter prendrà el valor de 0, en cas contrari tindrà el valor que tenia més 1.

En la Figura 13 dins de la regió marcada per l'el·lipse groga es veu el càlcul del senyal ADDR + OFFSET: primer de tot s'escull quina adreça base s'ha de sumar, aquesta s'escull amb la senyal NextCounter, la qual conté el valor que prendrà el senyal Counter al següent cicle. És necessari utilitzar el senyal NextCounter perquè l'adreça base es guarda primer en el registre Base@ abans de ser utilitzada.

En la Figura 13 dins de la regió marcada per cercle de color blau cel es pot veure que les dades d'entrada són filtrades per una màscara. Aquesta màscara s'assegura que les dades que entren a la memòria tenen la mida de la memòria corresponent.

Ara mateix, tal com està l'arquitectura a la Figura 12, es guardaria a cada flanc ascendent del rellotge ràpid la informació de les peticions de l'entrada quan solament es necessita guardar 1 cop cada 3 cicles del rellotge ràpid. En la Figura 13 dins de la regió verda el senyal Load és l'encarregada de carregar els registres de l'entrada amb la informació que hi ha a l'entrada. Aquesta càrrega només es produeix en el cicle de rellotge ràpid en què Counter és igual al nombre total de memòries a unificar – 1. Es podria utilitzar un altre cicle de rellotge ràpid per carregar la informació, en funció del valor inicial que tingui Counter, però s'ha decidit així perquè facilita la implementació del circuit.

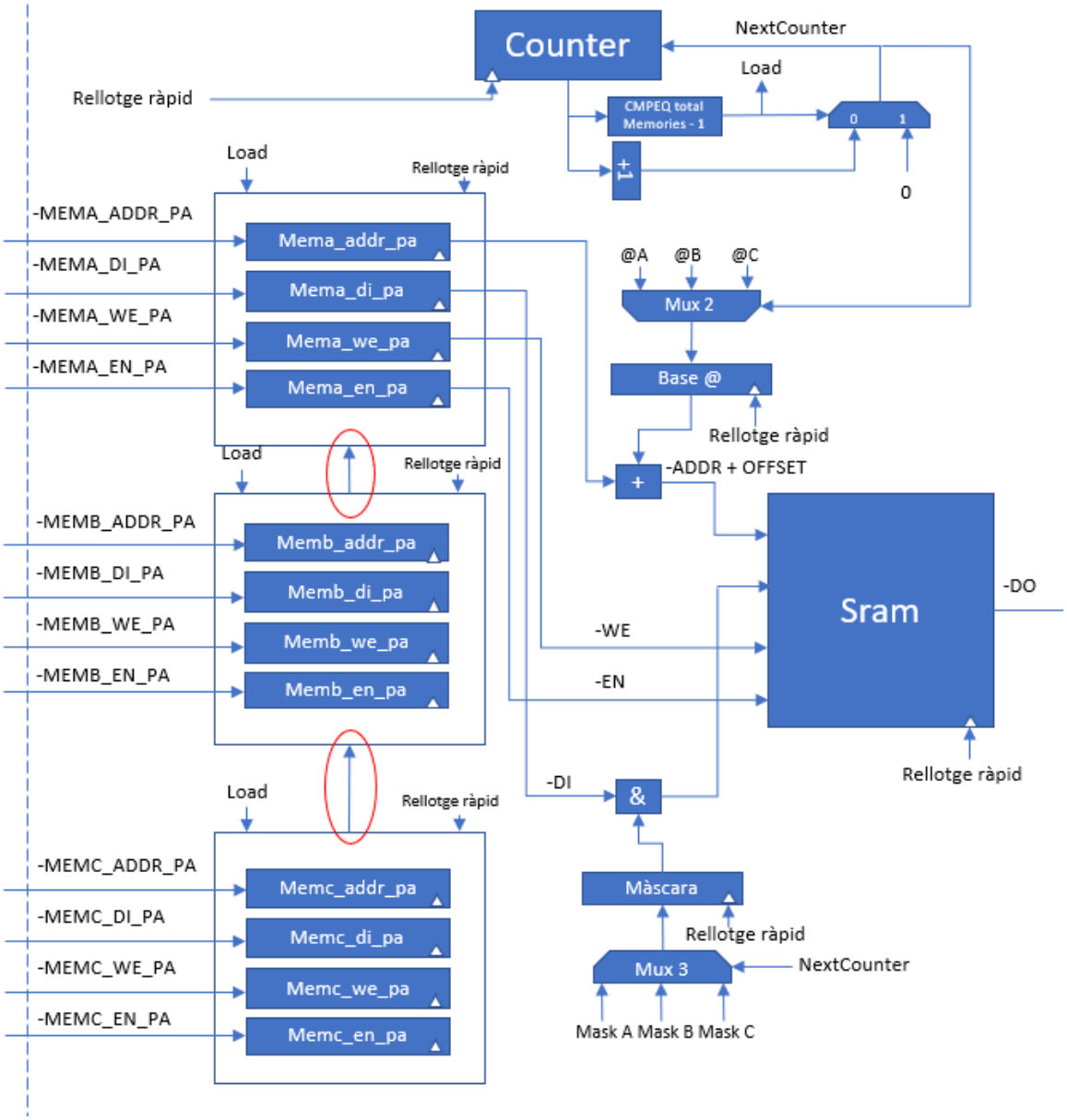

Figura 14: Implementació de l'arquitectura utilitzant *shift register* a l'entrada. Elaboració pròpia.

També es podria haver utilitzat el que s'anomena en anglès *shift register*, consisteix en un conjunt de registres que permeten transformar una entrada de dades paral·lela i convertir-la en seqüencial. Encara que no s'ha realitzat aquesta implementació amb *shift register* a l'entrada, es pensa que seria interessant implementar una arquitectura diferent i veure què implica a nivell de recursos i comportament del mòdul. A la Figura 14 es mostra un exemple de com hagués estat l'entrada utilitzant un model PISO (*Parallel Input Serial Output* – Entrada Paral·lela Sortida Seqüencial), s'utilitzen la mateixa quantitat de registres i el Mux 1 no seria necessari en el disseny. S'ha marcat amb el·lipses de color vermell les connexions que hi hauria entre els registres del *shift register*.

En aquesta arquitectura hi hauria un cicle de càrrega de dades determinada pel senyal Load i a la resta de cicles ràpids s'hauria de desplaçar totes les dades entre els 3 grups de registres, les de la memòria C anirien als registres de la memòria B i les de la memòria B aniria als registres de la memòria A.

9.4.2 Arquitectura de sortida

Un cop dissenyada l'entrada d'acord amb els diagrames temporals, cal dissenyar la sortida del mòdul per tal de quadrar adequadament la sortida de les dades. S'ha de convertir la sortida seqüencial de la memòria interna en una sortida paral·lela i a la vegada cal retardar les dades de sortida els cicles necessaris, per fer-ho s'ha utilitzat un *shift register* de tipus SIPO (Serial Input Parallel Output – Entrada Seqüencial Sortida Paral·lela).

A la Figura 15 es mostra aquesta implementació. Totes les dades que surten per DO es guarden a Memc_out i a cada cicle de rellotge ràpid aquesta informació es va desplaçant a través dels registres Memb_out i Mema_out que, com el seu nom indica, la informació que guarden correspon a la sortida de dades esperades de les memòries C, B i A respectivament.

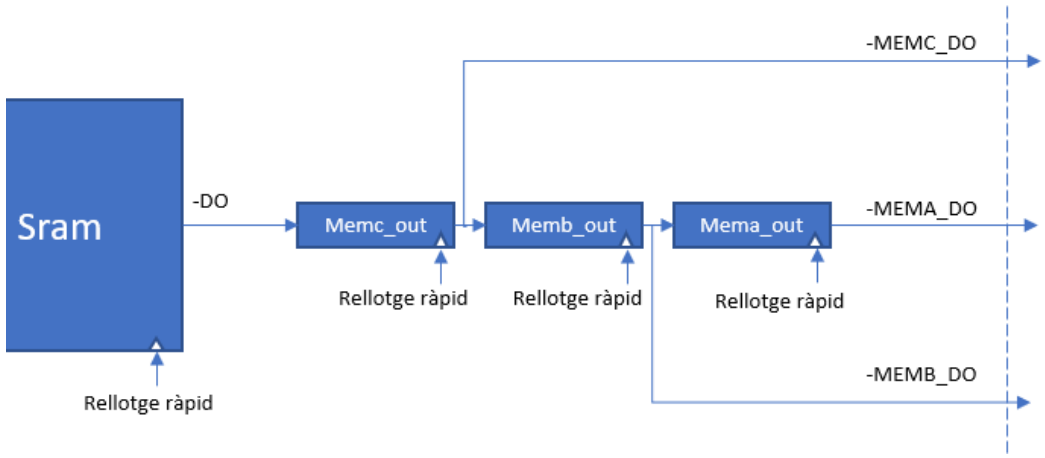

Figura 15: Arquitectura de la sortida de dades implementada amb SIPO. Elaboració pròpia.

Novament la implementació proposada no és l'única existent, es podria implementar una arquitectura simètrica a la de l'entrada, però s'ha desestimat aquesta opció per la complexitat afegida respecte a la proposada, ja que cada registre de la sortida guardaria

les dades a un cicle diferent, el que implica que es necessitaria una comparació entre el valor de Counter i una constant per cada registre i complica l'escalabilitat.

Més endavant es veurà que la quantitat de registres necessària a la sortida no és directament la quantitat de memòries que es desitja optimitzar amb el mòdul.

A continuació a la Figura 17 es mostra quin aspecte tindria tot el disseny complet.

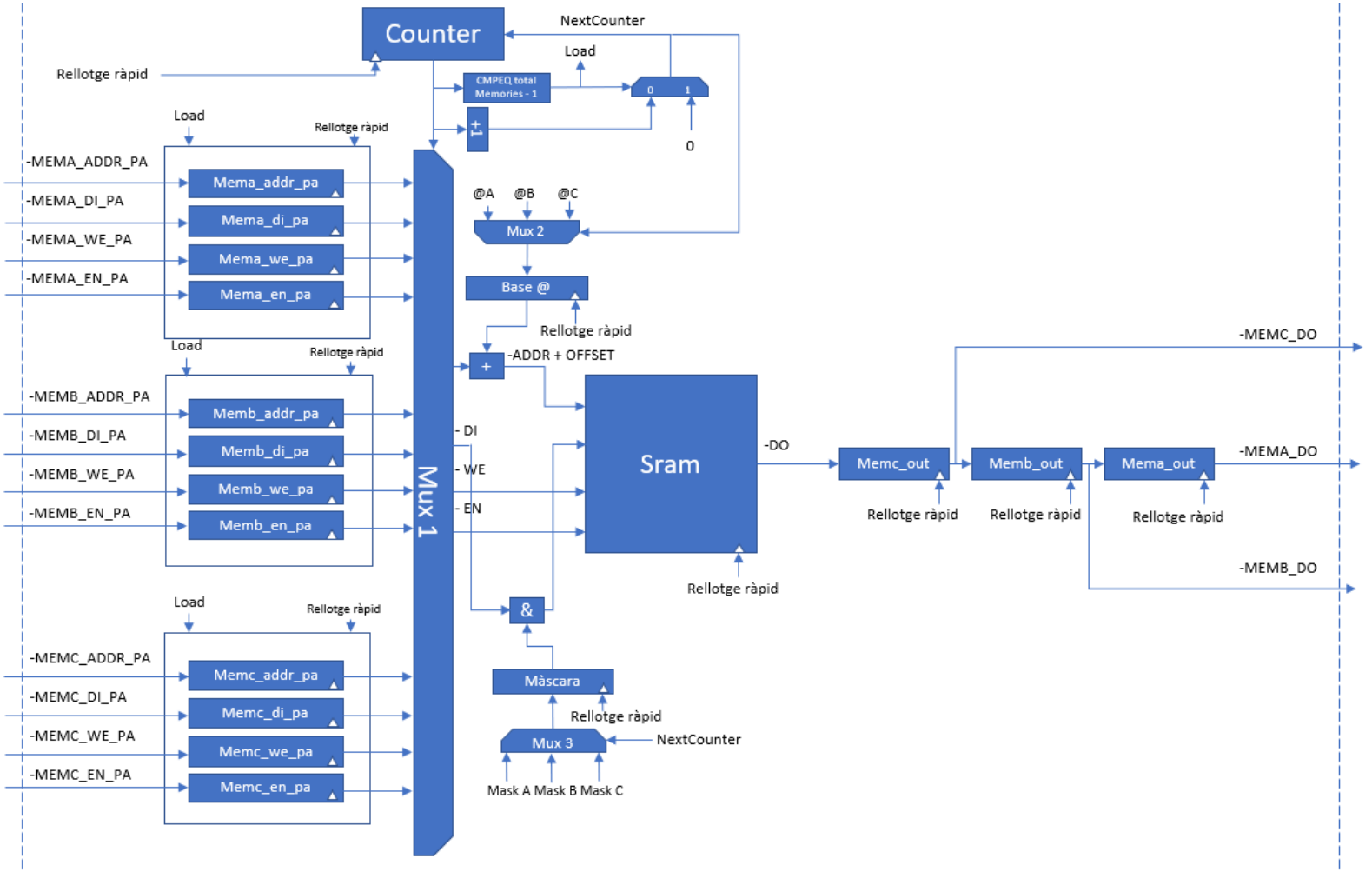

Figura 17: Unificació de l'arquitectura d'entrada i de sortida. Elaboració pròpia.

9.5 Diagrama temporal del disseny

Un cop dissenyada l'arquitectura cal comprovar que aquesta és vàlida mitjançant un diagrama temporal. En la Figura 18 es mostra aquest diagrama juntament amb les peticions de les memòries A, B i C com a exemple de cas d'ús. En el diagrama es mostren tots els punts d'interès esmentats en apartats anteriors.

Els senyals Mema, Memb i Memc representen els registres de la fase d'emmagatzematge, com es pot veure aquests recarreguen la informació quan el senyal Load està a 1, en la Figura 18 s'han marcat amb línies taronges.

Ara que s'ha afegit l'arquitectura necessària per mantenir les dades de sortida de les lectures durant més cicles, es pot apreciar a la Figura 18 com els resultats de les 3 peticions de lectura que arriben al flanc 13 estan disponibles paral·lelament just en el flanc 16 (assenyalat amb una línia vertical vermella). Les peticions de lectura que arriben al flanc 7 i al flanc 10 també estan disponibles un cicle de rellotge lent més tard.

Per entendre una mica millor el pas entre fases, es comenta a continuació que està passant entre el flanc ascendent 2 fins al flanc ascendent 6 del rellotge ràpid en la Figura 18.

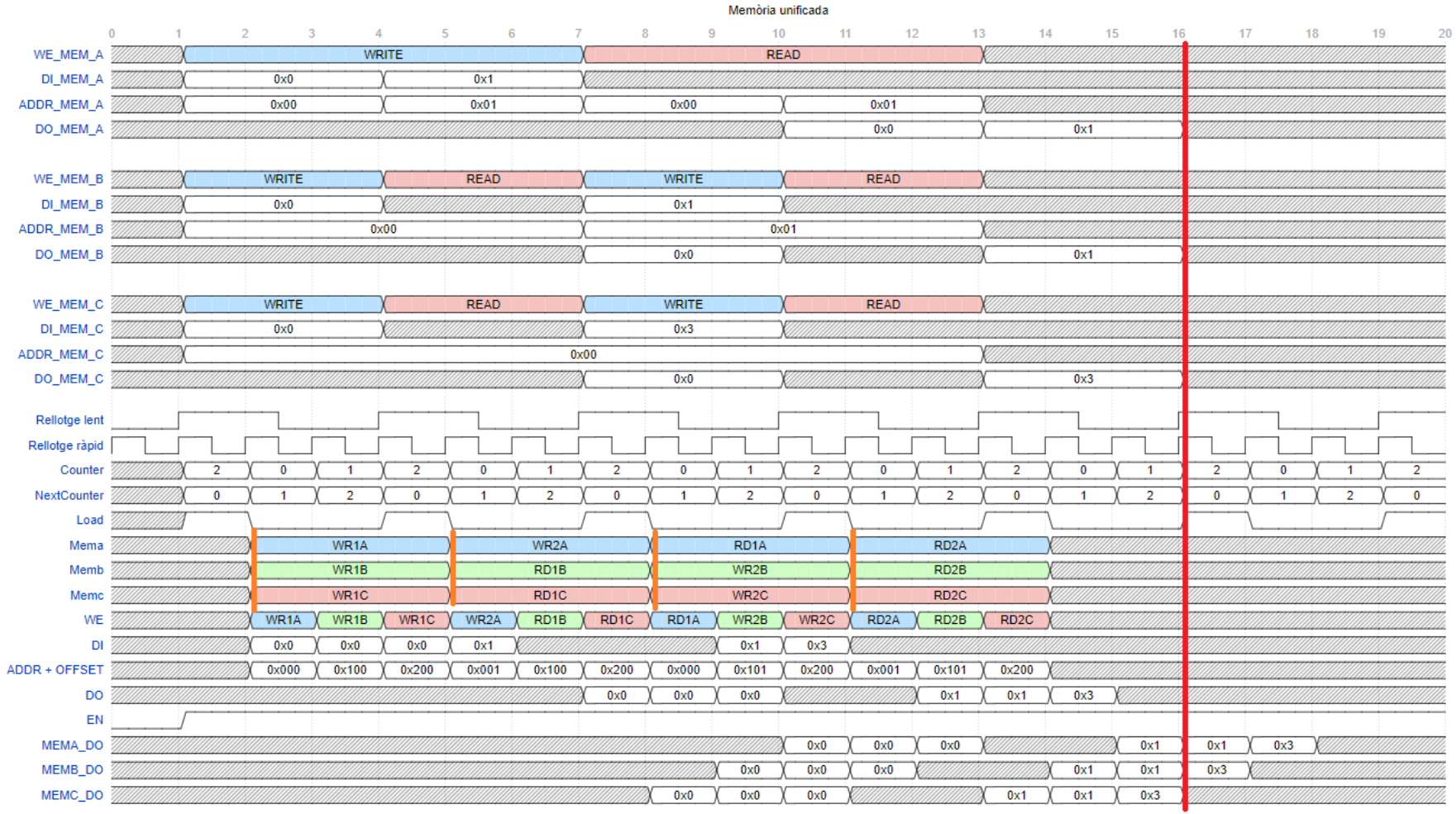

Figura 18: Diagrama temporal de l'arquitectura dissenyada. Elaboració pròpia

- Flanc 2: fase d'emmagatzematge de les 3 peticions als registres d'entrada.

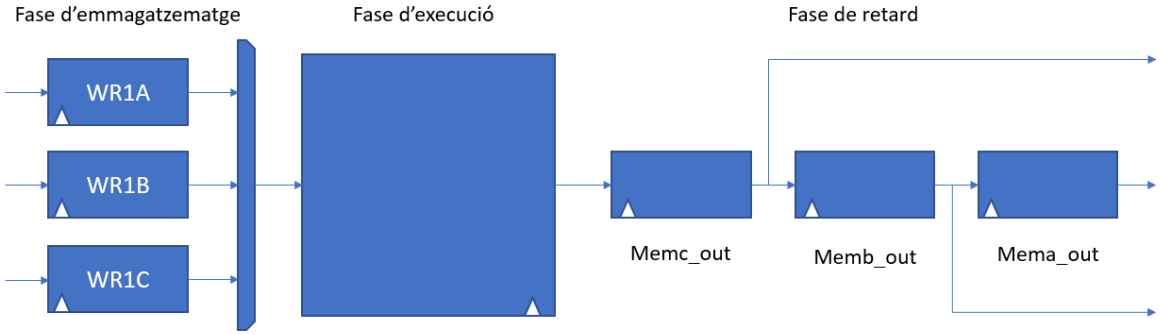

Figura 19: Estat en el flanc 2. Elaboració pròpia.

- Flanc 3: entra en execució la primera petició, WR1A.

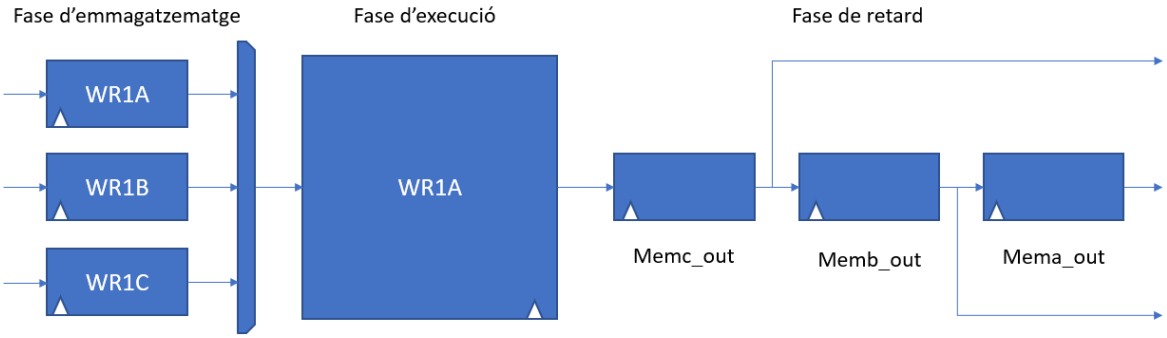

Figura 20: Estat en el flanc 3. Elaboració pròpia.

- Flanc 4: entra en execució la segona petició, WR1B i simultàniament és guarda el resultat de la petició WR1A al registre Memc_out.

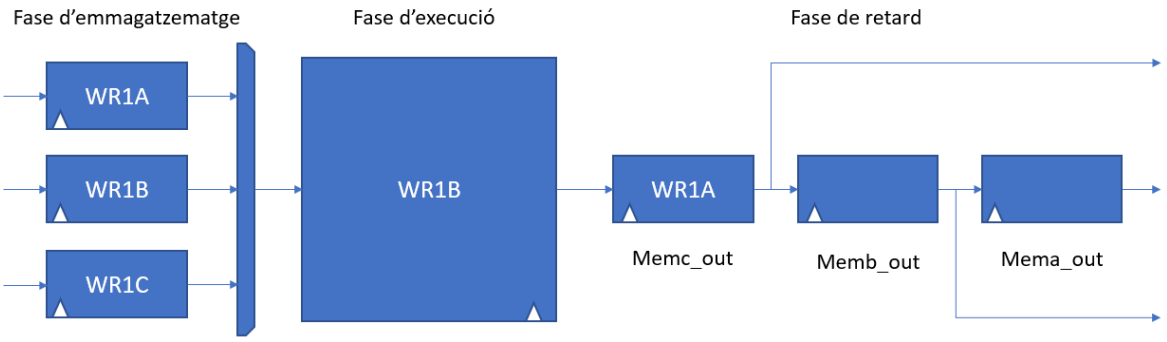

Figura 21: Estat en el flanc 4. Elaboració pròpia.

- Flanc 5: fase d'emmagatzematge de noves peticions, entra en execució la tercera petició WR1C, el valor del registre Memc_out es desplaça al Memb_out i es guarda el resultat de la segona petició al registre Memc_out.

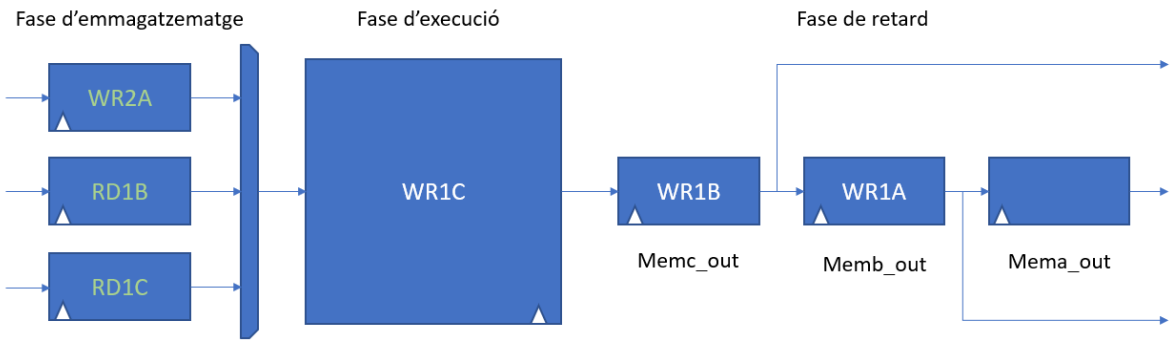

Figura 22: Estat en el flanc 5. Elaboració pròpia.

- Flanc 6: es desplaça el valor del registre Memb_out al Mema_out, el valor de Memc_out al Memb_out i finalment es guarda el resultat de la petició WR1C al registre Memc_out. Entra en execució la petició WR2A.

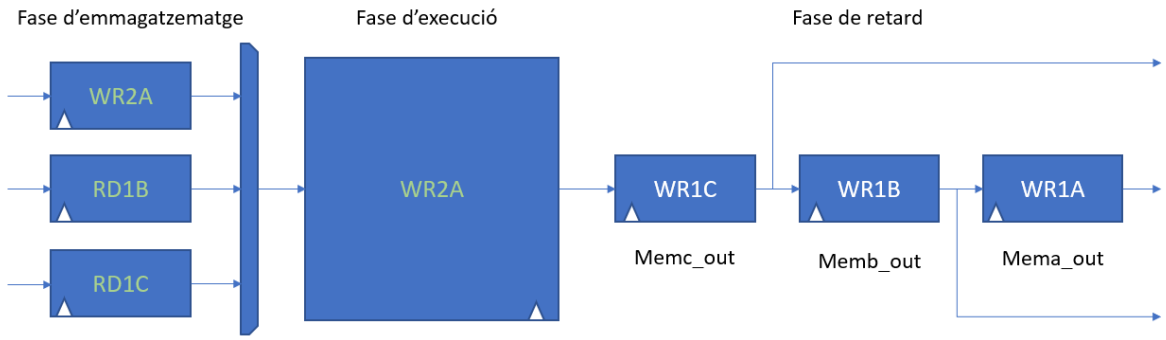

Figura 23: Estat en el flanc 6. Elaboració pròpia.

9.6 Generació del rellotge ràpid

Com s'ha anat veient fins ara gran part del mòdul unificador funciona amb el que s'ha anomenat rellotge ràpid. Aquest és el rellotge que utilitza el mòdul per treballar, el qual ha de ser síncron i a la vegada ha de ser el suficientment ràpid per processar N peticions en el que dura 1 sol cicle del rellotge lent, on N és el nombre de memòries. Tal com es diu en l'apartat 9.3 en el 5è punt d'interès, el creixement de la freqüència del rellotge ràpid és lineal.

Per generar el senyal de rellotge ràpid s'ha utilitzat un component anomenat MMCM (Mixed Mode Clock Manager). Aquest permet generar freqüències basades en una freqüència d'entrada, d'aquesta manera es pot generar la freqüència de rellotge desitjada per alimentar el mòdul unificador.

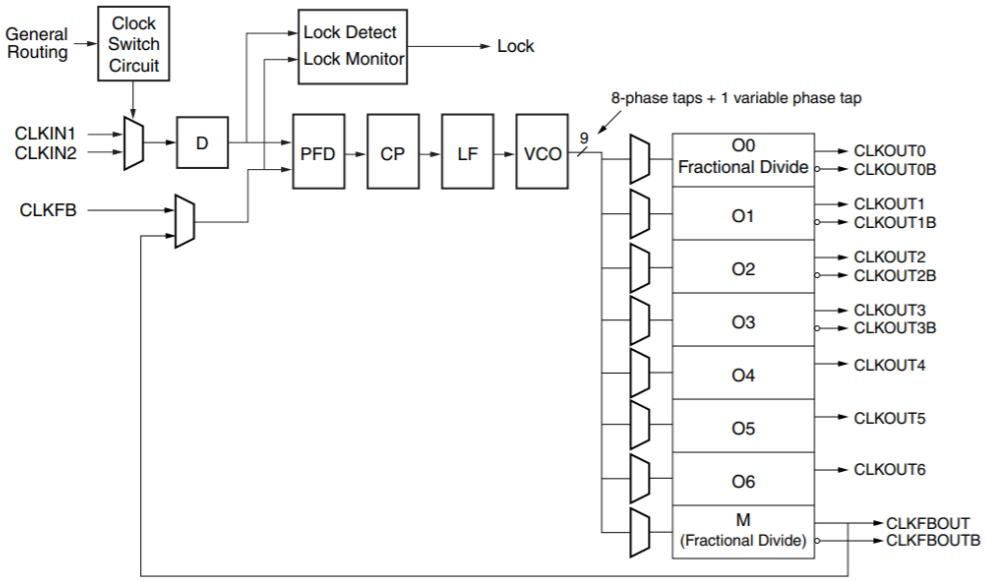

Figura 24: Diagrama del MMCM. Font: https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf

A la Figura 24 es mostra un diagrama de blocs del MMCM, aquest té vàries senyals d'entrada i de sortida, però per aquest projecte només interessen les següents:

- CLKIN1 / CLKIN2: Entrada del senyal de rellotge de referència. Es pot canviar dinàmicament entre la 1 i la 2, però aquesta característica no es farà servir. En el cas que ens ocupa aquí alimentaríem amb la freqüència de rellotge lent.
- CLKOUT1: Generació del senyal de rellotge ràpid.
- CLKFBOUT: Sortida de feedback, com es veu en el diagrama s'ha de connectar a l'entrada de CLKFB.
- CLKFB: Entrada que en aquest cas estarà connectada directament amb CLKFBOUT.
- Lock: Senyal de sortida que indica quan el senyal del rellotge ràpid és estable.

Hi ha una sèrie de paràmetres utilitzats per generar el rellotge ràpid que han de complir una sèrie de requisits. La freqüència de sortida del mòdul VCO es calcula amb la següent funció:

$$F_{VCO} = F_{CLKIN} \times \frac{M}{D}$$

Aquesta, tal com es veu a la funció, es calcula mitjançant dos paràmetres, M i D, que són definits per l'usuari a l'hora d'utilitzar un mòdul MMCM. El resultat de la funció ha d'estar dins del següent rang [600MHz, 1600MHz]. La freqüència del rellotge ràpid es calcula amb la següent funció:

$$F_{Rellotge\ lent} = \frac{F_{VCO}}{O1}$$

On novament O1 és un dels paràmetres definits per l'usuari a l'hora d'utilitzar un mòdul MMCM.

9.7 Reset del disseny

L'últim que queda per afegir al disseny és el senyal de reset, aquesta s'encarrega de reiniciar tots els senyals de control, en aquest cas el Counter. És necessari el senyal de reset perquè aquesta permet deixar l'arquitectura en un estat conegut. No es reinicia el que hi ha guardat a la memòria interna, ja que no té sentit fer peticions de lectura després d'haver fet un reset.

El disseny s'ha de reiniciar en un dels dos casos següents:

1- Arriba explícitament un reset.
2- El rellotge ràpid tarda uns cicles a estabilitzar-se, sempre que el senyal Lock del MMCM estigui a 0 el disseny estarà en constant estat de reset.

10 Escalabilitat

Com el disseny proposat ha de ser capaç d'unificar múltiples quantitats de memòries és interessant estudiar la seva escalabilitat per la seva correcta implementació i per tenir una idea de quan és adient utilitzar aquest disseny i quan no.

L'escalabilitat d'aquest disseny ve limitada principalment per la memòria interna, en el cas d'una BRAM poden emmagatzemar fins a 36Kb i en el cas d'una URAM fins a 288Kb. Sempre que no se superi aquest emmagatzematge en cada un dels dos casos no hi hauria d'haver problema.

Per altra banda tenim la freqüència del rellotge ràpid, aquesta creix seguint la següent fórmula:

$$F_{rellotge\ ràpid} = F_{rellotge\ lent} * Nº\ Memòries$$

Primer de tot les FPGA tenen un rang de freqüència de treball. Suposant que s'està generant una freqüència de rellotge ràpida dins d'aquest rang, s'ha d'anar amb compte perquè a l'hora de fer les diferents connexions del circuit potser es generen camins massa llargs que no poden assegurar l'arribada de les dades a temps. Per aquests motius com més baixa sigui la freqüència de rellotge ràpid més fàcil serà que les dades arribin a temps al seu destí.

La quantitat de registres necessaris a l'entrada és proporcional a la quantitat de memòries a unificar, en canvi, els registres necessaris de la fase de retard varien en funció de la quantitat de cicles que cal retardar cada una de les peticions. En la Taula X4 es mostra quants registres es necessiten a la fase de retard en funció del nombre de memòries.

Nº Memòries	Nº Registres
2	1
3	3
4	5
5	7

Taula 16: Nº de registres necessaris en fase de retard en funció del número de memòries. Elaboració pròpia.

De la Taula 16 es pot deduir la fórmula següent per tal de calcular el nombre de registres necessaris en funció del número de Memòries:

$$Nº\ Registres = Nº\ Memòries * 2 - 3$$

Amb tota la informació d'aquest apartat i els anteriors es pot començar a fer la implementació del mòdul unificador de memòries.

11 Implementació

En aquest apartat s'explica com s'ha implementat el disseny proposat. S'ha dividit la implementació amb 3 apartats.

11.1 Mòdul unificador

A continuació es mostren els fragments més rellevants de la implementació del mòdul unificador.

11.1.1 Paràmetres i senyals d'entrada / sortida

A la Figura 25 es mostren els paràmetres i senyals d'entrada / sortida del mòdul unificador. En els paràmetres, com el seu nom indica, permeten parametritzar el mòdul en funció del seu valor i en aquest cas tenim els següents paràmetres:

1. TOTAL_MEMORIES: nombre total de memòries a unificar.
2. PORTA_WR_DEPTH_ARRAY: vector que conté la profunditat de cada memòria a unificar.
3. PORTA_WR_WIDTH_ARRAY: vector que conté l'amplada de les dades de cada memòria a unificar.
4. MAX_DEPTH: indica el valor màxim de PORTA_WR_DEPTH_ARRAY.
5. MAX_WIDTH: indica el valor màxim de PORTA_WR_WIDTH_ARRAY.
6. OUTPUT_REG_A / B: indiquen si es vol afegir un registre a la sortida de la memòria, en aquest cas no s'ha implementat, però s'ha mantingut el paràmetre per si s'implementa en un futur.

De senyals d'entrada hi ha els dos rellotges, el senyal de reset, la de locked i els senyals corresponents a l'entrada i sortida de dades estan totes en el vector d'interfícies SramIF. Una interfície es pot entendre com un struct on les variables són senyals d'entrada i/o sortida.

```
1
2 module sram_tdp_multiple_generic_input #
3 (
4     parameter TOTAL_MEMORIES=3,
5     parameter int PORTA_WR_DEPTH_ARRAY [TOTAL_MEMORIES] = {50, 50, 50},
6     parameter int PORTA_WR_WIDTH_ARRAY [TOTAL_MEMORIES] = {32, 32, 32},
7     parameter int MAX_DEPTH=50,
8     parameter int MAX_WIDTH=32,
9     parameter OUTPUT_REG_A=0,
10    parameter OUTPUT_REG_B=0,
11 )
12 (
13    input  logic                              slow_clock,
14    input  logic                              fast_clock,
15    input  logic                              reset,
16    input  logic                              locked,
17
18    multiple_sram_if.sp SramIF [TOTAL_MEMORIES-1:0]
19 );
```

Figura 25: Paràmetres i senyals d'entrada / sortida. Elaboració pròpia.

11.1.2 Control de paràmetres

El control de paràmetres que es veu en la Figura 26 s'assegura que el valor tant de MAX_DEPTH com de MAX_WIDTH són el valor màxim present als vectors PORTA_WR_DEPTH_ARRAY i PORTA_WR_WIDTH_ARRAY respectivament.

L'usuari del mòdul és l'encarregat d'indicar quin és el màxim en els dos casos. Hagués estat interessant que una funció directament s'encarregués de trobar-los automàticament, però en SystemVerilog no és possible calcular un paràmetre basant-se en un vector de mida parametritzada, com és el cas.

```
21      initial begin
22          int maxDepth, maxWidth;
23
24          maxDepth = maxValueInArray(PORTA_WR_DEPTH_ARRAY);
25          maxWidth = maxValueInArray(PORTA_WR_WIDTH_ARRAY);
26
27          assert (MAX_DEPTH == maxDepth) else begin
28              $display("Error : MAX_DEPTH is %d when actual maximum depth is %d", MAX_DEPTH, maxDepth);
29              $finish(2);
30          end
31
32          assert (MAX_WIDTH == maxWidth) else begin
33              $display("Error : MAX_WIDTH is %d when actual maximum width is %d", MAX_WIDTH, maxWidth);
34              $finish(2);
35          end
36      end
```

Figura 26: Control de paràmetres. Elaboració pròpia

11.1.3 Memòria interna

A la Figura 27 es mostra com s'ha instanciat la memòria interna. Els dos paràmetres importants són el de PORTA_WR_DEPTH, el qual determina quantes paraules hi caben a la memòria, i el PORTA_WR_WIDTH, el qual determina de quants bits és cada paraula. L'amplada de la paraula de la memòria interna sempre és l'amplada de la paraula més gran de totes les memòries que es vol unificar.

```
217     sram_tdp #(.PORTA_WR_DEPTH(SUM_DEPTH), .PORTA_WR_WIDTH(MAX_WIDTH), .OUTPUT_REG_A(OUTPUT_REG_A), .OUTPUT_REG_B(OUTPUT_REG_B)) INNER_SRAM (
218             .PortA_Clk      (fast_clock       ),
219             .PortA_Addr     (iAddr_sram_A     ),
220             .PortA_WrData   (iWrData_sram_A   ),
221             .PortA_RdData   (outputA[TOTAL_OUTPUT_RESOURCES-1]),
222             .PortA_WrEn     (iWrEn_sram_A     ),
223             .PortA_Ena      (iEna_A           ),
224
225             .PortB_Clk      (fast_clock       ),
226             .PortB_Addr     (iAddr_sram_B     ),
227             .PortB_WrData   (iWrData_sram_B   ),
228             .PortB_RdData   (outputB[TOTAL_OUTPUT_RESOURCES-1]),
229             .PortB_WrEn     (iWrEn_sram_B     ),
230             .PortB_Ena      (iEna_B           )
231           );
```

Figura 27: Memòria interna. Elaboració pròpia.

11.1.4 Senyals i registres interns

En la Figura 28 es mostren tots els senyals i registres del mòdul unificador. A la línia 88 s'ha declarat un vector on es guarden totes les adreces base. A la 90 hi ha el vector de màscares. El paràmetre SUM_DEPTH de la línia 93 determina quantes paraules s'hauran de guardar en la memòria interna. A la línia 95 hi ha el paràmetre TOTAL_OUTPUT_RESOURCES, el qual indica quants registres es necessiten a la fase

de retard + 1, aquest últim + 1 fa referència al senyal de sortida de dades de la memòria interna. A les línies 98 i 99 hi ha un vector que correspon a les senyals de sortida de dades de la memòria interna i dels registres de la fase de retard. De la línia 102 fins la 110 hi ha tots els registres utilitzats per guardar les dades en la fase d'emmagatzematge. De la 113 a la 121 hi ha totes els senyals d'entrada a la memòria interna. A continuació es declaren els senyals i registres relacionats amb el reset. A la línia 129 hi ha declarat el registre Counter i a la 130 es declara el senyal de nextCounter. Per últim hi ha els registres que contindran el valor de la màscara per filtrar les dades i el valor de l'adreça base que s'ha de sumar a la petició.

```
87    // Array of base address
88    localparam intArrayTotalMemorySized BASE_ADDRESS = calculateBaseAddress(PORTA_WR_DEPTH_ARRAY);
89    // Array of width mask
90    localparam logicArrayTotalMemorySized WIDTH_MASK = calculateWidthBitMask(PORTA_WR_WIDTH_ARRAY);
91
92    // Total depth for internal memory
93    localparam SUM_DEPTH = getTotalDepth(PORTA_WR_DEPTH_ARRAY);
94    // Total output register + output from internal memory
95    localparam TOTAL_OUTPUT_RESOURCES = 2*TOTAL_MEMORIES - 2;
96
97    // Output data from internal memory output registers
98    logic [(MAX_WIDTH-1):0] outputA [TOTAL_OUTPUT_RESOURCES-1:0];
99    logic [(MAX_WIDTH-1):0] outputB [TOTAL_OUTPUT_RESOURCES-1:0];
100
101   // Input registers
102   logic [$clog2(MAX_DEPTH)-1:0] Addr_sram_A [TOTAL_MEMORIES-1:0];
103   logic [(MAX_WIDTH-1):0]       WrData_sram_A [TOTAL_MEMORIES-1:0];
104   logic                         WrEn_sram_A [TOTAL_MEMORIES-1:0];
105   logic                         Ena_A [TOTAL_MEMORIES-1:0];
106
107   logic [$clog2(MAX_DEPTH)-1:0] Addr_sram_B [TOTAL_MEMORIES-1:0];
108   logic [(MAX_WIDTH-1):0]       WrData_sram_B [TOTAL_MEMORIES-1:0];
109   logic                         WrEn_sram_B [TOTAL_MEMORIES-1:0];
110   logic                         Ena_B [TOTAL_MEMORIES-1:0];
111
112   // Input signals to the internal memory
113   logic [$clog2(SUM_DEPTH)-1:0] iAddr_sram_A;
114   logic [(MAX_WIDTH-1):0]       iWrData_sram_A;
115   logic                         iWrEn_sram_A;
116   logic                         iEna_A;
117
118   logic [$clog2(SUM_DEPTH)-1:0] iAddr_sram_B;
119   logic [(MAX_WIDTH-1):0]       iWrData_sram_B;
120   logic                         iWrEn_sram_B;
121   logic                         iEna_B;
122
123   // Asynchronous reset
124   logic asyncRst;
125   (* ASYNC_REG = "TRUE" *) logic [2:0] shiftRegisterRst = 3'b111;
126   logic registerRst;
127
128   // Counters
129   logic [$clog2(TOTAL_MEMORIES)-1:0]  counter = TOTAL_MEMORIES-1;
130   logic [$clog2(TOTAL_MEMORIES)-1:0]  nextCounter;
131
132   // Data Mask & base address
133   logic [(MAX_WIDTH-1):0] mask = 0;
134   logic [$clog2(SUM_DEPTH)-1:0] baseAddress = 0;
```

Figura 28: Senyals i registres interns del mòdul unificador. Elaboració pròpia.

11.1.5 Reset i counter

En la Figura 29 (línies 137 – 147) es mostra la implementació del reset asíncron. Amb aquesta implementació s'assegura que si el reset es desactiva abans d'arribar al flanc

ascendent del rellotge lent no es perdrà. El reset del registre Counter a un estat conegut en funció del senyal registerRst es mostra de la línia 149 a la 155.

```
137     assign asyncRst = ((reset == 1) || (locked == 0));
138
139     always @(posedge slow_clock or posedge asyncRst) begin
140             if (asyncRst) begin
141                 shiftRegisterRst <= 3'b111;
142             end else begin
143                 shiftRegisterRst <= {shiftRegisterRst[1:0], 1'b0};
144             end
145     end
146
147     assign registerRst = shiftRegisterRst[$left(shiftRegisterRst)];
148
149     always @(posedge fast_clock) begin
150         if (registerRst) begin
151             counter <= TOTAL_MEMORIES-2;
152         end else begin
153             counter <= nextCounter;
154         end
155     end
```

Figura 29: Implementació del reset del disseny. Elaboració pròpia.

11.1.6 NextCounter i assignació de la petició

En la Figura 30 es mostra com s'escull el valor de nextCounter en funció del valor actual de Counter. En el mateix bloc també es descriu la selecció de la petició en funció del valor de Counter i es pot apreciar tant en les línies 191 i 192 com les 196 i 197 com se suma l'adreça base i com s'utilitza la "&" bit a bit per assegurar que les dades tenen la mida indicada.

```
182     always_comb
183     begin
184
185         if (counter == TOTAL_MEMORIES-1) begin
186             nextCounter = 0;
187         end else begin
188             nextCounter = counter + 1;
189         end
190
191         iAddr_sram_A = Addr_sram_A[counter] + baseAddress;
192         iWrData_sram_A = WrData_sram_A[counter] & mask;
193         iWrEn_sram_A = WrEn_sram_A[counter];
194         iEna_A = Ena_A[counter];
195
196         iAddr_sram_B = Addr_sram_B[counter] + baseAddress;
197         iWrData_sram_B = WrData_sram_B[counter] & mask;
198         iWrEn_sram_B = WrEn_sram_B[counter];
199         iEna_B = Ena_B[counter];
200     end
```

Figura 30: Implementació del NextCounter i assignació de petició. Elaboració pròpia.

11.1.7 Senyals de sortida i emmagatzematge de les peticions

En la Figura 31 es mostra un bucle encarregat d'assignar els valors guardats en els registres de la fase de retard a la seva sortida corresponent (línia 160 i 161) i, a la vegada, també genera els blocs que guardaran les peticions que arriben.

```
157     genvar i;
158     generate
159         for (i = 0; i < TOTAL_MEMORIES; i = i + 1) begin : assigningOutputs
160             assign SramIF[i].RdData_A = outputA[i];
161             assign SramIF[i].RdData_B = outputB[i];
162 
163             always @(posedge fast_clock)  begin
164                 if (counter == TOTAL_MEMORIES-1) begin
165                     //store all data in registers
166 
167                     Addr_sram_A[i]     <= SramIF[i].Addr_A;
168                     WrData_sram_A[i]   <= SramIF[i].WrData_A;
169                     WrEn_sram_A[i]     <= SramIF[i].WrEn_A;
170                     Ena_A[i]           <= SramIF[i].En_A;
171 
172                     Addr_sram_B[i]     <= SramIF[i].Addr_B;
173                     WrData_sram_B[i]   <= SramIF[i].WrData_B;
174                     WrEn_sram_B[i]     <= SramIF[i].WrEn_B;
175                     Ena_B[i]           <= SramIF[i].En_B;
176 
177                 end
178             end
179         end
180     endgenerate
```

Figura 31: Assignació senyals de sortida i emmagatzematge de peticions. Elaboració pròpia.

11.1.8 Shift register, màscara i adreça base

A continuació en la Figura 32 es pot veure la porció de codi on el valor de la màscara i de l'adreça base s'escull en funció del nextCounter en les línies 206 i 207 tal com s'ha comentat. Per altra banda a les línies 209 i 210 es pot veure com s'ha implementat el *shift register* de la fase de retard.

```
204     always @(posedge fast_clock)
205     begin
206         mask            <= WIDTH_MASK[nextCounter];
207         baseAddress     <= BASE_ADDRESS[nextCounter];
208 
209         outputA[TOTAL_OUTPUT_RESOURCES-2:0] <= outputA[TOTAL_OUTPUT_RESOURCES-1:1];
210         outputB[TOTAL_OUTPUT_RESOURCES-2:0] <= outputB[TOTAL_OUTPUT_RESOURCES-1:1];
211 
212     end
```

Figura 32*: Shift register*, assignació de màscara i adreça base. Elaboració pròpia.

11.2 MMCM Wrapper

Per instanciar un MMCM cal indicar una sèrie de paràmetres perquè aquest generi la freqüència de rellotge ràpid desitjada. Per facilitar aquesta feina a l'usuari s'ha creat un wrapper del mòdul MMCM capaç de calcular els paràmetres CLKFBOUT_MULT_F, DIVCLK_DIVIDE i CLKOU1_DIVIDE automàticament (M, D i O1).

A la Figura 33 es mostra la funció getMultF() la qual s'encarrega de retornar un vector d'enters de dues posicions. Els dos valors continguts s'utilitzen per calcular el valor dels paràmetres CLKFBOUT_MULT_F i DIVCLK_DIVIDE.

```
 3 module mmcme2_base_wrapper #
 4 (
 5     parameter TOTAL_MEMORIES=3,
 6     parameter SLOW_PERIOD=10
 7 )
 8 (
 9     input  logic slow_clock,
10     input  logic rst,
11     input  logic pwrdwn,
12     output logic fast_clock,
13     output logic locked
14
15 );
16
17     //All frequencies in MHz
18     localparam real FIN = 1000/SLOW_PERIOD;
19     localparam real FOUT = FIN*TOTAL_MEMORIES;
20     localparam real minVCOout = 600;
21     localparam real maxVCOout = 1600;
22
23     localparam minMultF = 2;
24     localparam maxMultF = 64;
25     localparam minDivMaster = 1;
26     localparam maxDivMaster = 10;
27
28     typedef real arrayOfSize2 [2];
29
30     function automatic arrayOfSize2 getMultF();
31         real i, j;
32         real aux, aux2;
33         arrayOfSize2 value;
34         real MultF = 0.000;
35         for (j = minDivMaster; j <= maxDivMaster; j = j + 1) begin
36             for (i = int'(maxMultF/TOTAL_MEMORIES); (i >= minMultF) && (i >= j); i = i - 1) begin
37                 aux = i/j;
38                 MultF = FOUT*aux;
39                 aux = aux - int'(aux);        //getting decimal part, we have to check it doesnt have decimals
40                 if ((MultF >= minVCOout) && (MultF <= maxVCOout)  && (aux == 0.000)) begin
41                     value[0] = i;
42                     value[1] = j;
43                 end
44             end
45         end
46         return value;
47     endfunction
48
49     localparam arrayOfSize2 MULTF_DIV = getMultF();
```

Figura 33: Funció que calcula els paràmetres CLKFBOUT_MULT_F i DIVCLK_DIVIDE. Elaboració pròpia.

```
63     MMCME2_BASE #(
64         .BANDWIDTH("OPTIMIZED"),   // Jitter programming (OPTIMIZED, HIGH, LOW)
65         .CLKFBOUT_MULT_F(MULTF_DIV[0]*TOTAL_MEMORIES),     // Multiply value for all CLKOUT (2.000-64.000).
66         .CLKFBOUT_PHASE(0.0),      // Phase offset in degrees of CLKFB (-360.000-360.000).
67         .CLKIN1_PERIOD(SLOW_PERIOD),       // Input clock period in ns to ps resolution (i.e. 33.333 is 30 MHz).
68         // CLKOUT0_DIVIDE - CLKOUT6_DIVIDE: Divide amount for each CLKOUT (1-128)
69         .CLKOUT1_DIVIDE(MULTF_DIV[0]/MULTF_DIV[1]),
70         .CLKOUT2_DIVIDE(1),
71         .CLKOUT3_DIVIDE(1),
72         .CLKOUT4_DIVIDE(1),
73         .CLKOUT5_DIVIDE(1),
74         .CLKOUT6_DIVIDE(1),
75         .CLKOUT0_DIVIDE_F(1.000),    // Divide amount for CLKOUT0 (1.000-128.000).
76         // CLKOUT0_DUTY_CYCLE - CLKOUT6_DUTY_CYCLE: Duty cycle for each CLKOUT (0.01-0.99).
77         .CLKOUT0_DUTY_CYCLE(0.5),
78         .CLKOUT1_DUTY_CYCLE(0.5),
79         .CLKOUT2_DUTY_CYCLE(0.5),
80         .CLKOUT3_DUTY_CYCLE(0.5),
81         .CLKOUT4_DUTY_CYCLE(0.5),
82         .CLKOUT5_DUTY_CYCLE(0.5),
83         .CLKOUT6_DUTY_CYCLE(0.5),
84         // CLKOUT0_PHASE - CLKOUT6_PHASE: Phase offset for each CLKOUT (-360.000-360.000).
85         .CLKOUT0_PHASE(0.0),
86         .CLKOUT1_PHASE(0.0),
87         .CLKOUT2_PHASE(0.0),
88         .CLKOUT3_PHASE(0.0),
89         .CLKOUT4_PHASE(0.0),
90         .CLKOUT5_PHASE(0.0),
91         .CLKOUT6_PHASE(0.0),
92         .CLKOUT4_CASCADE("FALSE"), // Cascade CLKOUT4 counter with CLKOUT6 (FALSE, TRUE)
93         .DIVCLK_DIVIDE(MULTF_DIV[1]),         // Master division value (1-106)
94         .REF_JITTER1(0),         // Reference input jitter in UI (0.000-0.999).
95         .STARTUP_WAIT("FALSE")     // Delays DONE until MMCM is locked (FALSE, TRUE)
96     )
97     MMCME2_BASE_inst (
```

Figura 34: Paràmetres del mòdul MMCM. Elaboració pròpia.

En la Figura 34 es mostren els paràmetres del mòdul MMCM. Els que interessen en aquest cas són els de les línies 65, 69 i 93, que són els 3 paràmetres principal que permeten obtenir la freqüència del rellotge ràpid. També hi ha paràmetres com del desfasament (CLKFBOUT_PHASE) o el cicle de treball (DUTY_CYCLE) però aquests s'han deixat amb els valors per defecte.

11.3 Interfícies

Un dels requisits del mòdul unificador era que fos capaç d'unificar memòries que es trobarien a diferents nivells jeràrquics. L'ús d'interfícies ho facilita, ja que agrupa un conjunt de senyals com si en fos una de sola, aquesta característica facilita fer les connexions entre els diferents nivells. En la Figura 35 es mostra com s'ha implementat la interfície, hi ha dos modports, els quals permeten predefinir si uns senyals són d'entrada o de sortida. En aquest cas s'ha implementat en mode màster port (mp) que seria el que s'utilitza en els mòduls que envien peticions i el mode slave port (sp), que és el mode utilitzat en el mòdul unificador, en aquest cas el que rep les peticions.

Igual que els altres mòduls, aquest també és parametritzable. És feina de l'usuari parametritzar correctament cada un dels mòduls perquè aquests funcionin.

```
3  interface multiple_sram_if #(parameter ADDR_WIDTH=10, DATA_WIDTH=8, WEN_WIDTH=1) ();
4
5      logic [ADDR_WIDTH-1:0]      Addr_A    ;
6      logic [DATA_WIDTH-1:0]      WrData_A  ;
7      logic [WEN_WIDTH -1:0]      WrEn_A    ;
8      logic [DATA_WIDTH-1:0]      RdData_A  ;
9      logic                       En_A      ;
10
11     logic [ADDR_WIDTH-1:0]      Addr_B    ;
12     logic [DATA_WIDTH-1:0]      WrData_B  ;
13     logic [WEN_WIDTH -1:0]      WrEn_B    ;
14     logic [DATA_WIDTH-1:0]      RdData_B  ;
15     logic                       En_B      ;
16
17     modport sp (
18         input   Addr_A      ,
19         input   WrData_A    ,
20         input   WrEn_A      ,
21         output  RdData_A    ,
22         input   En_A        ,
23
24         input   Addr_B      ,
25         input   WrData_B    ,
26         input   WrEn_B      ,
27         output  RdData_B    ,
28         input   En_B
29         );
30
31     modport mp (
32         output   Addr_A      ,
33         output   WrData_A    ,
34         output   WrEn_A      ,
35         input    RdData_A    ,
36         output   En_A        ,
37
38         output   Addr_B      ,
39         output   WrData_B    ,
40         output   WrEn_B      ,
41         input    RdData_B    ,
42         output   En_B
43         );
44
45 endinterface : multiple_sram_if
```

Figura 35: Interfície implementada. Elaboració pròpia.

12 Tests

12.1 Test bench

Per tal de comprovar que el disseny és vàlid primer s'ha fet un test bench pel cas de 3 memòries a unificar. Fer un test bench per cada cas d'ús porta bastant feina i el seu manteniment es torna difícil, és per això que un cop passada la primera prova s'ha parametritzat el test de tal manera que aquest pugui simular casos diferents en funció de certs paràmetres.

El test s'ha parametritzat en funció de les següents variables:

1. Nombre total de memòries.
2. Nombre de paraules que pot guardar cada memòria.
3. Mida de la paraula per cada memòria.
4. Meitat del període del rellotge lent.

Els 3 primers paràmetres són evidents, ja que descriuen com són les memòries a unificar en el test. En canvi, l'últim, es va haver d'afegir per controlar el comportament del MMCM, ja que aquest no simulava correctament el rellotge ràpid en alguns casos concrets, per exemple: si es definia el període del rellotge ràpid a 10 nanosegons i es volia simular un cas on s'unifiquessin 3 memòries, el MMCM en comptes de generar el rellotge ràpid amb un període de $3,\hat{3}$ nanosegons el que feia era generar 2 cicles de rellotge ràpid amb un període de 3,3 nanosegons i 1 cicle de rellotge ràpid de 3,4 nanosegons.

El test bench s'ha dividit en dos processos principals:

1. Procés de generació de peticions.
2. Procés de comprovació de resultats.

El test s'ha programat seguint la següent estructura: en funció dels paràmetres d'entrada es genera un mòdul unificador i per altra banda es generen memòries TDP equivalents a les memòries a unificar. Un cop inicialitzat, es comencen a enviar fins a 10.000 peticions pseudoaleatòries tant en el port A com en el port B. Aquestes peticions s'envien tant a les memòries que van per lliure com al mòdul unificador i es compara que les sortides de dades del mòdul unificador coincideixen amb les dades de sortida de cada una de les memòries que van per lliure respectivament.

12.1.1 Generació de peticions

A la Figura 36 es mostra primer de tot el procés de generació de peticions. De les línies 313 fins la 320 es generen les màscares de dades que s'utilitzen per generar les dades de la mida adient.

En el requadre de color taronja hi ha una espera activa, abans de començar a enviar peticions cal assegurar-se que el rellotge ràpid generat pel MMCM és estable.

A continuació en la zona del requadre blau cel es mostra com es reinicia el mòdul unificador posant el senyal “reset_dut” a 1 durant un cicle. Un cop reiniciat s’inicialitza la memòria interna tal com es veu en el requadre de color verd.

```
305    //Process that initializes all memories and then generates petitions.
306    initial begin : sendRandomPetitions_SRAM_process
307
308        int width;
309        bit writePortA, WritePortB;
310
311        $display($time, " << Setting clocks and masks>> ");
312        slow_clock = 1;   //slow clock initial value
313        reset_dut = 0;
314        for (int x = 0; x < TOTAL_MEMORIES; x++) begin
315            width = WIDTH_ARRAY[x];
316            widthMask[x] = 0;
317            for(int y = 0; y < width; y++) begin
318                widthMask[x][y] = 1;
319            end
320            $display("Sram %h has mask of: %h", x, widthMask[x]);
321        end
322        //initialize signals to 0
323        initializePortSignals();            //Dejar mas centralizados los resets
324
325        //wait locked has a value diferent from x
326        repeat(CICLE_BEFORE_LOCK) @(posedge slow_clock);
327
328        $display($time, " << Waiting MMCM output to be LOCKED >> ");
329
330        //$display("%h locked value", LOCKED);
331        while (LOCKED == 0) begin
332            @(posedge slow_clock);
333        end
334
335        $display("%h locked value", LOCKED);
336
337        $display($time, " << MMCM output is LOCKED >> ");
338
339        @(posedge slow_clock);
340        #(WAIT);
341        reset_dut = 1;
342        @(posedge slow_clock);
343        #(WAIT);
344        reset_dut = 0;
345        repeat (5) @(posedge slow_clock);
346        #(WAIT);
347        //Initialize srams
348        for (int o=0; o < TOTAL_MEMORIES; o++) begin
349            for (int q=0; q < DEPTH_ARRAY[o]; q++) begin
350                wr_port_A(o, q, q, 1);
351                @(posedge slow_clock);
352                #(WAIT);
353            end
354            resetInputSignalsA(o);
355        end
356
```

```
357
358         //Send random petitions
359         for (int l = 0; l < NUMBER_ACCESS; l++) begin
360             for (int m = 0; m < TOTAL_MEMORIES; m++) begin
361
362                 //Petition to ports A
363                 wren_a[m] = $urandom_range(1);
364                 if (wren_a[m] == 1'b1) begin
365                     wr_random_port_A(m);
366                 end
367                 else begin
368                     rd_random_port_A(m);
369                 end
370
371                 //Petitions to ports B.
372                 wren_b[m] = $urandom_range(1);
373                 if (wren_b[m] == 1'b1) begin
374                     wr_random_port_B(m);
375                 end
376                 else begin
377                     rd_random_port_B(m);
378                 end
379
380                 //In case we generate 2 write petitions to the same address we disable the ports.
381                 if ((wren_a[m] == 1'b1) && (wren_b[m] == 1'b1) && (address_a[m] == address_b[m])) begin
382                     resetInputSignalsB(m);
383                     resetInputSignalsA(m);
384                 end
385
386             end
387             @(posedge slow_clock);
388             #(WAIT);
389
390         end
391         repeat (5) @(posedge slow_clock);
392         $display ("SIMULATION PASSED at time %d", $time);
393         $finish(2);
394     end
```

Figura 36: Procés d’enviament de peticions. Elaboració pròpia.

Per últim, el bucle que va des de la línia 353 fins la 380 és l'encarregat d'enviar peticions pseudoaleatòries. En el requadre de color vermell es mostra com es genera una petició a cada un dels ports, com la memòria interna és de tipus TDP tant el port A com el port B es poden fer peticions d'escriptura o de lectura. En el cas de generar dues peticions d'escriptura a la mateixa adreça podria generar un conflicte, en el cas que passés, en la zona marcada amb un requadre de color groc es té en compte i es reinicien els senyals de les peticions a 0 durant aquell cicle.

12.1.2 Comprovació de resultats

Les peticions que s'envien al mòdul unificador també són enviades simultàniament a un conjunt de memòries independents, d'aquesta manera es pot comparar resultats. En la Figura 37 es mostra el procés de comprovació de resultats, per a cada memòria independent es genera un procés el qual comprova que a la sortida de dades de les peticions de lectura realitzades coincideixin amb la sortida corresponent del mòdul unificador.

```
243     //Process that checks output petitions are as expected. It compares dut output with sram golden output
244     genvar aux;
245     generate
246     for (aux = 0; aux < TOTAL_MEMORIES; aux++) begin : GENCHECK
247         always@(posedge slow_clock) begin : checkPetitions_SRAM_process
248             if (LOCKED == 1) begin
249                 //check output when the petition from previous cicle has the port enabled and was a read operation
250                 if (delayed_wren_a[aux] == 1'b0 && delayed_ena_a[aux] == 1'b1)  begin
251                     if (data_out_a[aux] !== SramIF[aux].RdData_A) begin
252                         $display("ERROR: Output data from memory %d at time %d, doesnt match between sram and dut", aux, $time);
253                         $display("address A: %h", delayed_address_a[aux]);
254                         $display("sram data: %h dut data: %h", data_out_a[aux], SramIF[aux].RdData_A);
255                         $finish(2);
256                     end
257                     else begin
258                         $display("Memory %h address A: %h", aux, delayed_address_a[aux]);
259                         $display("Read data sram: %d, Read data DUT: %d", data_out_a[aux], SramIF[aux].RdData_A);
260                     end
261                 end
262
263                 if (delayed_wren_b[aux] == 1'b0 && delayed_ena_b[aux] == 1'b1)  begin
264                     if (data_out_b[aux] !== SramIF[aux].RdData_B) begin
265                         $display("ERROR: Output data from memory %d at time %d, doesnt match between sram and dut", aux, $time);
266                         $display("address B: %h", delayed_address_b[aux]);
267                         $display("sram data: %h dut data: %h", data_out_b[aux], SramIF[aux].RdData_B);
268                         $finish(2);
269                     end
270                     else begin
271                         $display("Memory %h address B: %h", aux, delayed_address_b[aux]);
272                         $display("Read data sram: %d, Read data DUT: %d", data_out_b[aux], SramIF[aux].RdData_B);
273                     end
274                 end
275             end
276         end
277     end
278     endgenerate
```

Figura 37: Procés de comprovació de resultats. Elaboració propia.

12.1.3 Simulacions

S'han simulat els següents casos:

A. Cas base on es comprova el funcionament amb memòries de la mateixa mida.
B. Cas on es comprova el funcionament amb memòries que tenen diferent nombre de paraules, però que tenen la mateixa mida de paraules.
C. Cas on es comprova el funcionament amb memòries que tenen el mateix nombre de paraules, però que les paraules són de mides diferents
D. Casos pseudoaleatoris.

En les simulacions de tipus A es comprova el correcte funcionament de casos base, permet identificar anomalies generals del disseny en cas de fallada.

En les simulacions de tipus B permet comprovar el correcte funcionament del càlcul de les adreces base i que no es produeixen conflictes en els diferents accessos a la memòria interna.

En les simulacions de tipus C permet identificar si les màscares que s'aplica a les dades d'entrada estan ben aplicades.

En les simulacions de tipus D permet comprovar casos poc habituals.

A la Taula 17 es mostra quins casos concrets s'han comprovat en simulació.

Tipus	Memòries	Nº paraules	Mida paraules
A	2	32, 32	4, 4
B	2	31, 32	4, 4
B	2	32, 31	4, 4
C	2	32, 32	3, 17
D	2	15, 153	57, 3
A	3	32, 32, 32	4, 4, 4
B	3	31, 32, 32	4, 4, 4
B	3	32, 31, 32	4, 4, 4
B	3	32, 32, 31	4, 4, 4
C	3	32, 32, 32	9, 4, 15
D	3	7, 25, 156	15, 11, 2
A	4	32, 32, 32, 32	4, 4, 4, 4
B	4	31, 32, 32, 32	4, 4, 4, 4
B	4	32, 31, 32, 32	4, 4, 4, 4
B	4	32, 32, 31, 32	4, 4, 4, 4
B	4	32, 32, 32, 31	4, 4, 4, 4
C	4	32, 32, 32, 32	4, 7, 19, 45
D	4	13, 137, 3, 19	4, 7, 19, 45

Taula 17: Casos simulats amb el mòdul unificador. Elaboració pròpia.

Per cada test s'han enviat 10.000 peticions pseudoaleatòries tant en el port A com en el port B i s'ha comprovat la sortida.

12.2 Incorporació en un projecte

L'últim pas que queda per comprovar el funcionament del mòdul consisteix a afegir-lo en un projecte existent, ser capaç de generar el *bitstream* (fitxer que conté la informació de programació d'una FPGA), pujar-lo en una placa, executar els tests d'aquell projecte i comprovar que tots passen satisfactòriament.

En un dels projectes de l'empresa es va buscar un punt de l'arquitectura on es fessin servir memòries BRAM. En la Figura 38 es mostra un esquema de com és una part de l'arquitectura del projecte on s'ha incorporat. En els nivells 3.1, 3.2 i 3.3 s'instancia una memòria de tipus SDP de 256 paraules de 32 bits cada paraula, això fa un total de 8Kb per memòria. En aquesta situació s'estan utilitzant 3 BRAM de 18 Kb. El màxim nivell

d'optimització, en aquest cas, s'obtindria unificant les 3 memòries en una BRAM de 36 KB.

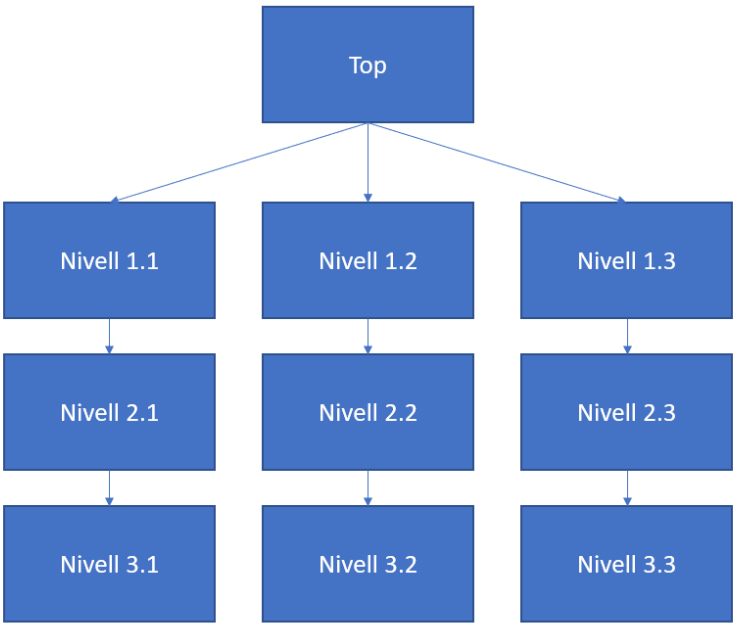

Figura 38: Esquema jerarquia del projecte. Elaboració pròpia.

Encara que el mòdul unificador internament tingui una memòria de tipus TDP, si es fan les connexions adequades es pot obtenir el comportament d'una memòria SDP. S'ha intentat unificar les 3 memòries, però dóna errors de *timing*, el que significa que les connexions són massa llargues o que la freqüència de treball del rellotge és massa ràpida i per tant els senyals, tant les de les peticions a l'entrada com les dades de sortida, no tenen temps suficient per arribar al seu destí.

En la Figura 39 es mostra aquest error de *timing*, a la part de l'esquerra es pot veure les peticions estan disponibles just després del flanc ascendent 1 del rellotge ràpid i aquestes han de passar a la fase d'emmagatzematge just en el flanc ascendent 2 del rellotge ràpid tal com s'indica amb les fletxes. El marge de temps que es dóna a les dades per anar des de l'origen de la petició fins a l'entrada dels registres de la fase d'emmagatzematge és massa petita. Si no s'usés el mòdul unificador el marge de temps que tindrien des d'on es genera la petició fins a la memòria seria el triple, concretament des del flanc ascendent 1 del rellotge ràpid fins al flanc ascendent 4 del rellotge ràpid.

A la part dreta de la Figura 39 també es mostra l'error de *timing* que ha donat els senyals de sortida, com es marca mitjançant 3 fletxes el temps que hi ha disponible des de l'origen des d'on surten les dades de la fase de retard fins al seu destí és massa curt.

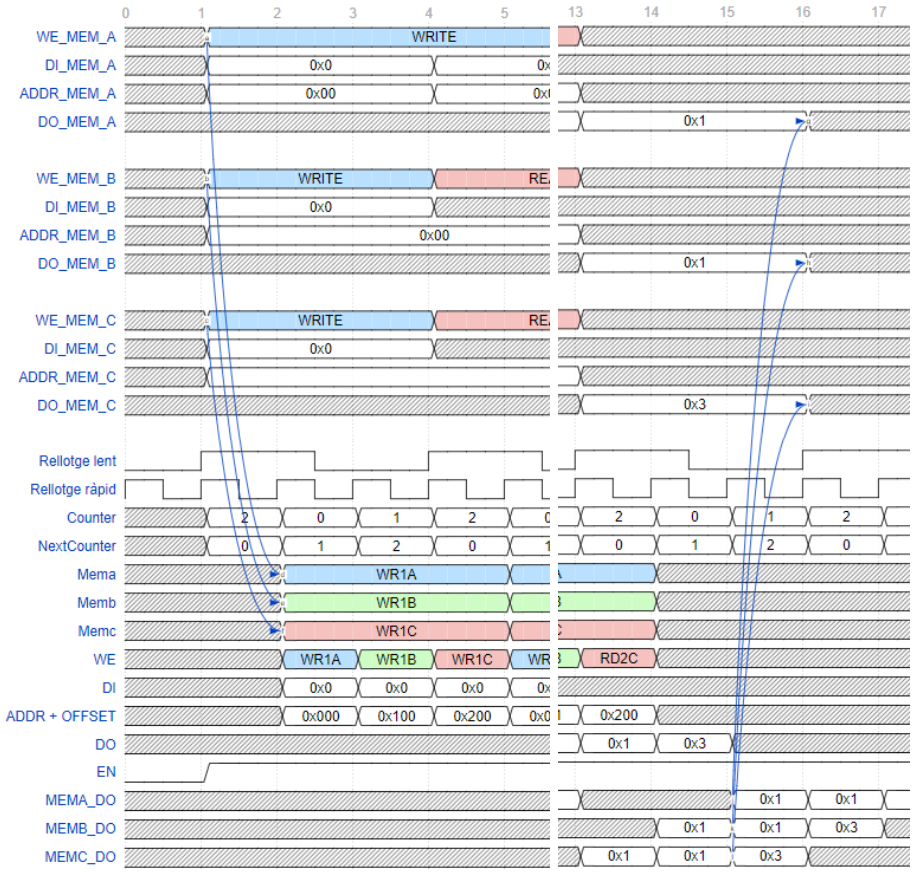

Figura 39: Origen de l'error de *timing* a l'entrada i a la sortida. Elaboració pròpia.

Vists aquests errors de *timing* es va proposar fer una segona prova unificant dues memòries, les del nivell 3.1 i 3.2. Els resultats aquest cop han estat satisfactoris, ja que al només unificar dues memòries la freqüència del rellotge ràpid es redueix lo suficient com per no generar els mateixos errors de timing.

En aquest cas el nivell d'optimització esperat seria de tenir una memòria BRAM18 per la memòria unificada dels nivells 3.1 i 3.2 i, una altra BRAM18 per la memòria del nivell 3.3. A la Taula 18 es mostren els recursos utilitzats. Contra tot pronòstic, en el mòdul unificador se li va assignar una BRAM36, això no suposa cap optimització, però fent més investigació es va descobrir que les memòries TDP sempre se'ls assigna una BRAM36.

LUTs lògiques	LUTRAMs	SRLs	FFs	RAMB36	RAMB18
122	0	0	138	1	0

Taula 18: Recursos utilitzats amb el mòdul unificador amb la memòria interna TDP. Elaboració pròpia.

En la Figura 40 es mostra la modificació que es va afegir al mòdul unificador per tal que es pogués escollir quin tipus de memòria interna vol l'usuari. Com es pot veure en funció del paràmetre MEMORY_TYPE es pot escollir el tipus de configuració de ports.

Es va decidir fer una segona prova afegint el canvi de la memòria interna, s'ha tornat a generar el bitstream de la FPGA i s'han tornat a executar els testos associats al projecte. Novament ha sigut un èxit la incorporació del mòdul i aquest cop s'ha aconseguit unificar les dues memòries dels nivells 3.1 i 3.2 en una sola memòria BRAM18.

```
216     generate
217         if (MEMORY_TYPE == "TDP") begin
218             sram_tdp #(.PORTA_WR_DEPTH(SUM_DEPTH), .PORTA_WR_WIDTH(MAX_WIDTH), .OUTPUT_REG_A(OUTPUT_REG_A), .OUTPUT_REG_B(OUTPUT_REG_B)) INNER_SRAM (
219                         .PortA_Clk      (fast_clock       ),
220                         .PortA_Addr     (iAddr_sram_A     ),
221                         .PortA_WrData   (iWrData_sram_A   ),
222                         .PortA_RdData   (outputA[TOTAL_OUTPUT_RESOURCES-1]),
223                         .PortA_WrEn     (iWrEn_sram_A     ),
224                         .PortA_Ena      (iEna_A           ),
225 
226                         .PortB_Clk      (fast_clock       ),
227                         .PortB_Addr     (iAddr_sram_B     ),
228                         .PortB_WrData   (iWrData_sram_B   ),
229                         .PortB_RdData   (outputB[TOTAL_OUTPUT_RESOURCES-1]),
230                         .PortB_WrEn     (iWrEn_sram_B     ),
231                         .PortB_Ena      (iEna_B           )
232                     );
233         end else if (MEMORY_TYPE == "SDP") begin
234            sram_sdp
235             #(
236                 .WRDEPTH (SUM_DEPTH),
237                 .WRWIDTH (MAX_WIDTH),
238                 .RDWIDTH (MAX_WIDTH),
239                 .OUTPUT_REG(1'b0),
240                 .COMBINATORIAL_READ(1'b0),
241                 .BYTE_EN (1'b0)
242             ) INNER_SRAM (
243                 .WrClk  (fast_clock),
244                 .WrAddr (iAddr_sram_A),
245                 .WrData (iWrData_sram_A),
246                 .WrEn   (iEna_A),
247                 .WrByteEn(1'b0),
248                 .RdClk  (fast_clock),
249                 .RdAddr (iAddr_sram_B),
250                 .RdData (outputB[TOTAL_OUTPUT_RESOURCES-1]),
251                 .RdEn   (iEna_B),
252                 .RdRegEn(1'b0)
253             );
254         end
255     endgenerate
```

Figura 40: Actualització de la memòria interna del mòdul unificador. Elaboració pròpia.

En la Taula 19 es veuen els recursos utilitzats per aquesta última prova utilitzant una memòria SDP com a memòria interna del mòdul unificador. Com es pot veure en aquest cas la memòria interna és de 18Kb i no de 36 Kb.

LUTs lògiques	LUTRAMs	SRLs	FFs	RAMB36	RAMB18
122	0	0	138	0	1

Taula 19: Recursos utilitzats amb el mòdul unificador amb la memòria interna SDP. Elaboració pròpia.

En aquesta última prova s'ha aconseguit mitjançant el mòdul unificador reduir el nombre de memòries utilitzades, concretament s'ha passat de consumir dues memòries BRAM18 (36Kb en total) a consumir-ne una sola BRAM18 (18Kb en total).

13 Conclusions

En aquest capítol es mostren les conclusions extretes dels resultats del projecte, conclusions en l'àmbit personal i es parla sobre quin seria un treball futur a realitzar sobre aquest projecte.

13.1 Conclusions del projecte

L'objectiu principal d'aquest projecte era estudiar si era viable o no actualitzar el HW d'una impressora de llarg format, concretament intentar demostrar si se'n podia fer un millor ús de les memòries BRAM i URAM que tenen les FPGA. No s'ha pogut donar suport per URAM perquè no hi havia el HW necessari. Finalment s'ha acabat dissenyat, implementant i verificant el funcionament un mòdul en SystemVerilog capaç d'unificar memòries de tipus BRAM i per tant fer-ne un millor ús dels recursos de memòries disponibles.

Tant el disseny com la implementació d'aquest mòdul s'han basat en els criteris i requeriments de l'empresa, a més a més s'ha enfocat de tal manera que aquest pogués ser utilitzable en diferents circumstàncies sense importar el context en què s'utilitza. S'ha assolit aquest nivell de versatilitat gràcies al fe que la implementació s'ha parametritzat i per tant el mòdul és capaç d'ajustar la seva arquitectura en funció del context.

Per verificar el funcionament del mòdul s'ha simulat el seu comportament mitjançant una sèrie de tests. Cada test simulava el funcionament del mòdul en diferents contexts i a la vegada el verificava comparant el seu comportament amb una arquitectura equivalent. Els tests van permetre confirmar que tant el disseny com la implementació d'aquests era vàlida.

Per últim s'ha incorporat el disseny en un projecte de l'empresa. Aquesta prova final és la que ha permès verificar la viabilitat del canvi.

Tot i que l'objectiu principal era verificar la tecnologia UltraScale+ s'ha acabat enfocant cap a una verificació més genèrica de la unificació de memòries. Tot i això, s'han assolit tots els objectius referents al disseny, implementació i verificació del mòdul.

13.2 Conclusions personals

Tot i que aquest projecte podria ser millorable en molts aspectes em sento molt orgullós d'haver sigut capaç de tirar-lo endavant i haver aconseguit els resultats obtinguts.

Per altra banda penso que totes les assignatures que he estudiat relacionades amb l'especialitat d'Enginyeria de Computadors han pogut aportar el seu petit gra de sorra dins del desenvolupament d'aquest projecte, ja sigui durant el plantejament inicial amb els diagrames temporals com en les etapes més finals d'implementació i test.

Per últim, encara que per a molta gent el TFG sigui un tràmit, per mi ha sigut una experiència enriquidora tant a nivell de coneixements com en l'àmbit personal i crec que tindrà una gran repercussió en la meva forma de veure els projectes d'ara endavant.

13.3 Treball futur

Durant el desenvolupament del projecte s'han detectat punts que haguessin estat interessants desenvolupar.

Seria interessant que el mòdul donés suport a què la memòria interna sigui una URAM. En cas que pogués fer-se obriria noves possibilitats per desenvolupar projectes en el futur amb aquesta tecnologia que fins ara encara no s'ha utilitzat a l'empresa. Si es continués aquest projecte però enfocat més en l'ús de les URAM podria donar peu a què l'empresa s'interessés per aquesta tecnologia.

Un altre punt que podria desenvolupar-se més és el de disseny, en aquest projecte només s'ha implementat una versió de la capa de gestió de peticions, seria interessant generar un mòdul unificador amb una variant de l'opció proposada. De fet algunes de les variants es comenten: la primera variant és la d'afegir un *shift register* per emmagatzemar i gestionar les peticions, però s'ha desestimat perquè no entrava dins del termini del TFG. La segona variant és la d'afegir un multiplexor als registres que retarden la sortida i fer que aquests actualitzin el seu valor en funció de Counter, però aquesta variant afegeix complexitat i tampoc estava clar que fos d'ajut.

14 Referències

[1] Documentació de Xilinx, recursos de memoria. Última visita: 29/09/2020

https://www.xilinx.com/support/documentation/user_guides/ug573-ultrascale-memory-resources.pdf

[2] Documentació de Xilinx, especificacions per producte UltraScale+. Última visita: 29/09/2020

https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf

[3] Deloitte. Definició de *Waterfall*. Última visita: 29/09/2020

https://www2.deloitte.com/es/es/pages/technology/articles/waterfall-vs-agile.html

[4] Wikipedia. Definició de GIT. Última visita: 29/09/2020

https://ca.wikipedia.org/wiki/Git

[5] Sou conveni. Última visita: 12/10/2020

https://www.fib.upc.edu/ca/empresa/practiques-en-empresa

[6] Sou professorat. Última visita: 12/10/2020

https://www.upc.edu/transparencia/ca/informacio-de-personal/2019_TR_PDI.pdf

[7] Preu portàtil. Última visita: 12/10/2020

https://store.hp.com/SpainStore/Merch/List.aspx?sel=NTB&ctrl=f&fc_form_nb=1

[8] Preu pantalla. Última visita: 12/10/2020

https://www.pccomponentes.com/lg-22m45hq-b-22-led

[9] Cost serveis NoMachine. Última visita: 12/10/2020

https://www.nomachine.com/buyonline

[10] Subscripció Microsoft 365. Última visita: 12/10/2020

https://www.microsoft.com/es-es/microsoft-365/business/compare-all-microsoft-365-business-products

[11] Subscripció Githu Enterprise. Última visita: 12/10/2020

https://github.com/pricing#feature-comparison