UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

GRAU EN ENGINYERIA INFORMÀTICA

ENGINYERIA DE COMPUTADORS

TREBALL DE FI DE GRAU

Detecció d'errors i re-execució en unitats aritmètiques d'una CPU

Eric Rufart Blasco

Director: Ramon Canal Corretger

Departament d'Arquitectura de Computadors

28 de juny de 2021

Abstracte

En aquest projecte, s'implementa un sistema de detecció d'errors i re-execució a les unitats aritmètiques d'una CPU. Aquest es desenvolupa sobre una unitat central de processament (CPU) de l'estàndard RISC-V. Per dur-ho a terme, s'utilitza la tècnica de ”residue checking” i es compara amb duplicar la unitat aritmètica. Aquesta tècnica permet la reducció de la mida de la segona unitat aritmètica reduint, així, el seu consum energètic.

Amb aquesta finalitat, es realitza la gestió del projecte on s'estableixen els seus objectius i es planifica el projecte. A continuació, es tracten les diverses implementacions de les esmentades modificacions sobre el processador base. Un cop aquestes modificacions són implementades, s'analitza l'impacte que genera aplicar-les i, per acabar, es valoren les diferents versions comparant el seu impacte, els seus avantatges i desavantatges i la seva viabilitat.

Abstracto

En este proyecto, se lleva a cabo la implementación de un sistema de dectección de errores y re-ejecución en unidades aritméticas de una CPU. Esto se desarrolla en una unidad central de procesamiento (CPU) del estándar RISC-V. Para ello se utiliza la técnica de ”residue checking” y es comparada a duplicar la unidad aritmética. Esta técnica permite la reducción del tamaño de la segunda unidad aritméitca y reduciendo así su consumo energético.

Con esta finalidad, se realiza la gestión del proyecto donde se establecen sus objetivos y planifica el proyecto. A continuación, se tratan las implementaciones de dichas modificaciones sobre el procesador base. Una vez implementadas las modificaciones, se analiza el impacto que genera su inclusión y, para terminar, se valoran dichas versiones comparando su impacto, sus ventajas y desventajas y su viabilidad.

Abstract

In this project, the implementation of an error detection and re-execution method in the arithmetic unit of a CPU is carried out. This is developed on a central processing unit (CPU) based on the RISC-V standard. For this, the ”residue checking” technique is employed and it will be compared to duplicating the arithmetic unit. This technique allows the reduction of the size of the second arithmetic unit thus reducing its energy consumption.

With this objective in mind, a project management is carried out where its objectives are established and the project is planned. The implementations of these modifications on the base processor are treated and, once the modifications have been carried out, the impact generated by their implementation is analyzed. Finally, these versions are compared, comparing their impact on the base processor, their advantages and disadvantages, and their viability.

Índex

1 Context

Com a part d'un grau universitari és requerit, per tal de completar aquest, realitzar un projecte denominat *treball de fi de grau* que empri els subjectes tractats durant l'esmentat grau. Aquest document és el meu treball de fi de grau i es tracta d'un projecte per a l'especialitat enginyeria de computadors del grau en enginyeria informàtica de la Facultat d'Informàtica de Barcelona (FIB) de la Universitat Politècnica de Catalunya (UPC).

1.1 Tema d'investigació

A les unitats centrals de processament (CPU) a causa de diverses raons, com degradacions o fluctuacions del voltatge o l'impacte de partícules s'hi poden generar errors. Un dels llocs on poden succeir aquests errors és dins de les unitats aritmètiques d'aquestes. En aquest projecte s'utilitza la tècnica anomenada *residue checking* per comprovar errors en les susdites unitats. Addicionalment, s'ampliarà la lògica de control de la CPU per tal que en cas de detectar un error es tracti de la forma més adient. Aquesta implementació és compararà amb el mètode més directe que es bassa en duplicar les prèviament esmentades unitats per tal de comprovar els errors i s'avaluarà la seva viabilitat. Tot això es durà a terme a nivell de llenguatge descriptor de hardware (HDL) i s'hi avaluarà l'impacte en la freqüència de treball i en el consum energètic a conseqüència d'incorporar aquest mecanisme de detecció d'errors. Amb aquest objectiu s'utilitzarà una CPU basada en l'estàndard *RISC-V* sobre la qual s'hi faran les modificacions adients.

Aquests errors poden causar que els programes executats retornin un resultat incorrecte o, fins i tot, la caiguda de l'equip. Tot i que són relativament infreqüents i per l'ús general de sistemes són relativament insignificants, existeixen diverses aplicacions altament precises o claus que es beneficien d'implementar mesures de consistència addicionals. Per exemple, l'any 2003, a Bèlgica s'hi van realitzar unes votacions amb recompte digital i s'hi va detectar un *bit flip* que causava que un dels candidats tenia 4096, o el que és el mateix 2^{12}, vots de més [1]. En casos com aquest, sempre que l'error s'hagés causat a la unitat aritmètica, la tècnica a implementar podria haver detectat l'error.

Addicionalment, a l'espai, el problema de l'impacte de partícules, procedents de la radiació còsmica, s'incrementa donada l'absència del camp magnètic de la Terra que funciona com a capa protectora d'una alta proporció d'aquestes partícules.

1.2 Terminologia

1.2.1 Residue checking

Residue checking és una tècnica de comprovació d'errors que deriva d'utilitzar una propietat matemàtica dels nombres enters en aritmètica residual que defineix que per a dos nombres **a** i **b** respectivament el resultat d'operar els dos nombres i aplicar-los l'operació mòdul és equivalent a operar el resultat d'ambdós nombres operats amb el mateix mòdul, com és descrit a continuació:

$$(a \ \mathbf{op} \ b) \ mod \ N \equiv (a \ mod \ N) \ \mathbf{op} \ (b \ mod \ N)$$

1.2.2 Bit flip

Bit flip és com es denomina l'error que es dóna quan el valor d'un bit s'ha invertit inintencionadament. Això pot donar-se per diverses causes prèviament esmentades com la degradació o fluctuació del voltatge o un impacte de partícules.

1.2.3 RISC-V

RISC-V (RISC Five) és un projecte de hardware lliure. Aquest és un estàndard d'ISA, segles de l'anglès *Instruction Set Architecture* i es bassa en RISC, *Reduced Instruction Set Computer*. Una ISA és una especificació que detalla les instruccions que una CPU que segueix susdita ISA pot interpretar i executar.

Figura 1: Logo RISC-V

Aquest projecte va començar l'any 2010 a la universitat de California, Berkeley. Actualment es troba administrat per RISC-V International [2] i es distribueix sota una llicència BSD. Això permet que qualsevol pugui utilitzar-lo i modificar-lo sense necessitat de pagar per llicències i és una de les raons que hagi guanyat en rellevància durant els últims anys.

Aquesta ISA proposa un estàndard modular de processador, permetent, així, que dos dissenys amb diferents mòduls i, per tant, diferents capacitats segueixen adherint-se a l'estàndard. Aquesta característica ha impulsat la creació de processadors RISC-V i grans empreses tecnològiques han començat a adoptar aquesta ISA e incorporar-la als seus productes. Per exemple, NVIDIA es troba desenvolupant un xip basat en RISC-V per tal d'utilitzar-lo com a controlador

dins les seves GPU (graphics processing unit) [3]. Western Digital també es troba desenvolupant les seves versions de RISC-V i un dels usos que li donaran serà com a controladors de memòria a les seves memòries [4].

2 Justificació

Per aquest projecte s'implementa una tècnica no utilitzada prèviament donat que, previ a la recent aparició de RISC-V, la mancança de projectes de codi obert sobre CPUs dificultava l'accés a materials i dissenys ja realitzats al públic general. Addicionalment, la majoria de projectes es realitzaven per processadors incrementals -requerint compatibilitat amb totes les seves versions prèvies- on la complexitat dels sistemes és major dificultant aquesta. Un exemple d'aquests models incrementals és, el amplament estès, x86, un conjunt d'instruccions que es va crear l'any 1978 i s'ha anat ampliant el nombre d'instruccions que requereix que suporti. Actualment aquest ha de suportar més de 1300 instruccions. El creixement d'aquest es pot observar a la figura 2, incrementa la complexitat dels dissenys contínuament i dificulta la creació de processadors personalitzats donats els alts requeriments d'aquests.

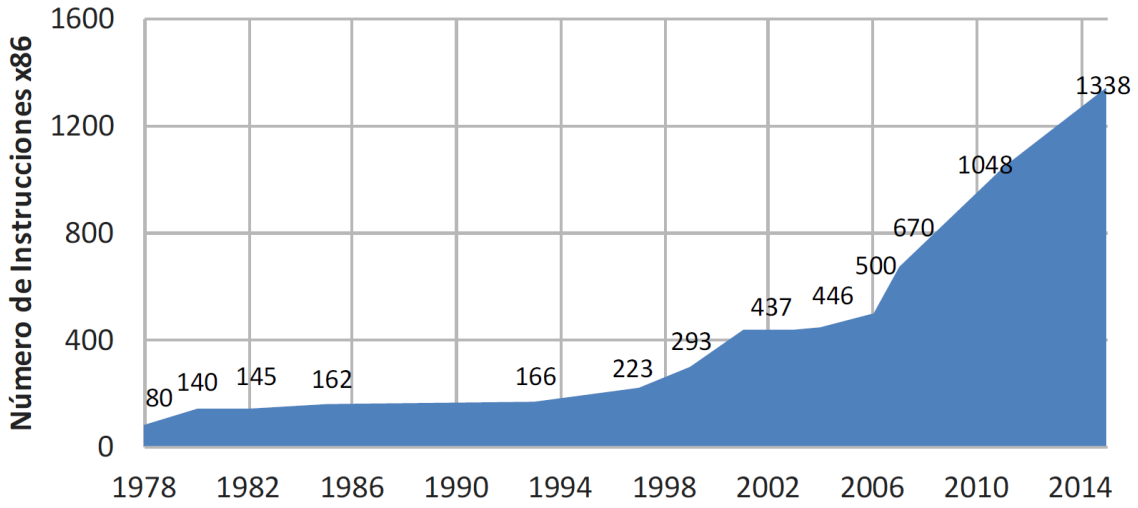

Figura 2: Creixement de x86.

Font: Guia practica de RISC-V pàgina 5 [5].

A contraposició, RISC-V es tracta d'un disseny de tipus modular. Això implica que només calgui suportar-se un conjunt d'instruccions base i les instruccions addicionals s'incloguin com a mòduls -anomenats extensions- de forma opcional, simplificant el disseny i la investigació realitzada sobre aquest. Però, donat el relativament curt període des de la seva creació no existeix una quantitat de recerca tan extensiva en aquest.

Per aquestes raons, conjuntament amb el fet que, per a processador d'ús general, no és una funcionalitat principal, hi existeix poca recerca a l'àmbit de la detecció d'errors d'aquest tipus hi manquen solucions implementades amb el mètode de *residue checking*.

Encara que existeixen diverses investigacions per a errors de consistència en CPUs, els esforços es troben a l'àrea de les memòries i no dins les unitats de processament. Els estudis existents centrats en utilitzar aquesta tècnica l'utilitzen per defensar-se de ciberatacs i no són aplicables a la utilitat intencionada.

La principal referència d'aquesta tècnica es troba al llibre "Residue arithmetic and its applications to computer technology" [6] on es proposen múltiples opcions per aplicar l'aritmètica residual al món de la computació i una d'aquestes és correspon a la utilització de *residue checking* com a mètode per comprovar errors en operacions aritmètiques i és la que defineix aquest projecte. Així i tot, aquest es tracta d'un llibre on es mostren diverses tècniques i possibles utilitats d'aquestes i no dissenya solucions per aquestes que es pugin aprofitar.

3 Abast

En aquesta secció es tracta l'abast d'aquest projecte i s'hi inclou una explicació més detallada de tots els temes a tractar en aquest document.

3.1 Introducció general a RISC-V i selecció del core

En primer lloc, s'explica com es divideixen les implementacions de RISC-V. Això ve donat pel fet que l'estàndard es troba dividit en una implementació bàsica, extremadament limitada i des d'aquesta base se li poden afegir diversos mòduls que permeten la personalització de les funcionalitats incorporades a cada implementació. El resultat d'aquesta modularització és la capacitat de tenir múltiples permutacions de funcionalitats.

A continuació, s'escull el core a modificar. Amb aquest objectiu s'hi analitzen diverses implementacions públicament disponibles i s'avalua les capacitats que aquestes disposen. Una de les principals característiques a avaluar serà els mòduls que implementa per tal de mantenir la complexitat d'aquesta implementació a un nivell adequat.

3.2 Comprensió del core

En aquesta secció s'hi inclou una descripció bàsica del funcionament de la implementació de RISC-V escollida. Aquesta descripció profunditza en les unitats aritmètiques i la lògica de control utilitzada en aquest core donat que són les principals seccions, on més endavant si realitzaran les modificacions necessàries.

3.3 Unitat de residue checking

En aquest apartat es tracta la implementació de la unitat que s'encarrega de fer el residue checking. Aquí s'hi tracten les diferents modificacions a realitzar per tal de poder afegir aquesta unitat i la implementació concreta d'aquesta.

3.4 Modificació de la lògica

En aquesta secció s'explica el funcionament de la lògica de control del core. Seguidament, s'hi expliquen les modificacions a realitzar per tal que en cas de detectar un error actuï de la manera adequada.

3.5 Anàlisis de l'impacte dels canvis

Finalment, s'analitza l'impacte dels canvis realitzats sobre el core original. S'hi comprova l'impacte sobre la freqüència de treball i el consum d'aquest. Aquí és on s'extreuen les conclusions del projecte.

3.6 Requeriments

Els requeriments esperats d'aquest projecte són:

- Disseny correcte que superi els tests durant la simulació.
- Disseny eficient amb el menor impacte respecte el disseny original.
- Anàlisi en detall de l'impacte del disseny.
- Avaluació del sistema implementat respecte a la seva alternativa.

3.7 Riscs i obstacles

Els possibles obstacles trobats durant el projecte són:

- Bugs: Els errors a la implementació o bugs són errors a afrontar durant el projecte que causen el mal-funcionament de la implementació i impliquen retards causats per la necessitat de solucionar-los.
- Desconeixement de la tecnologia: La manca de familiaritat amb les tecnologies emprades poden generar uns retards a la realització del projecte.
- Documentació: Una insuficient documentació implica un risc de retards futurs pel desconeixement del funcionament exacte de dissenys previament implementats.

4 Metodologia

Durant aquest projecte d'investigació s'utilitzarà la metodologia *Agile*. Amb aquesta s'organitza el projecte en diversos *sprints* de dos a tres setmanes cadascun.

Amb Agile es divideix el projecte en diverses etapes de curta durada anomenades sprints. A cada sprint s'hi realitzen les fases de planificació, anàlisis, disseny, implementació i avaluació. Això permet una planificació incremental del projecte.

El fet que cada sprint sigui de relativament curta duració permet que canvis al projecte es puguin aplicar sense descartar una amplia quantitat de planificació. És a dir, permet una ràpida adaptació davant de canvis. Addicionalment, la realització d'una avaluació al final de cada sprint permet reflexionar sobre l'estat del projecte i canvis a implementar.

Aquest projecte és dura a terme amb quatre sprints de tres setmanes i un cinquè sprint de dues setmanes. Els sprints seran els següents:

- Sprint 1: Durant el primer sprint es realitzarà la selecció del core i es familiaritzarà amb el funcionament d'aquest.
- Sprint 2: El segon sprint serà aquell en que es realitzarà la implementació de l'unitat de residue chekcing.
- Sprint 3: Al tercer sprint s'implementarà la lògica de control modificada al processador modificat. Addicionalment, es començaran els anàlisis dels impactes de les modificacions.
- Sprint 4: El quart sprint serà el període de temps per terminar els anàlisis dels impactes i finalitzar la mèmoria del TFG.
- Sprint 5: Durant l'últim sprint es realitzarà la preparació de la defensa del TFG.

4.1 Rigor

Respecte a la implementació es realitzaran múltiples tests de simulació del funcionament del disseny per tal de validar que aquest disseny sigui correcte. Aquests es corroboraran comparant-los amb els estats interns teòrics que hauriem de tindre.

Per la part de l'anàlisi, es realitzarà aquest a partir de les dades obtingudes de simular el disseny i del propi disseny amb un enfoc empíric i s'avaluarà que l'anàlisi s'adeqüi als requeriments inicials.

5 Planificació

Com s'ha esmentat prèviament aquest projecte es planificarà utilitzant una metodologia agile. El dia de la presentació del GEP serà considerat la data de començament d'aquest i la data de finalització serà definida per la data d'entrega de la memòria amb un marge de seguretat. Per tant, es considerarà el dia d'inici és el 23 de febrer de 2021, la data d'entrega de la memòria el 14 de juny de 2021 i la data de defensa el 28 de juny de 2021. El projecte es dividirà en múltiples tasques segons les diferents seccions del document. Aquestes es definiran a continuació amb una descripció d'aquesta i durant cada tasca s'hi inclou la generació d'una versió preliminar de la documentació que formarà part de la memòria.

5.1 T1: Gestió de projectes

La primera tasca del projecte corresponen a les entregues de l'assignatura de gestió de projectes (GEP). En aquest cas es conserven els lliuraments de l'assignatura de gestió de projectes com a subtasques d'aquesta.

5.1.1 T1.1: Abast i contextualització

Aquest és el primer lliurament a realitzar del GEP. Es tracta d'un document que contextualitza el projecte de fi de grau, el justifica i compara respecte a alternatives ja existents, defineix el seu abast i, per últim, descriu la metodologia que s'utilitza en aquest.

5.1.2 T1.2: Planificació temporal

Aquesta és la segona entrega a realitzar. En aquesta entrega es crea un document elaborant la planificació del projecte. En aquest cas, s'hi inclouen la divisió del projecte en tasques amb la seva durada predita i un diagrama de Gantt adaptat per la metodologia agile.

5.1.3 T1.3: Pressupost i sostenibilitat

Per la tercera entrega del GEP s'hi desenvolupa un document detallant els costos estimats del projecte i un informe de la sostenibilitat econòmic, social i ambiental del projecte.

5.1.4 T1.4: Document final

L'últim lliurament del GEP es tracta d'un document que comprèn els tres lliuraments anteriors modificats tenint en compte la retroacció rebuda en aquestes entregues.

5.2 T2: Introducció a RISC-V i selecció del core

L'objectiu d'aquesta tasca és escollir el core a modificar durant el projecte. Amb aquest objectiu s'hi duen a terme les següents accions.

5.2.1 T2.1: Introducció a RISC-V

Durant aquesta tasca s'introdueix RISC-V i es recerca les diverses versions de RISC-V amb l'objectiu de documentar les característiques d'aquestes facilitant així l'elecció del core en funció de quines característiques disposi.

5.2.2 T2.2: Recerca dels cores disponibles

A continuació, s'hi recerquen diverses opcions de cores disponibles per modificar i s'hi documenten les seves característiques.

5.2.3 T2.3: Selecció del core

Per finalitzar la tasca, s'escull el core. Amb aquest objectiu s'analitzaran les característiques desitjades del core escollit i les pròpies característiques implícites d'aquest.

5.3 T3: Comprensió del core

En aquesta tasca s'hi documentarà el core i el seu funcionament. L'objectiu és fer una introducció al funcionament del processador i com s'ha implementat el core escollit.

5.4 T4: Unitat de residue checking

Un cop entès el funcionament es realitza la tasca 4. En aquesta tasca es desenvolupa la unitat de residue checking i es documenta el procés d'implementació d'aquesta. Aquesta unitat és la que realitza el càlcul del residu per l'operació mòdul.

5.4.1 T4.1: Fonaments

Primer, es detalla tota la informació necessària per la implementació de la unitat. Això comprèn teoria sobre el seu funcionament i informació concreta sobre la implementació del RISC-V.

5.4.2 T4.2: Creació la unitat de residue checking

Correspon al procés de disseny de la unitat de residue checking i la documentació d'aquest procés i de la unitat dissenyada.

5.4.3 T4.3: Acoblament de la unitat

En aquesta etapa s'acobla la unitat generada prèviament al core escollit.

5.5 T5: Modificació de la lògica

L'ultima modificació que cal implementar és la modificació de la lògica de control del core.

5.5.1 T5.1: Funcionament de la lògica de control

En aquesta etapa es documenta el funcionament original de la lògica de control del processador.

5.5.2 T5.2: Especificació de la lògica modificada

A continuació, es descriuen les modificacions a aplicar a la lògica i el raonament d'aquestes.

5.5.3 T5.3: Implementació de la lògica modificada

Per acabar, s'implementen les modificacions a la lògica al processador i es documenta el procés d'implementació.

5.6 T6: Anàlisis de l'impacte dels canvis

L'ultima secció d'aquest projecte és l'anàlisi de l'impacte causat pels canvis introduïts al core original. En aquesta tasca es fan aquests anàlisis.

5.6.1 T6.1: Anàlisi de freqüència

Primerament, s'analitza els retards introduïts al processador per les modificacions realitzades i s'analitzen com afecten aquests a la freqüència de funcionament d'aquest.

5.6.2 T6.1: Anàlisi de consum

Per acabar, s'analitza l'impacte sobre el consum de dit processador.

5.7 T7: Finalització de la memòria

La finalitat d'aquesta tasca és finalitzar la memòria, incorporant tota la documentació prèviament generada en un únic i complet document final i corregir errors en aquesta.

5.8 T8: Preparació de la defensa

Per finalitzar, cal generar la presentació del TFG i prepar-se la defensa d'aquest. L'ultima tasca és la tasca on es duu aquest procés.

6 Estimacions

6.1 Estimació de la durada de les tasques

Identificador	Nom	Dependències	Durada
T1	Gestió de projectes		65h
T1.1	Abast i contextualització		25h
T1.2	Planificació temporal	T1.1	10h
T1.3	Pressupost i sostenibilitat		10h
T1.4	Document final	T1.1, T1.2, T1.3	20h
T2	Introducció a RISC-V i selecció del core		80h
T2.1	Introducció a RISC-V		30h
T2.2	Recerca dels cores disponibles	T2.1	30h
T2.3	Selecció del core	T2.2	20h
T3	Comprensió del core	T2	45h
T4	Unitat de residue checking	T3	110h
T4.1	Fonaments		35h
T4.2	Creació de la unitat de residue checking	T4.1	45h
T4.3	Acoblament de la unitat	T4.2	30h
T5	Modificació de la lògica	T4	80h
T5.1	Funcionament de la lògica de control		35h
T5.2	Especificació de la lògica modificada	T5.1	25h
T5.3	Implementació de la lògica modificada	T5.2	30h
T6	Anàlisis de l'impacte dels canvis	T5	60h
T6.1	Anàlisi de freqüència		30h
T6.2	Anàlisi de consum		30h
T7	Finalització de la memòria	T6	40h
T8	Preparació de la defensa	T7	50h
TFG	Treball de fi de grau		530h

Taula 1: Estimació de la durada de les tasques.

Elaboració pròpia.

6.2 Gantt

Al proper diagrama de Gantt el final de la setmana el final de la setmana 16 és el dia 14 de juny i la setmana 18 representa el dia 28 de juny. Aquest són respectivament els dies esperats per entregar la memòria i defensar el TFG.

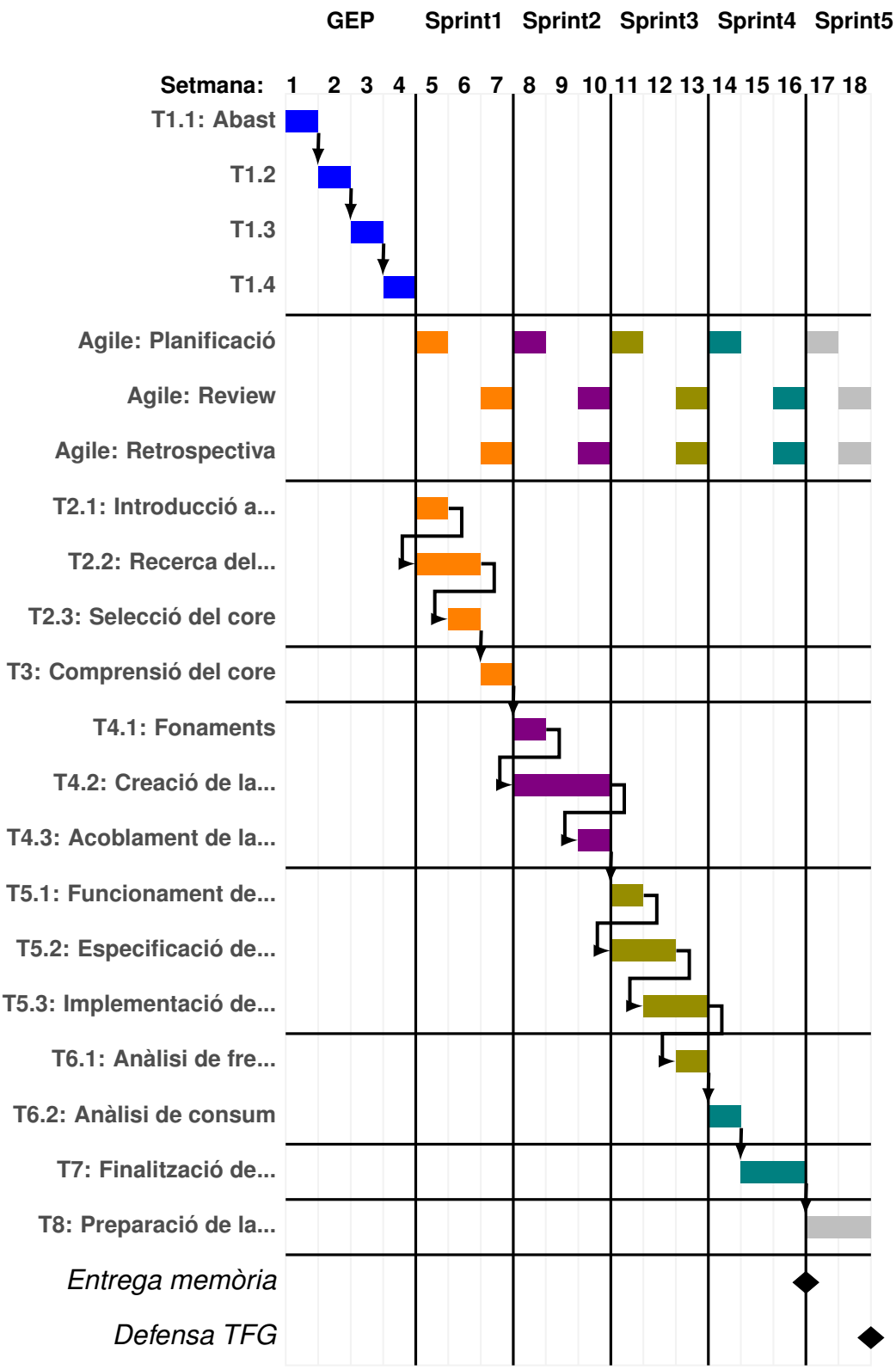

Figura 3: Diagrama de Gantt de les tasques

Elaboració pròpia.

6.3 Gestió de riscs

Donat a diverses raons, un canvi a la planificació original pot ser necessari. Amb aquest objectiu s'ha tingut en compte un període d'una setmana durant la planificació. És a dir, tot i que la data de lliurament de la memòria seria el dia 21 de juny, s'ha planificat com si aquesta fos el dia 14, això acompte per retards inesperats. Els possibles retards inesperats provenen de bugs i la falta de familiaritat amb el core escollit.

La falta de familiaritat s'espera que tingui baixa probabilitat donat que s'assigna una tasca a comprendre el funcionament d'aquest, però, donat que la tasca es focalitza en desenvolupar la documentació d'aquest per la memòria, familiaritzarse amb aquest és un benefici col·lateral en comptes del centre d'aquesta. Amb aquesta raó se li assigna una baixa probabilitat de succeir amb 20 hores de retard.

L'altre possible causa de retards és l'aparició de bugs al disseny de les modificacions. A aquest tipus de retard se li assigna una probitat mitja-alta i, en aquest cas, es calculen un màxim de 40 hores de retards possibles causats per bugs.

7 Pressupost

Amb la realització d'un projecte s'associen uns costos materials en aquesta secció s'analitza el cost previst de desenvolupament i els factors que afecten a aquest.

7.1 Costs de personal

El primer tipus de cost ve associat amb el personal utilitzat pel projecte. En aquest projecte hi participarien un cap de projecte i un enginyer de hardware.

L'enginyer de hardware s'encarrega de tota tasca relacionada al desenvolupament del projecte i la documentació d'aquest. Per altra banda, al cap de projecte li són assignades les tasques relacionades amb planificació i la defensa del projecte.

Pel que fa als salaris s'utilitzen els salaris mitjans per l'àrea de Barcelona segons Glassdoor. Pel que fa al enginyer de hardware s'utilitzen 41.198€ [7] i pel cap de projecte es calculen 48.923€ [8]. Aquest són els salaris anuals i es convertirà a sou per hora dividint entre les 1750 hores aproximades de les que constaria un treball al llarg d'un any. Addicionalment, es multiplicaran per 1.3 per tindre en compte els costs de la seguretat social.

Càrrec	Anual	Per hora	Amb S.S
Enginyer de hardware	41.198€/any	23.54€/h	30.60€/h
Cap de projecte	48.923€/any	27.95€/h	36.34€/h

Taula 2: Sous dels càrrecs

Elaboració pròpia.

Amb aquests salaris juntament amb l'estimació del temps de treball per empleat es calculen els costs en personal del projecte com es pot observar a la propera taula. En aquesta es classifica els costos per cada tasca depenent de les hores que cada empleat es necessari.

Identificador	Nom	Enginyer de Hardware	Cap de projecte	Durada	Cost
T1	Gestió de projectes	0	65	65h	2.362€
T2	Introducció a RISC-V i selecció del core	75	5	80h	2.477€
T3	Comprensió del core	40	5	45h	1.405€
T4	Unitat de residue checking	105	5	110h	3.394€
T5	Modificació de la lògica	75	5	80h	2.477€
T6	Anàlisis de l'impacte dels canvis	55	5	60h	1.865€
T7	Finalització de la memòria	35	5	40h	1.252€
T8	Preparació de la defensa	0	50	50h	1.817€
TFG	Treball de fi de grau	385	145	530h	17.049€

Taula 3: Cost en personal per tasca

Elaboració pròpia.

7.2 Costs materials

Un cop comptabilitzats els costs del personal, queda comptabilitzar els costs del programari i els materials. En aquest cas només es necessari un ordinador donat que tota execució es farà utilitzant un simulador i es tractarà de fer l'anàlisi de manera teòrica basant-se en informació procedent del disseny creat.

Per tant, el cost material s'estima de 1000€. Aquest s'amortitza segons la següent formula:

$$Amortitzacio = Preu * Durada\ del\ projecte/(Anys * Hores\ laborals\ anuals)$$

Donats els límits imposats per hisenda estimarem la esperança de vida d'aquest en 4 anys. Amb aquestes dades i utilitzant l'estimació prèvia de 1750h de treball anuals l'amortització equival a 75.71€.

$$1000 * 530h/(1750h * 4\ anys) = 75.71$$

7.3 Costs en programari

L'últim cost restant és el cost en programari. Donat que el core no es troba escollit durant el procés de planificació, el llenguatge descriptor de hardware pot variar. Com a conseqüència no es pot donar una estimació del cost adient. Per tant, es calcula un cost de 1500€ màxim en llicencies de programari.

7.4 Cost de contingència

Per terminar, s'hi ha de calcular un cost de contingència per impediments que puguin sorgir durant el desenvolupament. En aquest cas, s'ha escollit una contingència del 20%. Donats els costs prèviament calculats de 17.049€, 1000€ i 1500€ pel, respectivament, personal, material i programari. Obtenim que el cost del projecte és de 19.549€. Un 20% d'aquest equival a 3.910€. Amb aquest valors calculats s'estableix el cost del projecte en 23.459€

7.5 Control de gestió

Aquest pressupost, degut a l'aparició de riscos no identificats o desviacions als càlculs dels riscos emprats, poden no complir-se. Per tal de calcular les desviacions amb l'original s'utilitzen les següents formules:

- Desviació a la durada de les tasques = Cost * (Hores estimades - Hores reals)
- Desviació del cost en personal per tasca: (Cost estimat – Cost real) * Hores
- Desviació en hores: Hores estimades – Hores reals
- Desviació total dels costs: Total estimat – Total real.

8 Lleis reguladores

Donat el caràcter d'investigació d'aquest projecte, l'única llei aplicable és la llei de propietat intel·lectual. En aquest cas afecta a causa de l'ús d'una implementació de RISC-V ja existent. La implementació utilitzada es tracta de CV32E40P, aquesta es troba sota una llicència Solderpad Hardware License, Versió 0.51. Aquesta llicència permet l'ús d'aquesta implementació per a aquest projecte, cito:

> "*Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable license under the Rights to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form and do anything in relation to the Work as if the Rights did not exist.*"

9 Sostenibilitat

9.1 Autoavaluació

L'exercici d'autoavaluació dels coneixements propis és en si un exercici pràcticament fútil. Les raons d'aquesta futilitat venen donades per l'efecte Dunning-Kruger[9]. Aquest descriu com els coneixements que té una persona i les que creu tindre difereixen en la gran majoria dels casos. En concret, quant menys coneixements d'un tema es tenen, més fàcil és de sobreestimar els coneixements propis. Per altra banda, a rangs de coneixement més profunds es té tendència de subestimar els coneixements propis.

Dit això, donada la meva persona poden considerar-se punts forts l'anàlisi de la sostenibilitat dels projectes. Per altra banda, els punts febles són aquells en que cal proposar alternatives o projectes sostenibles, és a dir, aquells més relacionats amb la creativitat respecte anàlisi empíric.

9.2 Dimensió econòmica

El pressupost prèviament definit incorpora totes les variables necessàries per tal d'avaluar la viabilitat d'un projecte.

En aquest cas, aquest projecte s'orienta a investigar els impactes d'aplicar una tècnica de correcció d'errors sobre les unitats aritmètiques. Aquest no es troba orientat a millorar una alternativa ja existent sinó que ofereix una mesura de consistència addicional a les existents per altres components, com, per exemple, les memòries ECC (Error-Correcting Code), per a les unitats aritmètiques.

9.3 Dimensió mediambiental

Donada la orientació del projecte aquest no requereix materials per a la seva realització. Això implica que l'únic impacte mediambiental que implica aquest projecte és el degut a l'ús d'electricitat.

9.4 Dimensió social

Aquest projecte proposa una solució a possibles errades a les unitats aritmètiques de processadors. Això permet afegir una capa de consistència addicional que incrementa la correcció dels resultats i beneficia a qualsevol operació que requereixi alta precisió i consistència.

10 Unitat de càlcul del residu

En aquest apartat es discuteixen diversos mètodes per convertir un nombre al seu valor en aritmètica residual. Els mètodes a tractar són el càlcul mitjançant l'operació divisió, el còmput ràpid del mòdul i la implementació del càlcul mitjançant les propietats dels nombres múltiples de dos i dels nombres primers de Mersenne.

10.1 Divisió modificada

El primer mètode es tracta d'utilitzar el valor de residu obtingut des d'operacions de divisió ja implementades. Tot i que la versió del core escollida no suporta per defecte l'operació mòdul, una senzilla modificació a l'operació divisió permet la implementació d'aquesta. El principal desavantatge d'aquest mètode es troba en el seu alt cos de computació, equiparant-lo al cost de l'operació divisió la implementació escollida requereix entre 3 i 35 cicles del processador per obtenir el resultat.

10.2 Múltiples de dos

Aquest mètode utilitza les característiques dels nombres múltiples de dos en binari per calcular el mòdul. L'avantatge d'aquest mètode és trivialitzar el càlcul del mòdul. El seu inconvenient és que es perden els bits no utilitzats per calcular aquest.

10.3 Nombres primers de Mersenne

Els nombres primers de Mersenne són aquells nombres primers que es representen de forma $2^n - 1$. Els nombres 3 i 7 són exemples d'aquest donat que la seva representació és "11" i "111" respectivament. Aquests nombres permeten el càlcul utilitzant la fórmula $x \mod z = (x/(z+1) + x \mod (z+1)) \mod n$ donat que z+1 es tracta d'un nombre de forma 2^n aquest càlcul és el més eficient computacionalment donat que només requereix $\lfloor \frac{a}{n} \rfloor + 1$ sumes essent "a" el nombre de bits del nombre "x" i "n" l'exponent que forma el nombre "z" donat $z = 2^n - 1$. Aquest nombre sorgeix del fet que en calcular el mòdul, l'últim càlcul de la dita forma sempre serà menor que $(2^n - 1) * 2$. Aquest mètode serà l'escollit finalment i s'exemplificarà el seu funcionament durant la secció d'implementació.

10.4 còmput ràpid del mòdul

[11] exposa diversos mètodes per a computar el mòdul d'un nombre x mod z. Aquest permet el còmput del mòdul utilitzant n-m etapes on n és el nombre de bits de x i m el nombre de bits de z. Aquesta operació la realitza mitjançant LUT (LookUp Tables) per accelerar el temps de còmput. La desavantatge principal d'aquest mètode és el requeriment de LUTs de mida escalable amb la quantitat de bits del nombre.

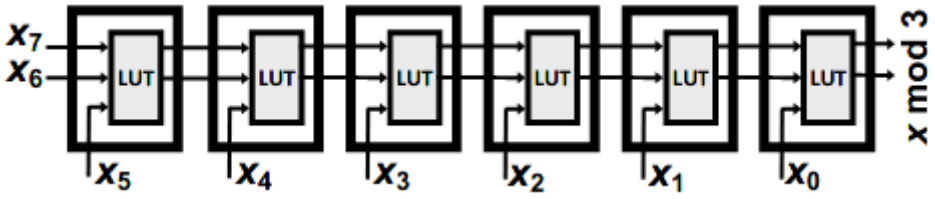

Figura 4: còmput del mòdul utilitzant LUTs

Font: [11]

10.5 Elecció

L'objectiu d'aquest projecte és la comprovació d'errors dins les unitats aritmeticològiques de les CPUs. Amb aquest objectiu sorgeixen diversos requeriments, el principal requeriment és la menor afectació sobre la freqüència de treball possible. Tampoc pot generar un alt consum energètic ja que la tècnica s'empra per intentar reduir el consum generat simplement duplicant les unitats aritmeticològiques.

Per això, el mètode que ha sigut escollit és la utilització d'una nombre primer de Mersenne com a base. Les raons darrere d'aquesta decisió són la inviabilitat dels dos primers mètodes la divisió modificada i l'ús de nombres múltiples de dos donat el cost en cicles i la pèrdua de bits d'aquests. L'elevat nombre de recursos necessaris pel còmput ràpid del mòdul en comparació amb l'ús d'un nombre primer de Mersenne juntament amb la ràpida velocitat de càlcul d'aquest han estat els altres factors a l'hora de decidir el mètode.

Respecte al nombre utilitzat com a base s'ha decidit utilitzar el nombre 131071 o el que és el mateix el nombre primer de Mersenne de 17 bits. Aquest nombre permet el càlcul del residu utilitzant únicament dos sumadors -dos sumadors addicionals són requerits per corregir pel funcionament de complement a dos- permetent, així, el càlcul del mòdul a gran velocitat. No s'incorpora l'ús d'un segon nombre donat que la incorporació d'aquest implicaria una ALU addicional i el seu cost en potencia ja superaria el de senzillament duplicar l'ALU original.

10.6 Representació dels nombres negatius

La representació dels nombres negatius en aritmètica residual genera dues possibilitats a l'hora d'expressar aquests nombres. La primera possibilitat és la implementació de residus estrictament positius o la utilització de residus negatius. En la implementació realitzada s'a escollit utilitzar nombres estrictament positius degut a la seva facilitat d'implementació aquesta es troba més detalla al següent apartat.

Per tant, l'operació mòdul es defineix utilitzant la següent expressió:

$$x \mod n \equiv n * q + R \tag{1}$$

On $n \in \mathbb{N}$, $q \in \mathbb{Z}$ i $0 \leq R < n$.

10.7 Implementació

El codi següent, realitzat amb SystemVerilog, calcula el mòdul de l'entrada Op_I i $2^{17} - 1$. Per fer-ho utilitza la formula $x \mod (2^{17} - 1) = \frac{x}{2^{17}} + x \mod 2^{17}) \mod (2^{17} - 1)$. Donades les característiques de binari, obtenir el residu i el quocient amb 2^{17} és trivial.

```
module cv32e40p_mod #(
    parameter C_WIDTH     = 32
) (

    input  logic [    C_WIDTH-1:0] Op_I,
    output logic [    16:0] NegMod_17,
    output logic [    16:0] Mod_17
);

  logic [17:0] High_17, Low_17, Add_17, Subbed_17, Out_17,
               OutNeg_17;
  logic [16:0] OutNegoverflow_17;

  assign Low_17 = {1'b0,Op_I[16:0]};
  assign High_17 = {3'b000,Op_I[31:17]};
  assign Add_17  = Low_17 + High_17;
  assign Subbed_17 = Add_17 - 18'h1ffff;
  assign Out_17 = Subbed_17[17] == 1'b1 ? Add_17 : Subbed_17;
  assign Mod_17 = Out_17[16:0];
  assign OutNeg_17 = Out_17 - 18'h08000;
  assign OutNegoverflow_17 = Out_17[16:0] - 17'h08001;
  assign NegMod_17 = OutNeg_17[17] == 1'b1 ? OutNegoverflow_17 :
                                             OutNeg_17[16:0];
```

Respecte a l'últim mòdul de la funció s'aplica sobre una suma la qual el seu resultat és menor a $2 * (2^{17} - 1)$ per tant simplement restant-li $2^{17} - 1$ es pot calcular el seu valor utilitzant un multiplexor que depengui del bit de més pes per comprovar quin dels dos nombres és menor, que $2^{17} - 1$. Per exemple,

La implementació de la sortida negativa realitza una resta de 2^n on n és el nombre de bits del valor original menys el nombre de bits del primer de Mersenne -essent aquests 32 i 17 en aquest cas per un total de 15- sobre el resultat prèviament obtingut. Aquesta resta corregeix l'error incorporat pel complement a dos als nombres negatius. La propera taula demostra el funcionament del càlcul del residu per a x mod 7 on x és un nombre de 4 bits. En aquesta la correcció d'error es tracta de 2^1 donat que l'exponent prové de la diferència entre el nombre de bits 4-3=1. Per exemple, -8 mod 7, seguint la definició prèviament donada, ha de resultar en un residu de 6. Sense correcció aquest resulta en 1 i un cop aplicada la correcció $6 \mod 7 \equiv (1-2) \mod 7$. També pot observar-se que OutNeg utilitza un bit més que la sortida, aquest s'utilitza per corregir el cas on es realitza undreflow, utilitzant-se OutNegOverflow on es corregeix amb $2^n + 1$ donat que al realitzar underflow es crossa el valor del divisor i aquest no forma part dels residus possibles.

Binari	Natural	Mod 7	Enter	NegMod 7	Out7	OutNeg7	OutNeg Overflow7
0000	0	0	0	X	X	X	X
0001	1	1	1	X	X	X	X
0010	2	2	2	X	X	X	X
0011	3	3	3	X	X	X	X
0100	4	4	4	X	X	X	X
0101	5	5	5	X	X	X	X
0110	6	6	6	X	X	X	X
0111	7	0	7	X	X	X	X
1000	8	1	-8	6	001	1111	110
1001	9	2	-7	0	010	0000	111
1010	10	3	-6	1	011	0001	000
1011	11	4	-5	2	100	0010	001
1100	12	5	-4	3	101	0011	010
1101	13	6	-3	4	110	0100	011
1110	14	0	-2	5	000	1110	101
1111	15	1	-1	6	001	1111	110

Taula 4: Càlcul de x mod n per n=7

Elaboració pròpia

10.8 Verificació

```
verilator -Wall --cc -CFLAGS "-g" --trace --sv cv32e40p_mod.sv \
        --exe sim_cv32e40p_mod.cpp
cd obj_dir
make -j -C obj_dir -f Vcv32e40p_mod.mk Vcv32e40p_mod
cp obj_dir/Vcv32e40p_mod testbench_verilator
```

La verificació d'aquest circuit es realitza utilitzant verilator. El script anterior genera un fitxer executable testbench_verilator que al executar-se avalua el model seguint les característiques del codi en C++ localitzat a l'annexe. En aquest s'avalua el model pels nombres entre -131071000 i 131071000 -no s'inclouen tots els nombres en 32 bits a causa d'un requeriment d'hores per avaluar aquesta quantitat de models- i escriu per pantalla els resultats junt amb el seu valor teòric. Addicionalment, el programa termina si es detecta una discrepància entre aquests valors i escriu el resultat teòric, l'actual el valor calculant-se abans d'aturar-se.

```
modulo 2^17-1 de -130910698 = 29231 k 29231
modulo 2^17-1 de -130910697 = 29232 k 29232
modulo 2^17-1 de -130910696 = 29233 k 29233
modulo 2^17-1 de -130910695 = 29234 k 29234
modulo 2^17-1 de -130910694 = 29235 k 29235
modulo 2^17-1 de -130910693 = 29236 k 29236
modulo 2^17-1 de -130910692 = 29237 k 29237
```

Figura 5: Sortida parcial del testbench

Elaboració pròpia

11 Introducció a RISC-V

RISC-V es tracta d'un projecte de repertori d'instruccions, ó ISA, de maquinari lliure (open-source). És un conjunt d'instruccions de tipus modular i es pot dividir en mòduls base i extensions. Aquest disseny modular permet simplificar el disseny dels microprocessadors basats en aquest donat que no han d'incloure suport per operacions innecessàries per la tasca per la qual són dissenyats. Addicionalment, això, disminueix la mida del *die*, disminuint, com a conseqüència, el cost de fabricació i l'impacte ambiental donat que es requereix menys material per die juntament amb el fet que a major mida de die major és el percentatge de xips defectuosos.

La segona conseqüència del disseny modular és una de les raons per optar a utilitzar RISC-V, la simplicitat d'aquest disseny modular facilita la validació de dissenys realitzats per aquest. Una altra raó és que donada la simplicitat del disseny el manual de l'especificació es pot contenir en unes 200 pàgines a contraposició de les més de 2000 pàgines necessàries que ocupen els manuals d'ISA incrementals com x86 facilitant familiaritzar-se amb aquest.

El repertori d'instruccions d'una implementació de RISC-V concreta és format per una selecció dels mòduls exposats a la taula 4. En aquesta taula un estat "congelat" implica que no s'esperen modificacions als mòduls prèviament a la ratificació del document. Aquests mòduls es defineixen més endavant en aquest document i les diferències entre les versions resultants depenent dels mòduls que implementen són importants a l'hora d'escollir el core sobre el que es farà la recerca.

Base	Versió	Estat
RVWMO	2.0	Ratificat
RV32I	2.1	Ratificat
RV64I	2.1	Ratificat
RV32E	1.9	Esborrany
RV128I	1.7	Esborrany
Extensió	**Versió**	**Estat**
Zifencei	2.0	Ratificat
Ziscr	2.0	Ratificat
M	2.0	Ratificat
A	2.0	Congelat
F	2.2	Ratificat
D	2.2	Ratificat
Q	2.2	Ratificat
C	2.0	Ratificat
Zsto	0.1	Congelat
Counters	2.0	Esborrany
L	0.0	Esborrany
B	0.0	Esborrany
J	0.0	Esborrany
T	0.0	Esborrany
P	0.2	Esborrany
V	0.7	Esborrany
Zam	0.1	Esborrany

Taula 5: Tabla

Font: Prefaci de l'especificació RISC-V [10].
Addicionalment a aquests mòduls, el mòdul "G" es pot utilitzar com a acurtament de "IMAFD"

11.1 Mòduls Base

Aquests defineixen els conjunts d'instruccions bàsics que un core basat en RISC-V ha de suportar. A excepció del mòdul RVWMO aquests són incompatibles entre ells donat que cadascun es tracta d'un conjunt d'instruccions base.

11.1.1 RVWMO

La raó que aquest mòdul no sigui incompatible amb els altres és que, tot i que es defineix amb els mòduls base, aquest mòdul és un model de consistència de memòria, anomenat RISC-V Weak Memory Ordering, en comptes d'un conjunt d'instruccions.

11.1.2 RV32I

Aquesta es la variant base de RISC-V, es tracta d'un conjunt d'instruccions per un processador de 32 bits amb trenta-dos registres de propòsit general.

11.1.3 RV64I

La variant RV64I amplia la variant base RV32I canviant la mida dels registres i l'espai d'adreces de memòria a 64 bits. Addicionalment, inclou noves instruccions per suportar correctament les operacions de 64 bits.

11.1.4 RV32E

Es tracta d'una variant del model RV32I dissenyat per microcontroladors encastats. En aquesta variant la quantitat de registres de propòsit general es veu reduïda de trenta-dos a setze. Aquesta modificació redueix en aproximadament un 25% l'àrea del xip reduint així el seu consum i alliberant bits del conjunt d'instruccions per a possibles extensions personalitzades.

11.1.5 RV128I

La variant RV128I es crea imitant al procés de creació de RV64I. Però, en aquest cas la mida dels registres i l'espai d'adreces de memòria s'incrementen a 128 bits en comptes dels 64 bits de RV64I.

11.2 Mòduls d'extensió

Es tracta de mòduls opcionals oficials que poden incloure's depenent de la funcionalitat objectiu del processador.

11.2.1 Zifencei

Aquest mòdul ve format per la instrucció "FENCE.I" que s'encarrega de la sincronització entre els canals d'instruccions i dades. És un mòdul de coherència de memòria dins d'un fil de hardware, de l'anglès hardware thread.

11.2.2 Ziscr

El mòdul Ziscr afegeix un gran nombre de registres de control i estat juntament amb instruccions noves per permetre l'ús d'aquests registres.

11.2.3 M: Multiplicació i divisió d'enters

El mòdul de multiplicació i divisió d'enters afegeix noves instruccions per operar d'aquestes formes amb enters situats als registres de propòsit general.

11.2.4 A: Instruccions atòmiques

En aquest mòdul es proposa noves instruccions per tal de realitzar lectures i escriptures de memòria sincronitzant diferents fils de hardware que comparteixen una única memòria.

11.2.5 F: Coma flotant de precisió simple

Aquest mòdul depèn de l'extensió "Ziscr" i afegeix suport per operacions amb nombres decimals de 32 bits.

11.2.6 D: Coma flotant de precisió doble

Aquesta extensió construeix a partir de la prèvia extensió de coma flotant de precisió simple utilitzant registres de 64 bits en comptes de 32 bits.

11.2.7 Q: Coma flotant de precisió quàdruple

En aquest cas s'actua com prèviament al mòdul de coma flotant de precisió doble utilitzant registres de 128 bits per obtenir una major precisió.

11.2.8 C: Instruccions comprimides

El mòdul d'instruccions comprimides proposa suport per instruccions de 16 bits de llargada.

11.2.9 Zsto

Aquesta extensió proposa modificacions al mòdul RVWMO definint el model de memòria RVTSO (RISC-V Total Store Ordering(en aquest.

11.2.10 Counters

Amb aquest mòdul s'afegirien registres per utilitzar-se com comptadors i rellotge. Addicionalment, s'afegirien instruccions noves que permeten interactuar amb aquests registres..

11.2.11 L: Coma flotant decimal

Aquest és un mòdul que proporciona suport per coma flotant codificada en decimal.

11.2.12 B: Manipulació de bits

Es tracta d'un mòdul per manipulació de bits concrets a registres.

11.2.13 J: Llenguatges dinàmics

El mòdul de llenguatges dinàmics amb l'objectiu d'implementar operacions que millorin l'eficiència d'aquests i els seus garbage collectors.

11.2.14 T: Memòria transaccional

És un mòdul que s'implementarà amb l'objectiu de permetre l'execució de lectures i escriptures de forma atòmica.

11.2.15 P: Packed-SIMD

Es tracta d'un mòdul que permet l'execució d'instruccions packed-SIMD (Single Instruction, Multiple Data).

11.2.16 V: Operacions vectorials

El mòdul d'operacions vectorials proposa suport per a l'execució d'instruccions amb paralelisme de dades.

11.2.17 Zam

El mòdul Zam amplia el previ mòdul d'instruccions atòmiques amb suport per a operacions atòmiques de memòria no alineada.

12 Cores

Degut a que RISC-V es tracta d'un projecte que separa l'arquitectura de la implementació cal escollir la implementació sobre la qual es realitzarà aquest projecte d'investigació. En aquest apartat es detallen les diverses opcions considerades i s'explica el procés pel qual es decideix la implementació a utilitzar.

SweRV EH1

Llenguatge: SystemVerilog
SweRV EH1 es tracta d'un core RV32IMC superescalar desenvolupat per Western Digital que permet la petició de dues instruccions simultàniament i es troba desenvolupat sota una llicencia Apache 2.0.

SweRV EL2

Llenguatge: SystemVerilog
SweRV EL2 es tracta d'un core RV32IMC igual que la implementació previament explicada es troba desenvolupat per Western Digital sota una llicencia Apache 2.0, però es tracta d'una implementació més senzilla de tipus superpipelined, és a dir, que només permet la petició d'una instrucció a cada cicle del processador.

CV32E40P

Llenguatge: SystemVerilog
Implementant els mòduls RV32IM[F]C, CV32E40P formava part, conjuntament amb els propers CVA6 i Ibex, d'un projecte anomenat PULP. Mantingut originalment per la PULP platform ha sigut recentment contribuït al OpenHW Group. Desenvolupat amb una llicencia SolderPad Hardware License v.0.51, es tracta d'una implementació petita i eficient.

CVA6

Llenguatge: SystemVerilog
RV[32/64]GC formava part del projecte PULP i recentment ha sigut contribuït al OpenHW Group. Es tracta d'una implementació més complexa que l'esmentada anteriorment i implementa una major quantitat d'extensions en comparació a l'anterior.

Ibex

Llenguatge: SystemVerilog
La implementació Ibex, s'origina al projecte PULP sota el nom Zero-riscy, actualment la manté el grup lowRISC i es tracta d'una implementació que s'adhereix a dues possibles versions donada la seva parametritazció, RV32I[M]C/RV32E[M]C.

NOEL-V

Llenguatge: VHDL
Es tracta d'una implementació parametritzable desenvolupada per Cobham Gais-

ler. La versió més senzilla es tracta d'un RV32IM i la més complexa es tracta d'una versió superescalar RV64GC.

SCR1

Llenguatge: SystemVerilog
El core SCR1 és desenvolupat per Syntacore i es tracta d'un RV32I/E[MC] que es troba sota la llicencia SolderPad Hardware License v.2.0.

Taiga

Llenguatge: SystemVerilog
Taiga és una implementació desenvolupada pel Reconfigurable Computing Lab de la universitat canadenca Simon Fraser. És un RV32IMA i es troba desenvolupat sota una llicenia Apache 2.0

12.1 Selecció

La configuració preferida es tracta d'un mòdul base amb l'extensió M. De les possibilitats contemplades, NOEL-V sembla el més adient, però aquest depèn de la parametrització i altres projectes desenvolupats per Cobham Gaisler resultant en un dels codis més complexos sobrepassant l'abast d'aquest projecte. CVA6 també es descarta degut a la seva complexitat.

De les implementacions restants totes es tracten de models d'aparent similar complexitat. Finalment, s'utilitzarà CV32E40P sota la llicencia Solderpad Hardware License v. 0.51.

13 Implementació escollida

La implementació escollida és finalment el processador CV32E40P. En aquesta secció es tractarà la seva instal·lació i el seu funcionament

13.1 Instal·lació

En primer lloc cal instal·lar els riscv-gnu-tools[12] i els seus requeriments. Aquestes són necessàries per a testejar adequadament la CPU. Aquestes eines inclouen, entre altres, el compilador GCC per a RISC-V component necessari per poder traduir codi a llenguatge assemblador i testejar el funcionament del processador. El següent script descarrega i instal·la aquestes eines en un sistema basat en Linux que utilitzi apt-get.

```
#!/bin/bash
sudo apt-get install autoconf automake autotools-dev curl python3
 ↪  libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison
 ↪  flex texinfo gperf libtool patchutils bc zlib1g-dev
 ↪  libexpat-dev
git clone https://github.com/riscv/riscv-gnu-toolchain
cd riscv-gnu-toolchain/
sudo mkdir /opt/riscv
./configure --prefix=/opt/riscv --with-arch=rv32imc
make
```

Addicionalment, cal descarregar el projecte. El següent script permet la descarrega i testeig del processador existent.

Cal denotar que l'execució de "git clone" descarrega, addicionalment, fitxers necessaris per als projectes germans a l'escollit -CV32E40P-. Un altre aspecte a remarcar és que el primer git clone no descarrega els fitxers font del processador sinó els necessaris per la simulació entre d'altres i és durant "make verilate" que s'executa internament una crida a "git clone https://github.com/openhwgroup/cv32e40p" per descarregar els fitxers font del processador sempre que aquests no existeixin durant la seva execució.

Per acabar, tot i que els fitxers per testejar dit processador es descarreguen sota el directori "core-v-verif-master/cv32e40p/sim/core/" un cop executat "make verilate" els fitxers font es descarreguen al directori "core-v-verif-master/core-v-cores/cv32e40p/rtl/".

```
#!/bin/bash
git clone https://github.com/openhwgroup/core-v-verif
cd core-v-verif-master/cv32e40p/sim/core/
make verilate #si aquest no existeix clona l'ultima versió del
 ↪  projecte a core-v-verif-master/core-v-cores/ i compila per
 ↪  verilator el projecte d'aquella localització
make sanity-veri-run #Executa un test simulant l'execució d'un
 ↪  programa "hello world" al processador
```

13.2 Funcionament

El processador CV32E40P es tracta d'un processador RISC-V que implementa els mòduls RV32IM[F]C, CV32E40P formava part, conjuntament amb CVA6 i Ibex, d'un projecte anomenat PULP[13]. Originalment anomenat RI5CY, CV32E40P és un processar RISC-V segmentat de quatre etapes en ordre d'ample 1 -només demana i executa màxim una instrucció per cicle-. Aquest processador utilitza un registres de mida 32 bits. La propera figura mostra el diagrama de blocs d'aquest processador.

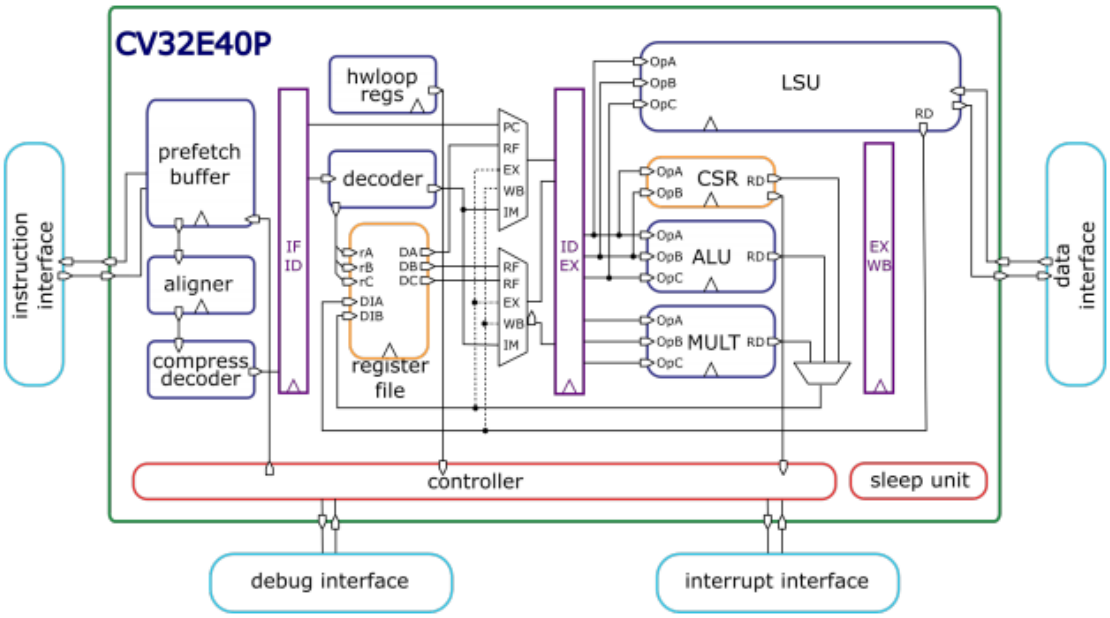

Figura 6: Diagrama de blocs de CV32E40P.

Font: Documentació de CV32E40P.

Les quatre etapes que conformen CV32E40P són: l'etapa IF o instruction fetch, l'etapa ID o instruction decode, l'etapa EX o etapa d'execució i l'etapa WB o Writeback.

13.2.1 Etapa IF o Instruction Fetch

Durant aquesta etapa es cerca la propera instrucció de la memòria. Aquesta proporciona la instrucció a executar sempre que la interfície de memòria estigui disponible. Addicionalment, descodifica les instruccions de tipus RVC (RISC-V Compressed). En aquesta etapa es troben els següents mòduls -unitat de disseny- Verilog: cv34e40p_aligner, cv34e40p_compressed_decoder i cv34e40p_prefetch_buffer. Aquest últim, cv34e40p_prefetch_buffer inclou els mòduls cv32e40p_prefetch_controller, cv34e40p_fifo i cv34e40p_obi_obi_interface.

13.2.2 Etapa ID o Instruction Decode

Durant aquesta etapa es realitza de descodificació de les instruccions i lectures dels registres necessaris. Addicionalment, el jumps es realitzen des d'aquesta etapa. El mòdul cv34e40p_id_stage inclou els mòduls cv32e40p_register_file, cv34e40p_decoder, cv34e40p_controller, cv34e40p_int_controller i cv32e40p_hwloop_regs. En aquest mòdul Verilog s'inclouen tots els mòduls necessaris per realitzar les esmentades tasques i el controlador del processador.

13.2.3 Etapa EX o etapa d'execució

L'etapa d'execució conté l'ALU la unitat de multiplicació i la unitat de divisió. Durant aquesta etapa, es realitza l'execució de les operacions Branch en cas que compleixin les seves condicions. En cas que executi instruccions de múltiples cicles de durada, retarda la sortida de les dades. Al mòdul cv34e40p_ex_stage s'inclouen els mòduls: cv32e40p_alu, cv32e40p_alu_div, cv32e40p_mult i cv32e40p_apu_disp.

13.2.4 Etapa WB o Writeback

L'última etapa és l'etapa de Writeback. Durant aquesta s'actualitzen els valors dels registres situats al mòdul cv32e40p_register_file de l'etapa de descodificació en cas que l'operació prèviament realitzada guardi el seu resultat a un registre.

14 Modificació de la lògica de control

En aquest apartat es tracten les modificacions realitzades a la lògica de control. Amb aquest objectiu, es tractarà la modificació realitzada a la versió del processador que utilitza la tècnica de duplicació en comptes de ”residue checking” per tal de simplificar el procés.

La primera modificació realitzada es tracta de la creació d'una nova instància del mòdul cv32e40p_ex_stage dins de cv32e40p_core. Aquesta nova instància que s'ha anomenat cv32e40p_ex_stage_clon. Les entrades d'aquesta instància es comparteixen amb les entrades de la versió original -a excepció de la introducció d'un sistema per generar errades a les entrades- i a les sortides han rebut un canvi de nom -afegint-li ”_clon” al senyal- per tal de poder comparar els senyals.

Com es pot observar a l'annexe 2, s'ha creat un nou senyal que detecta errors a la sortida de les unitats anomenada ”alu_er”. Aquesta ho realitza mitjançant una comparació dels valors de la sortida de la unitat original i la nova unitat clon.

Per tal d'introduir errors s'ha afegit el següent codi al sistema. Aquest, cada 256, cicles introdueix una errada al sistema modificant els valors d'entrada de la unitat aritmeticològica.

```
always_ff @(posedge clk) begin
  error_generate_counter = error_generate_counter + 8'h01;
end

assign alu_operand_a_ex  = error_generate_counter == 8'hFF ?
                (~alu_operand_a_ex_id)  : (alu_operand_a_ex_id);
assign alu_operand_b_ex  = error_generate_counter == 8'hFF ?
                (~alu_operand_b_ex_id)  : (alu_operand_b_ex_id);
assign alu_operand_c_ex  = error_generate_counter == 8'hFF ?
                (~alu_operand_c_ex_id)  : (alu_operand_c_ex_id);
assign mult_operand_a_ex = error_generate_counter == 8'hFF ?
                (~mult_operand_a_ex_id) : (mult_operand_a_ex_id);
assign mult_operand_b_ex = error_generate_counter == 8'hFF ?
                (~mult_operand_b_ex_id) : (mult_operand_b_ex_id);
assign mult_operand_c_ex = error_generate_counter == 8'hFF ?
                (~mult_operand_c_ex_id) : (mult_operand_c_ex_id);
```

Addicionalment, s'han realitzat modificacions al mòdul de control cv32e40p_controller. El següent fragment de codi s'utilitza per a detindre l'actualització de la pipeline entre les etapes de descodificació i execució. També modifica el PC de l'etapa Instruction Fetch per tal que es correspongui amb el correcte.

```
if (alu_error_i & (alu_error_ctr_cs != 4'b111))
  begin
    is_decoding_o     = 1'b0;
    pc_mux_o      = PC_ALU_ERROR;
    pc_set_o      = 1'b1;
    alu_error_ctr_ns = alu_error_ctr_cs + 4'b001;
  end
```

15 Mancances de la tècnica de residue checking

En aquest apartat es tracten diverses mancances i traves de la tècnica residue checking. Aquestes inclouen la representació de nombres negatius, la classificació dels valors -operacions de comparació-, overflow de les operacions i un mètode de divisió.

15.1 Overflow

Quan succeix un overflow o underflow el sistema de detecció d'errors utilitzant aritmètica residual realitza un fals positiu. L'operació que genera l'overflow perd bits representatius i, per tant, el sistema erra detectant un error. Un exemple és utilitzant 4 bits i mòdul 7, l'operació 8+8. Aquesta resulta en 16, però donat que només s'utilitzen 4 bits el valor resultant és 0. La unitat que empra aritmètica residual, en canvi, calcula correctament l'operació donant un resultat de 2. Aquesta discrepància activaria un fals positiu del sistema de detecció.

A l'article "Overflow Detection in Residue Number System, Moduli Set {2n-1,2n,2n+1}[14], s'hi expressen diversos mètodes per detectar overflows. Així i tot, aquests mètodes afegeixen complexitat al sistema. El mètode més senzill i eficient en termes de l'impacte sobre la freqüència és modificar la unitat aritmeticològica per a que detecti overflows afegint-li un bit per marcar quan es realitza una d'aquestes situacions. Per exemple, utilitzant el carry a una suma o una porta "OR" amb els bits que es perden durant una operació shift.

Ambdues solucions generen problemes pel sistema de detecció. En cas de detectar un overflow, la solució és desactivar el sistema de detecció d'errades per aquella instrucció. El raonament darrere aquesta decisió és el fet que cal una segona unitat de mida completa per testejar aquests casos invalidant l'objectiu de reduir el cost energètic del sistema. Això implica que en certes instruccions el sistema d'errades es troba desactivat i poden succeir errades no detectades.

15.2 Classificació

La segona dificultat d'utilitzar aritmètica residual és la comparació de nombres. En aquest cas donada la naturalesa d'aquesta on, per exemple, $8 \mod 7 \equiv 1 \mod 7$, la comparació i classificació d'aquests no es pot realitzar sense aplicar modificacions, ja que aquestes resultarien incorrectament.

L'article[15] discuteix l'existència de tres mètodes per comparar nombres en aritmètica residual. Els problemes d'aquests són elevats temps de càlcul o la seva falta de fiabilitat. L'article també proposa un nou mètode basat en PIPIC on utilitza informació addicional per a classificar els nombres en aritmètica residual.

Els majors inconvenients d'utilitzar aquesta tècnica són la lenta velocitat de càlcul requerint aquests realitzar transformacions sobre els operands allargant instruccions freqüents que requereixen un únic cicle a múltiples cicles. A més a més, requereixen la implementació d'un sistema de comparació de nombres completament diferent del ja implementat que genera un major consum energètic al circuit de classificació numèrica major que a un sistema on es duplica l'ALU original.

15.3 Divisió

El problema de les operacions de divisió es troba en el fet que no es tracten d'operacions lògicament equivalents a aritmètica modular. Per tant, el lema definit originalment

$$(a \ \mathbf{op} \ b) \ mod \ N \equiv (a \ mod \ N) \ \mathbf{op} \ (b \ mod \ N)$$

no aplica a aquestes. Això és fàcilment demostrable mitjançant un exemple. Si utilitzem la divisió de 26 i 2 realitzada en mòdul 7 obtenim que $\frac{26}{2} \mod 7 \equiv \frac{26 \mod 7}{2 \mod 7}$ això implicaria que $13 \mod 7 \equiv 5 \mod 7$ o el que és el mateix que $6 \equiv 5$ aquesta és una afirmació clarament falsa que demostra per contraexemple que el lema original no era completament correcte i cal acotar-lo a operacions de suma, resta i multiplicació.

La solució més senzilla d'implementar al disseny utilitzat és utilitzar un senyal que arribi al mòdul utilitzant i mantenir el disseny de la unitat de divisió original.

16 Anàlisis de freqüència

L'anàlisi de l'impacte de les modificacions realitzades s'ha realitzat mitjançant Vivado, un programari de Xilinx per tal de sintetitzar per a FPGAs dissenys realitzats amb llenguatges descriptors de hardware. Quartus és una alternativa a aquest programari, però aquest no té suport[16] per l'estàndard SystemVerilog (IEEE Standard 1800-2017) que es requereix pel codi del processador.

Aquests programes permeten calcular estimacions dels retards i el consum de potència energètica del disseny un cop implementat a una FPGA. Aquestes són estimacions realitzades mitjançant software i poden generar un resultat relativament precís dins d'un marge d'error. Donada la similitud dels dissenys a comparar, aquest valor es prendran com a referència per analitzar aquests impactes sense tindre en compte aquest marge.

Les propera figura mostra la sortida amb els camins crítics resultants d'executar l'analitzador de temps de Vivado sobre el processador original. Aquesta surt com a fallida -en vermell- donat que els requeriments amb els quals compara el resultat són més restrictius que aquest.

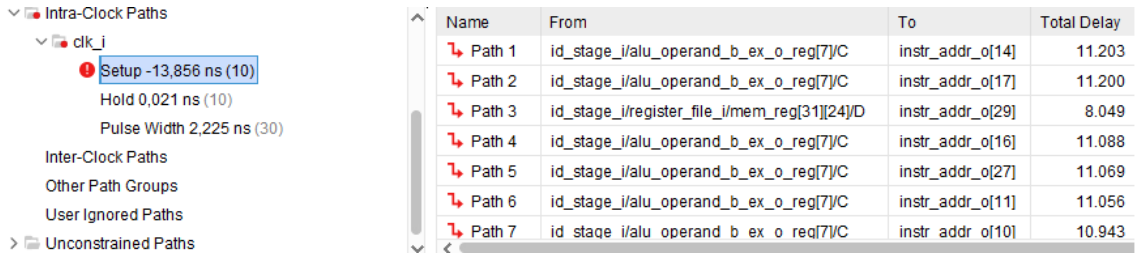

Figura 7: Retards dels camins crítics

Elaboració pròpia mitjançant Vivado

A continuació es mostra una taula amb els orígens, destins i retards dels camins crítics obtinguts de les diverses versions dels processadors. Cal denotar que la versió que empra la tècnica residue checking es tracta d'una versió ideal on únicament s'ha reduït la mida dels seus senyals i registres. Aquesta versió no ha sigut modificada per tindre en compte les mancances i errades tractades prèviament a la secció 15.

Versió	Origen	Destí	Retard
Base	Id_stage: alu_operand_b_ex_o	Core: instr_addr_o	11.203ns
Duplicació	Core: error_generate_counter	Core: instr_addr_o	13.238ns
Residue checking	Core: error_generate_counter	Core: instr_addr_o	16.989ns

Taula 6: Camins crítics de les diferents versions

Elaboració pròpia

Com es pot observar l'increment d'aplicar la tècnica de duplicació es troba en 2.035ns o un 18%. Per altra banda, aplicar la tècnica de residue checking implica un increment de 5.786ns o un 51.6% respecte al disseny original. Aquests resultats mostren com l'impacte sobre la freqüència de treball d'utilitzar residue checking és considerablement més elevat respecte a utilitzar una duplicació de la unitat aritmeticològica. Aquest és un dels majors desavantatges de la tècnica i pot tractar-se com a un argument en contra de la utilització d'aquesta a la funcionalitat analitzada.

17 Anàlisis del consum

Com exposat prèviament durant l'anàlisi de la freqüència, s'utilitza el programari Vivado de Xilinx per estimar els consums de potència energètica.

Les dues figures següents inclouen la configuració tèrmica de la FPGA i les condicions de la font d'alimentació d'aquesta durant l'estimació.

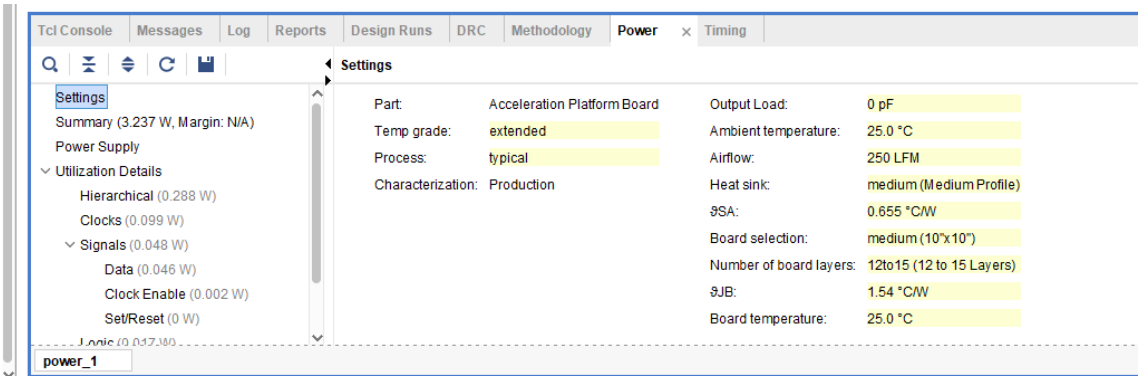

Figura 8: Configuració del paràmetres per l'estimació del consum

Elaboració pròpia mitjançant Vivado

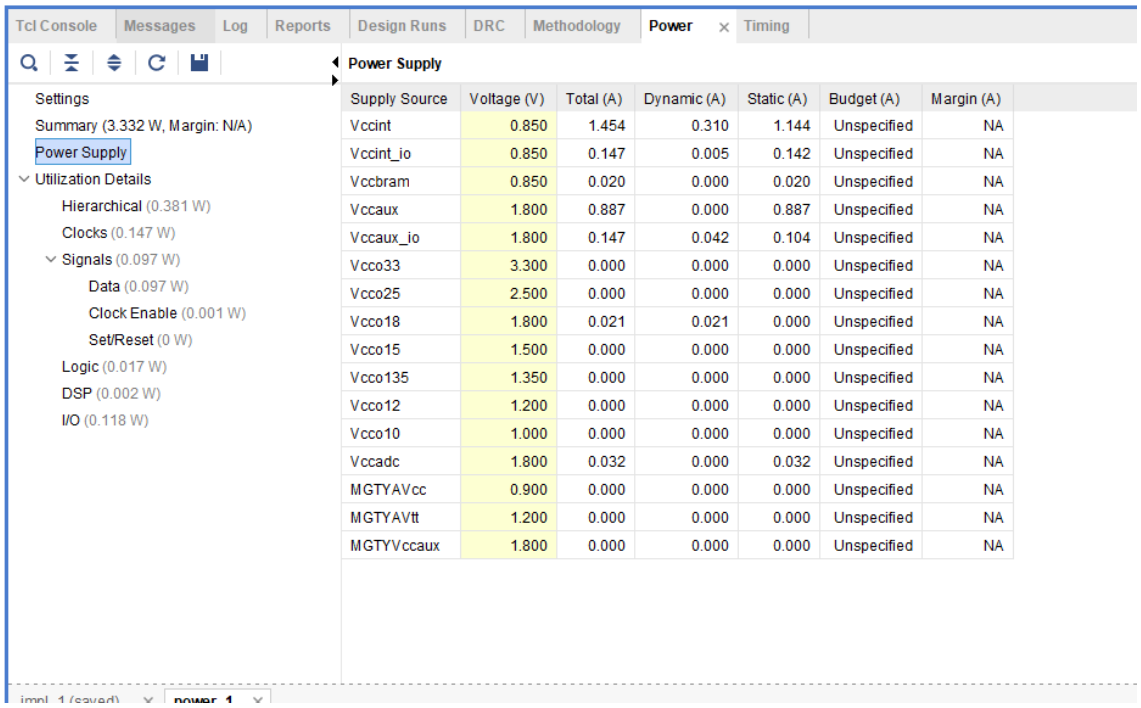

Figura 9: Voltatges a la font d'alimentació

Elaboració pròpia mitjançant Vivado

La propera figura conté el desglos del consum de potència energètica del processador base. El consum d'aquesta versió són aproximadament 3.237W.

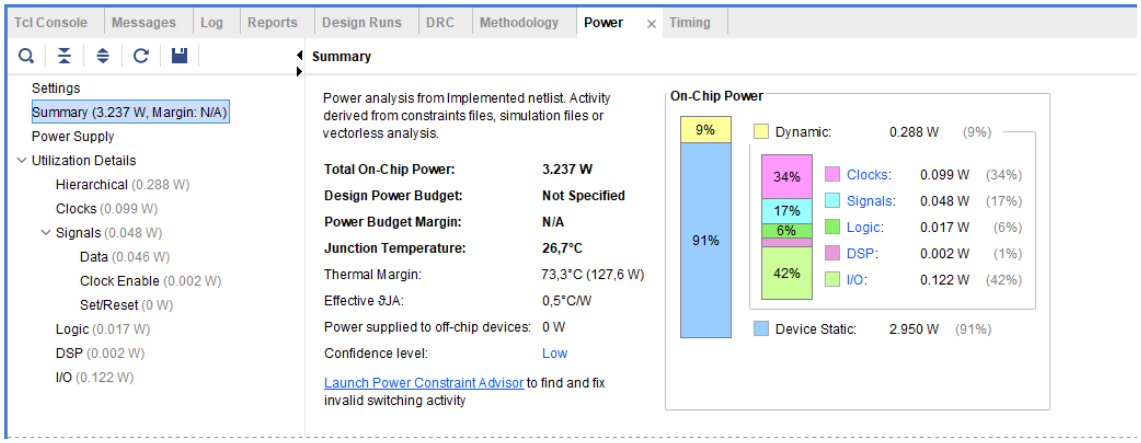

Figura 10: Base: Consum energètic desglossat

Elaboració pròpia mitjançant Vivado

La propera figura mostra el desglossament del consum de potència energètica de la versió amb sistema de comprovació d'errors mitjançant duplicació. El consum d'aquesta versió són aproximadament 3.332W.

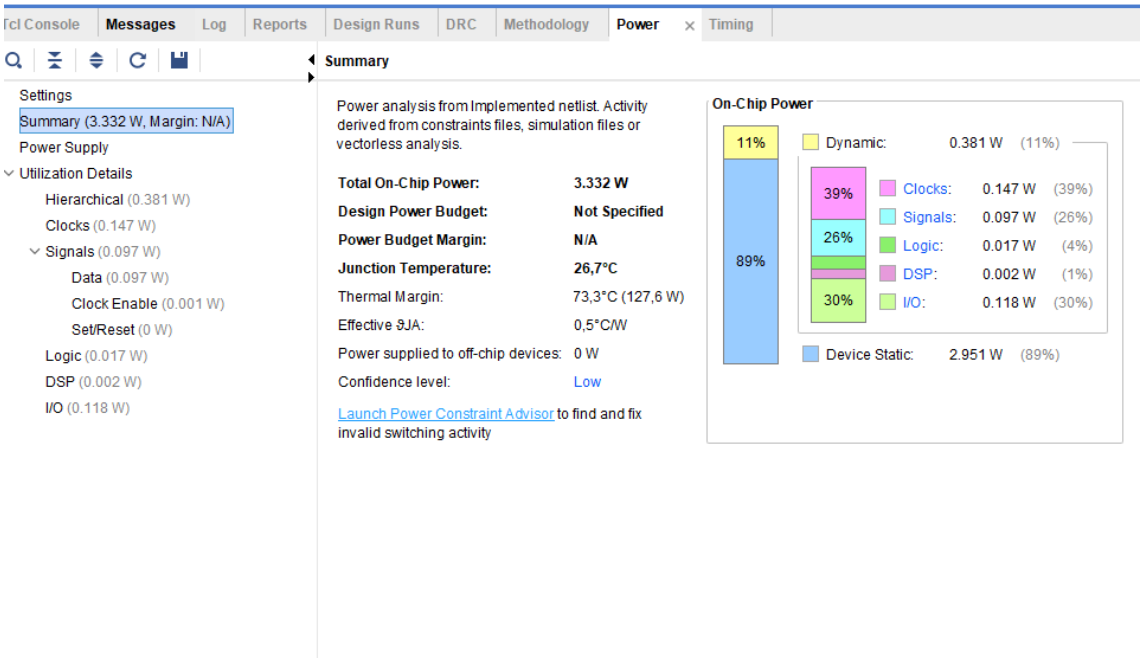

Figura 11: Duplicació: Consum energètic desglossat

Elaboració pròpia mitjançant Vivado

Per finalitzar, la propera figura conté el desglos del consum de potència ener-

gètica del processador amb un sistema de detecció d'errors basat en residue checking. Cal denotar que aquesta és una versió ideal d'aquest i és no funcional degut a les mancances d'aquest. El consum d'aquesta versió són aproximadament 3.283W.

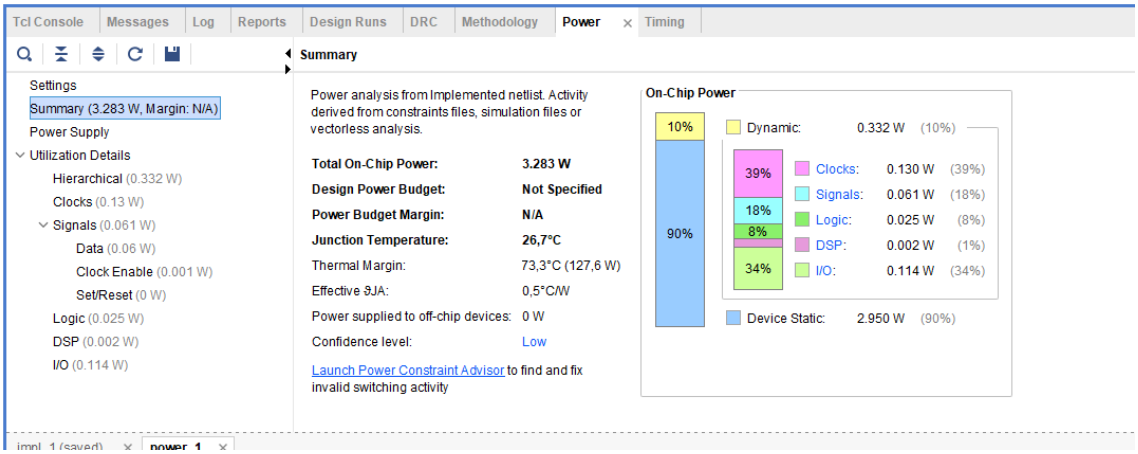

Figura 12: Residue checking: Consum energètic desglossat

Elaboració pròpia mitjançant Vivado

Utilitzant el processador base com a referència, la versió amb residue checking augmenta el consum un 1.4%. Per altra banda, la versió de duplicació augmenta el consum un 2.9%.

Tot i que aquests resultats indiquen que els objectius inicials de reduir el consum entre les diferents versions amb sistema de detecció d'error es pot donar per complert, les unitats aritmeticològiques dins un processador RISC-V -com és CV32E40P- representen un percentatge del consum gairebé negligible i la viabilitat de la incorporació de la tècnica "residue checking" no depèn únicament d'aquest sinó d'altres factors com les seves limitacions.

18 Conclusions

L'objectiu establert per aquest projecte era la comprovació d'errors dins de les unitats aritmeticològiques de les CPUs. De les diverses tècniques que poden aplicar-se, la tècnica que ha sigut avaluada durant aquest projecte és la utilització de les propietats dels nombres en aritmètica residual per comprovar la correcció dels resultats obtinguts. Una alternativa a utilitzar aquest mètode és la duplicació d'aquestes unitats.

Amb aquest objectiu s'ha realitzat la modificació un processador RISC-V per tal d'afegir un sistema de detecció d'errades i re-execució d'instruccions en aquest.

Addicionalment, s'han tractat les mancances del sistema de comprovació d'errades mitjançant aritmètica residual i s'ha analitzat l'impacte d'aplicar aquests sistemes sobre el processador original.

Amb la informació prèviament exposada en aquest document es pot concloure que la implementació d'un sistema de detecció d'errades mitjançant la tècnica "residue checking" no és una opció viable pel processador de propòsit general escollit.

Pel que fa a la fiabilitat d'aquest sistema, aquesta implementació implica per certs grups d'instruccions no és possible comprovar l'existència d'errades, minimitzant l'aplicabilitat d'aquest sistema i permetent l'execució d'instruccions amb resultats erronis no detectats.

Respecte a la freqüència de treball del processador, afegeix un retard al camí crític del processador reduint la freqüència màxima de treball d'aquest i, per tant, l'eficiència del processador.

Quant al consum energètic, l'increment energètic d'aplicar aquesta tècnica, en condicions ideals, és menor respecte a la duplicació de les unitats aritmeticològiques. Contraposant a la reducció de potència energètica obtinguda quan es compara amb la versió de duplicació, els retards que afegeix redueixen la freqüència de treball, contraresten els efectes al disminuir la productivitat del processador (instruccions per segon).

En conclusió, tot i que la tècnica a avaluar proporciona un sistema de detecció d'errades amb menor consum de potència energètica que la duplicació, la quantitat de consum energètic total per una mateixa tasca no es veu disminuïda donada la reducció de la freqüència de treball i de la productivitat del processador causats per la implementació d'aquest. Addicionalment, aquesta tècnica és poc fiable, ja que diverses instruccions generen dificultats i impossibilitats a l'hora de comprovar les seves errades.

En definitiva, un cop finalitzat l'estudi, "residue checking" no és una tècnica adient pel supòsit plantejat donat que no disminueix el consum energètic notablement respecte a la duplicació de les unitats aritmeticològiques i es tracta d'un sistema inequívocament menys fiable.

19 Annexe 1: C++ per testejar el càlcul del mòdul

```
#include <stdlib.h>
#include<iostream>
#include "Vcv32e40p_mod.h"
#include "verilated.h"
#include <verilated_vcd_c.h>
using namespace std;


int main(int argc, char **argv) {
Verilated::commandArgs(argc, argv);


// Create an instance of the module
Vcv32e40p_mod *tb =new Vcv32e40p_mod;


int x = -131071000, k = 0;
do{

   tb->Op_I = x;
   tb->eval();
   //cout << x << "   "  << tb->NegMod_19 << endl;
   if (tb->NegMod_17 != k) {
     cout << "error 17" << endl;
     cout << "modulo 2^17-1 de " << x << " = " << tb->NegMod_17
          << " pos " << tb->Mod_17 << " k " << k << endl;
     exit(EXIT_SUCCESS);
   }
   cout << "modulo 2^17-1 de " << x << " = " << tb->NegMod_17
        << " k " << k << endl;
   k = ++k % 131071;
   ++x;

} while (x < 0);


do{

   tb->Op_I = x;
   tb->eval();

   if (tb->Mod_17 != k) {
     cout << "error 17" << endl;
```

```
      cout << "modulo 2^17-1 de " << x << " = " << tb->Mod_17
           << " " << tb->NegMod_17 << " k " << k << endl;
      exit(EXIT_SUCCESS);
    }
    cout << "modulo 2^17-1 de " << x << " = " << tb->Mod_17
         << " k " << k << endl;
    k = ++k % 131071;
    ++x;

} while (x < 131071000);

exit(EXIT_SUCCESS);
}
```

20 Annexe 2: Detecció d'errades

```
logic alu_er;
assign alu_er = (mult_multicycle != clon_mult_multicycle)
          | (fflags_we != clon_fflags_we)
          | (apu_read_dep != clon_apu_read_dep)
          | (apu_write_dep != clon_apu_write_dep)
          | (perf_apu_type != clon_perf_apu_type)
          | (perf_apu_cont != clon_perf_apu_cont)
          | (perf_apu_wb != clon_perf_apu_wb)
          | (apu_ready_wb != clon_apu_ready_wb)
          | (apu_busy != clon_apu_busy)
          | (apu_op_oe != clon_apu_op_oe)
          | (apu_operands_oe !=clon_apu_operands_oe)
          | (apu_req_oe != clon_apu_req_oe)
          | (regfile_waddr_fw_wb_o != clon_regfile_waddr_fw_wb_o)
          | (regfile_we_wb != clon_regfile_we_wb)
          | (regfile_wdata != clon_regfile_wdata)
          | (regfile_alu_waddr_fw != clon_regfile_alu_waddr_fw)
          | (regfile_alu_we_fw != clon_regfile_alu_we_fw)
          | (regfile_alu_wdata_fw != clon_regfile_alu_wdata_fw)
          | (jump_target_ex != clon_jump_target_ex)
          | (branch_decision != clon_branch_decision)
          | (ex_valid != clon_ex_valid)
          | (ex_ready != clon_ex_ready);
```

21 Bibliografia

Referències

[1] How Space Weather Can Influence Elections on Earth. Vice.com [en línia], [Consulta: 28 de febrer de 2021] Disponible a: <https://www.vice.com/en/article/9agbxd/space-weather-cosmic-rays-voting-aaas>.

[2] RISC-V, RISC-V International. [en línia], Disponible a: <https://riscv.org/>.

[3] RISC-V International. Tuesday @ 1100 NVIDIA RISC V Evaluation Story Joe Xie, NVIDIA [en línia]. 2016. [Consulta: 28 de febrer de 2021] Disponible a: <https://www.youtube.com/watch?v=gg1lISJfJI0>

[4] Western Digital Corporation. [en línia]. [Consulta: 20 de març de 2021] Disponible a: <https://www.westerndigital.com/company/innovations/risc-v>

[5] David Patterson, Andrew Waterman. Guía Práctica de RISC-V: El Atlas de una Arquitectura Abierta. Strawberry Canyon, 2018. ISBN: 978-0-9992491-2-3. Disponible a: <http://riscvbook.com/spanish/>.

[6] Szabo, Nicholas S., i Richard I. Tanaka. Residue arithmetic and its applications to computer technology. McGraw-Hill, 1967. ISBN 9780070626591

[7] Sueldos para Hardware Engineer en Área Barcelona. Glassdoor [en línia]. [Consulta: 15 de març 2021] Disponible a: <https://www.glassdoor.es/Sueldos/barcelona-hardware-engineer-sueldo-SRCH_IL.0,9_IM1015_KO10,27.htm?clickSource=searchBtn>

[8] Sueldos para It Manager en Área Barcelona. Glassdoor [en línia]. [Consulta: 15 de març 2021] Disponible a: <https://www.glassdoor.es/Sueldos/barcelona-it-manager-sueldo-SRCH_IL.0,9_IM1015_KO10,20.htm?clickSource=searchBtn>

[9] Dunning-Kruger Effect. A: Wikipedia [En línia]. Wikimedia Foundation, 2018. [Consulta: 15 de març 2021]. Disponible a: <https://en.wikipedia.org/wiki/Dunning-Kruger_effect>.

[10] Editors Andrew Waterman and Krste Asanovic. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213, RISC-V Foundation, Desembre 2019.

[11] Butler, J. and Tsutomu Sasao. “Fast Hardware Computation of x Mod z”. 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, p294-297, 2011.

[12] RISCV-GNU-Toolchain. A: GitHub [En línia]. [Consulta: 5 de maig de 2021]. Disponible a: <https://github.com/riscv/riscv-gnu-toolchain>.

[13] PULP platform. A: PULP platform [En línia]. PULP platform, 2021. [Consulta: 3 de maig 2021]. Disponible a: <https://pulp-platform.org/>.

[14] Babak Tavakoli, Mehdi Hosseinzadeh, Somayeh Jassbi. "Overflow Detection in Residue Number System, Moduli Set {2n-1,2n,2n+1}". Journal of Advances in Computer Engineering and Technology, 2016.

[15] V. A. Krasnobayev, A. S. Yanko, and S. A. Koshman. "A METHOD FOR ARITHMETIC COMPARISON OF DATA REPRESENTED IN A RESIDUE NUMBER SYSTEM". Cybernetics and Systems Analysis, Vol. 52, No. 1, Gener, 2016.

[16] Quartus, 2017 Intel Corporation [En línia]. [Consulta 15 de maig de 2021]. Disponible a <https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/hdl/vlog/vlog_list_sys_vlog.htm>

Índex de figures

Índex de taules