INTRODUCCIÓ D'UN MÒDUL D'ENCRIPTACIÓ EN L'ACCÉS A MEMÒRIA

Universitat Politècnica de Catalunya

Tutor: Ramon Canal Corretger

Autor: Carles Ureta Boza

Curs: 2021 - 2022

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

ÍNDEX

ÍNDEX DE FIGURES

RESUM

La protecció de les dades és de gran importància en el món actual, hi ha molta informació i de molt valor i és necessari tenir mètodes per protegir-la. Hi ha moltes maneres de fer-ho, entre elles la encriptació per hardware. Aquesta es basa en l'ús d'un dispositiu físic dedicat a xifrar i desxifrar dades anomenat encriptador. Un encriptador es caracteritza principalment per el algorisme d'encriptació que utilitza.

En aquest treball es porta terme el disseny i la implementació d'un mòdul d'encriptació que utilitza l'algorisme d'encriptació RC5. S'analitza varis avantatges que aquest algorisme proporciona davant d'altres algorismes més populars, el principal punt a favor és que permet encriptar blocs de només 32 bits (normalment els algorismes treballen amb blocs de 64 o 128 bits com a mínim), i es veurà com això simplifica molt la incorporació d'un mòdul d'encriptació extern en processadors senzills.

1 CONTEXT

Dins del grau d'Enginyeria Informàtica impartit a la Facultat d'Informàtica de Barcelona hi ha varies mencions, aquest treball de fi de grau pertany a la menció de enginyeria de computadors, concretament en l'àmbit de disseny de circuits integrats i sistemes basats en microprocessadors.

1.1 INTRODUCCIÓ

El principal problema o preocupació de la nova era de la informació és la seguretat de les grans quantitats de dades que s'emmagatzemen. Si la informació emmagatzemada no té cap tipus de protecció pot suposar un perill per la privacitat de l'usuari.

Aquest problema apareix amb l'aparició de les memòries RAM no volàtils, les quals poden mantenir la informació que s'hi ha gravat encara que es talli el flux de corrent. Amb aquests tipus de memòries és possible fer un anàlisis de les dades un cop extraviades.

Encara que existeix una amplia gamma de eines per portar a terme el xifratge de les dades, la gran majoria són de software, aquest tipus de protecció va protegit amb una contrasenya per tal d'accedir a les dades. Si la contrasenya és obtinguda per un tercer, la informació es veuria compromesa. A més a més el principal avantatge de la encriptació per hardware és que es disposa de hardware especialitzat i, a diferència de l'encriptació per software, no utilitza recursos del processador. Per això és tan interessant la introducció de hardware dedicat a la encriptació de les dades en el camí d'accés a memòria.

En aquest projecte es dissenyarà i s'implementarà un mòdul d'encriptació per tal d'encriptar tots els accessos a memòria, de tal manera que totes les dades estiguin xifrades dins de memòria. Això es portarà a terme en un processador RISC-V. Es realitzarà a través de programes d'especificació de hardware, que permeten definir hardware a través de codi. S'utilitzarà un processador bàsic ja especificat obtingut d'una font fiable d'internet. A aquest processador se li afegirà un mòdul extern que simularà la memòria principal i el mòdul d'encriptació i desencriptació en el camí entre el processador i el mòdul de memòria.

1.2 DEFINICIÓ DE CONCEPTES

1.2.1 RISC-V

Aquesta iniciativa és un set d'instruccions estàndard obert i lliure sota llicències *Open-Source*. Això permet a petits desenvolupadors construir hardware sense haver de pagar comissions, i la seguretat és punt d'inflexió important per l'evolució d'aquesta arquitectura.

RISC-V ja incorpora certes característiques per augmentar la seguretat, entre elles una arquitectura privilegiada basada en quatre modes d'execució i una unitat de *Physical Memory Protection* (PMP) que serveixen per a protegir d'accessos a memòria perillosos. En aquest informe s'estudiarà una forma diferent de protecció de la memòria que no és nativa en el RISC-V però que pot augmentar significativament la seguretat de la informació mitjançant la introducció de un mòdul de encriptació en el camí d'accés a memòria principal.

L'arquitectura de RISC-V compte amb certs aspectes relacionats amb la seguretat. En la mencionada arquitectura privilegiada, els 4 modes d'execució són mode usuari (U), mode supervisor (S), que es sol usar per mòduls de *Kernel* i *drivers* de components; mode màquina

(M), el qual es reserva per arrencar el *bootloader* i firmware generalment. I el mode *hipervisor* (H) que ha sigut introduït com un esborrany en l'última especificació de l'arquitectura privilegiada de RISC-V.

A més a més, RISC-V també inclou una unitat anomenada *Physical Memory Protection* (PMP) que comprova a cada lectura i escriptura de memòria de les adreces físiques usades per els programes executats en modes usuari i supervisor sobre un conjunt de restriccions. Aquestes restriccions es poden configurar en el mode màquina.

1.2.2 Algorismes d'encriptació

L'algorisme d'encriptació és el mètode usat per transformar text en dades encriptades. Aquests algorismes solen utilitzar una clau, que sol ser un nombre aleatori, per encriptar les dades. Aquesta clau s'utilitza després per desencriptar les dades. L'avantatge de utilitzar una clau per l'encriptació és que el algorisme en sí és públic, per molt que tercers coneguin el algorisme d'encriptació no seran capaços de desencriptar les dades sense la clau, la qual pot ser qualsevol nombre dins un rang de valors, depenent de l'algorisme. En aquest projecte s'estudien diferents algorismes d'encriptació i es decideix quin és el que s'adapta millor per el projecte en qüestió.

1.2.3 Hardware Description Language

En enginyeria de computadors, els llenguatges de descripció de hardware (en anglès *Hardware Description Language*, HDL) són llenguatges que s'utilitzen per descriure les estructures i el comportament de circuits electrònics i circuits digitals lògics. També permeten la especificació de les connexions entre diversos circuits. És una forma formal i precisa de definir hardware que permet simular i analitzar el seu comportament.

Hi ha varis llenguatges de descripció de hardware, en aquest treball el llenguatge de descripció de hardware que s'utilitzarà anirà lligat amb el processador base que es seleccioni, ja que és molt probable que s'hagi d'utilitzar el llenguatge amb el que s'ha definit el processador (depèn del programari que s'utilitzi, hi ha programaris que permeten definir diferents mòduls amb diferents HDLs).

1.3 ACTORS

En aquest projecte hi ha varis actors implicats:

1.3.1 Dissenyador del mòdul

És la persona a càrrec del disseny, recerca, documentació i implementació del mòdul d'encriptació. A més a més, és responsable de la gestió de projecte tant a nivell de metodologies escollides com de gestió del temps. Aquest actor treballa de manera coordinada amb el director del projecte.

1.3.2 Director

El seu paper és el de guiar i donar consells al dissenyador. La seva acció pot ser clau a l'hora de detectar possibles errors en el projecte o ajudar a la hora de prendre decisions en el disseny. En particular Ramon Canal, professor associat del Departament de Arquitectura de Computadors actua com a director del projecte.

1.3.3 Beneficiaris

Aquest projecte no té un client directe. Si més no el disseny d'un nou mòdul d'encriptació en l'accés a memòria pot servir com a base per a futures implementacions i recerques en l'entorn RISC-V, com per exemple la implementació d'un mòdul d'encriptació post-quantum.

2 ESTAT DE L'ART

Actualment ja s'han portat a terme projectes que es treballen en la seguretat i protecció de dades en l'accés a memòria en l'entorn RISC-V. Algunes iniciatives es concentren en aprofitar al màxim els mecanismes de arquitectura privilegiada i la *Physical Memory Protection* que ja estan especificats per RISC-V i resulten en un entorn d'execució més robust i segur.

En la Universitat de Nàpols Federico II s'ha estudiat el disseny i la implementació d'un mòdul de encriptació en l'accés a memòria, presentat per Alessandro Cilardo [1]. En aquest cas utilitzen l'algoritme *ChaCha* com a xifrat de flux. El xifrat de flux consisteix en xifrar bit per bit els text original. Els accessos a memòria son de blocs de 32 bits. Hi ha també xifratge per blocs. Hi ha diferents algorismes que permeten diferents mides de blocs.

Intel també ha treballat en la introducció d'un mòdul de encriptació de memòria, en aquest cas en un entorn x86, introduint TME (*Total Memory Encryption*) i MKTME (*Multi-key Total Memory Encryption*) [2]. TME i MKTME són extensions del set d'instruccions x86 que proporciona suport complet de xifratge de memòria física per a DRAM i NVRAM. TME és l'extensió base que afegeix les capacitats per a utilitzar una única clau de curta durada. MKTME és una millora addicional de TME que proporciona suport per al xifratge de memòria a nivell de pàgines mitjançant el suport per a múltiples claus de xifratge. En aquest cas utilitzen un algoritme de xifratge més comú, l'AES.

Com Intel, AMD també ha proposat un extensió al set d'instruccions de x86 per el xifratge de memòria a nivell de pàgines, en aquest cas utilitzant només una sola clau efímera [3]. L'extensió presentada per AMD rep el nom de SME (*Secure Memory Encryption*).

Tot i això, la avantatge de utilitzar RISC-V és que és una arquitectura *open-source*, no és d'ús comercial i per tant es pot treballar amb ella sense haver de pagar comissions. A més a més RISC-V té un set d'instruccions més senzill que poden ser executades en un cicle de rellotge. Això redueix el temps d'execució i fa que sigui una arquitectura més eficient.

3 ABAST

3.1 OBJECTIU

L'objectiu final d'aquest projecte és dissenyar i implementar un mòdul d'encriptació per l'accés a memòria en un entorn RISC-V. Aquest mòdul utilitzarà l'algoritme de encriptació AES. Ha de ser un mòdul robust i segur. Es busca que permeti encriptar i desencriptar en temps real tots els accessos a memòria del processador.

3.2 SUBOBJECTIUS

Selecció i especificació de l'entorn. El primer pas és seleccionar i especificar l'entorn que s'utilitzarà. Això es farà en la pròxima reunió programada entre el director i el dissenyador, on s'especificarà el programari que s'utilitzarà per l'especificació del hardware i el processador en sí.

Comprensió del funcionament del processador. Un cop el processador ha estat determinat. És necessari dedicar-hi temps en la comprensió del funcionament dels senyals d'aquest, per tal d'agilitzar la especificació del mòdul i la presa de decisions en el futur.

Comprensió del funcionament dels accessos a memòria. Com en el processador, també cal fer un estudi de com aquest es comunica amb la memòria cache i la memòria principal i quins senyals utilitza.

Comprensió de l'entorn de treball. En aquest cas ens referim a les eines que s'utilitzaran per portar a terme el projecte. En distingim 2 eines: El software que s'utilitzarà per la descripció del hardware i l'especificació del processador base amb el que es realitzarà el projecte.

Disseny del mòdul d'encriptació. Aquest subobjectiu es podria categoritzar com el més important, o el principal. Aquí és on realment es treballarà i on es dedicarà més temps. És la tasca principal del projecte.

Optimització del mòdul d'encriptació. Es contempla com a objectiu una possible optimització que pugui ser realitzada un cop dissenyat el mòdul, per tal de reduir el nombre de cicles per operació de encriptació en l'accés a memòria, si és possible. Aquest objectiu és opcional i s'estudiarà la seva necessitat i/o possibilitat més endavant.

Documentació. Finalment, el treball realitzat requereix estar documentat per tal de facilitar la seva comprensió un cop acabat, però també el seu seguiment durant la realització.

3.3 REQUERIMENTS

Els requeriments del mòdul que es dissenyarà són els següents:

1. Eficiència. El mòdul ha de mantenir els accessos a memòria el més ràpid possibles. S'ha d'evitar un alt increment del temps d'execució. Encara que aquest no és un requeriment funcional, és interessant mantenir-lo com a primordial per tal que el disseny proposat sigui considerat com a opció pràctica.
2. Independència. El mòdul no pot afectar al funcionament del processador o la memòria, tot el procés d'encriptació es porta a terme de manera independent dins el mòdul.
3. Consistència. Tot i ser independent, els accessos a memòria ara passaran primer per el mòdul d'encriptació abans d'arribar al controlador de memòria. Per tant, el mòdul ha de mantenir la consistència de memòria en els accessos.

4 ANÀLISI D'ALTERNATIVES

En aquest projecte hi ha dos punts clau on una decisió ha de ser presa i que afecten al resultat final del treball, un és el processador amb el que es treballarà i l'altre és l'algorisme d'encriptació que utilitzarà el mòdul d'encriptació. En aquesta secció s'expliquen les diferents alternatives que existeixen per cada cas i la justificació de la decisió final.

4.1 SELECCIÓ DEL PROCESSADOR

Per a la selecció del processador s'han tingut en compte les següents característiques:

- **Llenguatge:** El llenguatge de descripció de hardware amb el que s'ha descrit el processador. Hi ha varis però els més coneguts són Verilog, SystemVerilog i VHDL. Verilog és un llenguatge *weakly typed*. Tots els tipus de dades estan predefinits en Verilog i tots tenen una representació a nivell de bits. La sintaxis és semblant a C. SystemVerilog inclou un set d'extensions sobre Verilog per permetre dissenyar i verificar dissenys més grans i complexos [4].

  VHDL en canvi és un llenguatge *strongly typed* i més redundant i detallista, en conseqüència VHDL es més auto descriptiu. La sintaxis no és emblant a C i es requereix de codi extra per passar d'un tipus de dades a un altre.

  El fet que VHDL sigui auto descriptiu es considera una gran avantatge a la hora de utilitzar codi de tercers, que és el que es farà en aquest projecte quan es treballi amb el processador. A més a més, que el dissenyador tingui experiència amb el llenguatge fa que VHDL sigui la opció més buscada a la hora d'escollir un processador.

- **Documentació:** La claredat i qualitat de la documentació del processador és un punt important ja que és la eina que ens ajudarà a entendre el seu comportament, si el processador no té una bona documentació pot complicar molt el procés de comprensió de l'entorn i codi. En el cas d'aquest projecte s'observa sobretot la secció de memòria, que és en la que es treballarà.

- **Execució:** Per cada processador es prova que faci el anàlisis i síntesis correctament i que es pugui simular, es té en compte si és senzill arribar a simular amb el processador o si requereix configuracions complexes.

- **Claredat del codi:** Observem si el codi és comprensible, si conté comentaris que el complementen correctament i és fàcil de entendre els noms de les variables. Aquesta característica té bastanta relació amb la documentació.

A part d'aquestes característiques principals, també s'han fet observacions interessants de cada processador si es considerava important: com la seva finalitat, si és Von Neumann o Harvard o la complexitat del processador en sí.

Els processadors que s'han tingut en compte procedeixen tots de la base de dades de *RISC-V Exchange*, una plataforma on es poden obtenir de manera gratuïta processadors fets amb RISC-V i que compleixen la normativa RISC-V. A partir d'aquí s'han buscat diferents processadors, alguns amb SystemVerilog i Verilog, i la majoria en VHDL. Amb els processadors més interessants, per prendre una decisió s'han observat les diferents característiques que oferia cada un i es va crear la taula 1.

Nom	Llenguatge	Claredat codi	Execució	Documentació	Observacions
RPU	VHDL	- Senzill, però té alguna senyal confusa. (Fals byteEnable).	- Analisis i síntesis correcte. - Compila.	- No té una documentació oficial.	- Senzill. - Fet com a hobby. - Mateixa memòria instruccions i dades.
Maestro	VHDL	- Codi datamem molt clar i senzill.	- Analisis i síntesis. - No compila.	- No té documentació clara.	- Acadèmic. - Memòria senzilla implementada.
NEORV32	VHDL	- Codi datamem complex. - Conté comentaris aclaratius.	- Conté setups predefinits per Quartus. - Problemes amb les llibreries al simular.	- Molt bona documentació.	- Processador complex. - Memòria implementada.
REONV	VHDL	- Directoris complexes.		- Documentació confusa i complexa.	
Ibex	SystemVerilog				

Taula 1. Resum de característiques dels processadors observats. En verd el processador seleccionat. Font pròpia.

A partir del fitxer de *GitHub* de Colin Riley, es pot observar que el processador RPU és molt senzill, cosa que al principi semblava que hagués de facilitar les coses, però la documentació és també molt senzilla i pobre, cosa que complica molt la correcte comprensió del codi [5]. A més a més conté senyals confuses i el codi en general es veu brut. Es va contactar amb el creador del processador per *Twitter* per a que opinés sobre l'ús del seu processador per aquest projecte i efectivament va recomanar que s'utilitzés un altre processador més professional.

La següent opció és el processador Maestro, un processador acadèmic desenvolupat a la universitat de Rio Grande do Norte's Federal University a Brasil, presentat per Joao Chrisóstomo [6]. Aquest processador es manté a la mateixa línia que l'anterior en l'aspecte de complexitat i qualitat de la documentació, és un processador acadèmic, més ben acabat que l'anterior, però no suficientment.

El NEORV32 és un processador que proporciona moltes configuracions diferents per adaptar-se a diferents necessitats i incorpora una sèrie d'eines per tal de treballar més còmodament amb el processador [7]. És notòria la qualitat de la documentació, on es descriuen totes les senyals i constants importants de cada mòdul que incorpora, s'explica detalladament la configuració de l'espai d'adreces de memòria i el funcionament general del processador. A més a més també conté una documentació sobre la instal·lació de les eines proporcionades i els passos a seguir per a simular amb el processador. El codi és més complex que els processadors anteriors, però gràcies a la documentació, és molt més entenedor, és a dir, això no genera cap inconvenient.

Finalment, l'últim processador observat amb dedicació i descrit amb VHDL és el REONV, un processador que és una versió modificada de Leon3, el Leon3 compleix amb la arquitectura SPARC V8 i el REONV és una modificació que implementa RISC-V RV32I [8]. Aquest processador és utilitzat en supervisió de satèl·lits arreu del món (la versió que compleix amb SPARC V8). La versió modificada esta feta de tal forma que es pugui utilitzar la mateixa documentació que el processador Leon3, per tant la documentació és la *GRLIB IP Library*. És una documentació molt bona però molt més complexa i extensa que les anteriors.

També es va tenir en compte la possibilitat de treballar amb un processador descrit amb un altre llenguatge que no fos VHDL, es va analitzar el processador IBEX, un processador escrit en *SystemVerilog* [9]. La desavantatge de que el llenguatge de descripció de hardware no sigui dominat per el dissenyador es va fer òbvia mentre s'analitzava, i es va descartar la opció directament.

Finalment, el processador seleccionat és el NEORV32 com es pot veure en verd a la taula 1. El processador REONV també és una molt bona opció, però la complexitat i l'extensió de la seva documentació i dels seus fitxers ha fet que la balança s'acabés inclinant cap a l'altre costat.

A continuació es descriuen les característiques més importants del processador i que més afecten al desenvolupament del projecte.

4.1.1 Processador definitiu – NEORV32

El NEORV32 és un processador Harvard, el que significa que hi ha dos interfícies diferents per els accessos a dades i els accessos a instruccions. Les dos interfícies s'ajunten en un sol bus a través d'un interruptor de bus amb prioritat (els accessos de dades son prioritaris). Totes les direccions de memòria s'assignen en un únic espai d'adreces de 32 bits.

El processador ha estat dissenyat per tal de proporcionar màxima flexibilitat en les configuracions de memòria. El processador pot utilitzar les memòries internes per accedir a l'espai d'adreces d'instruccions o a l'espai d'adreces de dades. El processador també permet afegir de manera senzilla memòria externa. Es permet qualsevol de les combinacions possibles, com es pot observar a la figura 1. El que s'utilitzarà en aquest projecte serà la configuració C, on la memòria de dades és externa al processador, però la de instruccions és interna.

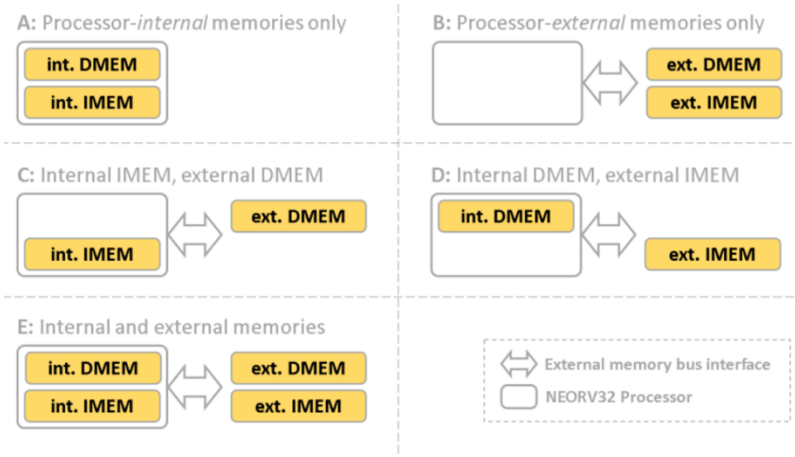

Figura 1. Configuració de les memòries del processador. Font: https://stnolting.github.io/neorv32/img/neorv32_memory_configurations.png

La connexió entre el processador i el mòdul de memòria externa es basa en el protocol *Wishbone*, per tant el mòdul que es crearà en aquest projecte haurà de complir amb aquest protocol per tal de comunicar-se correctament amb el processador.

4.2 ALGORISME D'ENCRIPTACIÓ

4.2.1 Advanced Encryption Standard

En quan a mètodes d'encriptació, hi ha varies opcions. El mètode més utilitzat en unitats d'encriptació de memòria en altres entorns és el AES (Advanced Encryption Standard). Aquest mètode va ser establert per l'Institut Nacional de Estàndards de Tecnologia (NIST) dels Estats Units el 2001 [10]. AES especifica un algoritme de encriptació aprovat per FIPS (*Federal Information Processing Standards*) que pot ser usat per protegir dades electròniques. És un encriptador de bloc simètric. L'encriptació converteix les dades en un format inintel·ligible, en anglès anomenat *ciphertext*. Quan el *ciphertext* és desencriptat es converteix en informació en la seva forma original i rep el nom en anglès de *plaintext*. L'algoritme de AES és capaç de utilitzar claus d'encriptació de 128, 192 i fins a 256 bits per encriptar i desencriptar dades de blocs de 128 bits.

4.2.2 Algorisme RC5

L'algorisme RC5 es caracteritza per la seva flexibilitat i simplicitat sense sacrificar seguretat. RC5 es un algoritme de xifratge per blocs dissenyat per Ronald Rivest el 1994. RC són les sigles en anglès de Cifratge de Rivest. Un dels algorismes candidats a ser el AES era el RC6, el qual esta basat en RC5. Aquest xifratge permet treballar amb blocs de dades de 32, 64 i 128 bits. Es pot decidir el nombre de rondes des de 0 fins a 255 i la mida de la clau en bytes des de 0 fins a 255 també.

RC6 és un successor de RC5 i és més famós degut a que va ser presentat com a candidat per AES. La diferencia entre RC5 i RC6 és que RC5 divideix el bloc de dades d'entrada en 2 sub-blocs i RC6 ho divideix en 4 sub-blocs. A més a més, RC6 afegeix una multiplicació per un enter en el procés d'encriptació. RC6 només treballa amb mides de bloc de 128 bits.

4.2.3 Algorisme SHA

SHA respon a les inicials *Secure Hash Algorithm* (Algorismes de Hash Segurs) en anglès. Aquests són una família de funcions de *Hash* criptogràfiques publicades per l'Institut Nacional de Estàndards de Tecnologia (NIST) dels Estats Units com a estàndard federal del procés d'informació. Una de les principals característiques d'aquest algorisme es la capacitat de no reflexivitat, donada una cadena de bits de sortida resulta impossible intentar trobar una cadena origen que retorni el mateix contingut. Una conseqüència d'aquesta propietat és que els algorismes *Hash* no es poden desencriptar, el que fa que no sigui la opció més òptima per el projecte en qüestió.

4.2.4 Algorisme DES

DES respon a les inicials *Data Encryption Standard* (Estàndard d'encriptació de Dades) en anglès. S'ha vist que aquest algoritme és vulnerable a atacs molt forts i el seu ús ha anat en decaiguda. Aquest algoritme treballa amb blocs de 64 bits i la mida de la clau és de 56 bits [11].

4.2.5 Decisió final – RC5

En un principi es va pensar utilitzar el algorisme d'encriptació AES ja que és un mètode molt estès i conegut per la seva robustesa. Una de les característiques d'aquest algorisme que ja s'ha mencionat anteriorment és que treballa amb blocs de 128 bits. Tenint en compte que el processador seleccionat és de 32 bits, els accessos a memòria no tenen la mida suficient per a ser encriptats individualment. Una de les possibles solucions a aquest problema és introduir una memòria cau entre la CPU i el mòdul d'encriptació, la qual treballés amb blocs de 128 bits, llavors

en els accessos a memòria principal des de la memòria cau es farien les encriptacions i desencriptacions.

El processador seleccionat té la opció de introduir una memòria cau, el problema és que aquesta només és per a la memòria d'instruccions. Degut a la falta de temps es va decidir evitar fer una memòria cau des de 0. Per tant les encriptacions s'han de fer en blocs de 32 bits, cosa que no es pot fer amb el algorisme de AES. És per això que es va decidir canviar de algorisme de encriptació a RC5.

Aquest és un dels pocs algoritmes que permet treballar amb 32 bits i evitar l'ús una memòria cau per tal de poder encriptar dades. Aquesta característica pot ser interessant per a un determinat grup d'aplicacions que requereixen d'un mòdul d'encriptació senzill, eficient i petit com podria ser en alguns sistemes encastats.

Actualment a opencores.org[1] hi ha diversos mòduls d'encriptació però cap utilitza el algorisme RC5 la gran majoria utilitzen AES. Podria ser interessant l'aportació d'un mòdul d'encriptació de blocs de 32 bits amb l'algorisme RC5 *open-source*.

[1] Lloc web que ofereix als usuaris accés obert per veure, descarregar, reutilitzar i compartir dissenys de components. OpenCores s'especialitza en paquets de fitxers estructurats que formen unitats autoconfinades, més coneguts com a "nuclis" de propietats intel·lectuals (IP), codificats en llenguatge de descripció de maquinari (HDL).

5 DESCRIPCIÓ DEL PROCESSADOR

5.1 ARQUITECTURA

El NEORV32 és un processador multicicle, per tant cada operació s'executa com una sèrie de micro-operacions. Aquest processador es divideix en dos etapes, que se lis assigna el nom de *front-end* (cerca de la instrucció) i *back-end* (execució de la instrucció). Amb aquesta arquitectura es pot començar a fer la cerca de la instrucció mentre el *back-end* processa la anterior instrucció.

En la primera etapa s'obtenen de memòria els 32 bits de la instrucció i s'introdueixen en una cua FIFO (*First In First Out*) anomenada *Instruction Prefetch Buffer*.

La segona etapa és responsable per la actual execució de la instrucció. Dins el *Execute Engine* es fa la descodificació de la instrucció.

Les dos etapes treballen en paral·lel, per tant el CPI (Cicles Per Instrucció) òptim és de 1. Tot i això, els CPI poden augmentar degut a accessos a memòria o executant operacions multicicle de la ALU (*Arithmetic Logic Unit*) com la divisió, entre d'altres. A la figura 2 es pot observar la organització del *pipeline* del processador.

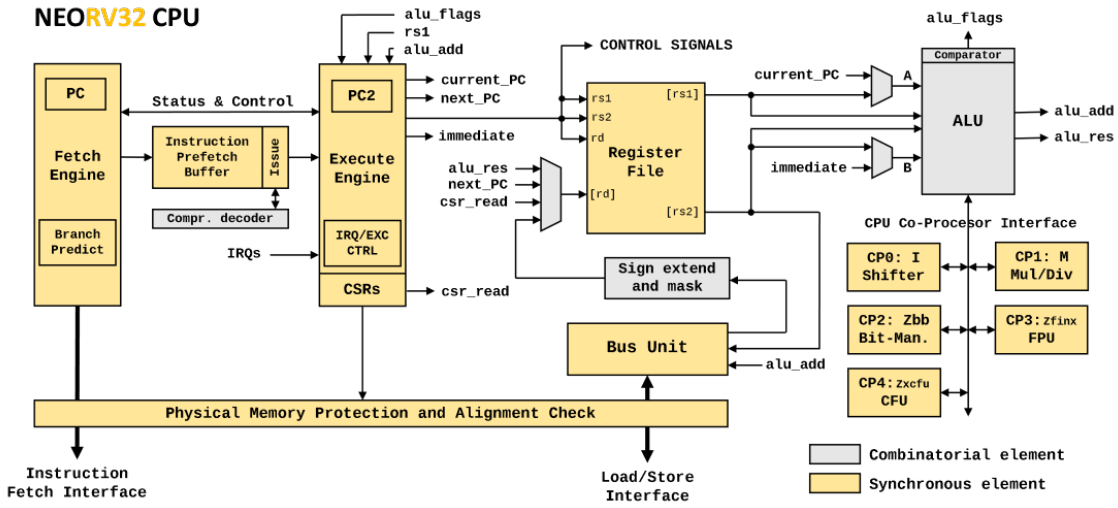

Figura 2. Arquitectura NEORV32. Font: https://stnolting.github.io/neorv32/img/neorv32_cpu.png

En un accés a memòria, després de la descodificació de la instrucció dins l'*Execute Engine,* es fa la lectura dels registres font del banc de registres. RS2 conté el contingut a escriure a memòria en cas que es tracti d'un accés d'escriptura. L'adreça d'accés s'obté a partir de la suma de RS1 i l'immediat dins la ALU. El resultat, *alu_add* i RS2 s'envien al mòdul *Bus Unit* que s'encarrega de fer l'accés a memòria. En cas que sigui una operació de lectura, el contingut de memòria que rep el mòdul *Bus Unit* s'envia al banc de registres i s'escriu al registre destí.

5.2 ACCÉS A MEMÒRIA

Com a màquina Harvard, el processador té dos interfícies diferents per l'accés a instruccions i a dades. Aquestes dues interfícies s'ajunten en un únic bus intern del processador mitjançant un commutador de bus de prioritat (els accessos de dades tenen prioritat més alta). Totes les ubicacions de memòria, inclosos els dispositius perifèrics, s'assignen a un únic espai d'adreces de 32 bits unificat.

La CPU pot accedir a tot l'espai d'adreces des de l'accés a instruccions i també des de l'accés a dades. Aquestes dues interfícies de CPU es multiplexen mitjançant un commutador de bus en un únic bus intern del processador. Totes les memòries internes del processador, els perifèrics i també la interfície de memòria externa estan connectades a aquest bus, com es pot veure a la figura 3. Per tant, ambdues interfícies de la CPU (recuperació d'instruccions i accés a dades) tenen accés al mateix espai d'adreces (idèntic).

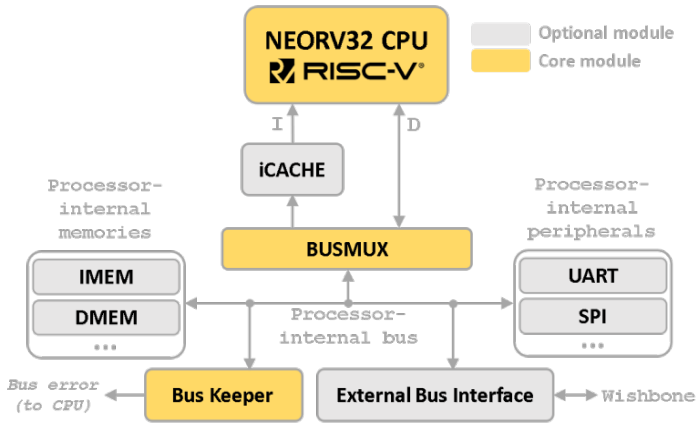

Figura 3. Arquitectura de l'accés a memòria. Font: https://stnolting.github.io/neorv32/img/neorv32_bus.png

El bus intern del processador pot aparèixer com un coll d'ampolla. Per tal de reduir els embussos en aquest bus (quan l'accés a instruccions i l'accés a dades coincideixen al bus), l'accés d'instruccions de la CPU està equipada amb un buffer (*Prefetch buffer* a la figura 2). Les instruccions es poden conservar a la memòria cau intermèdia mitjançant l'*i-cache*. A més, els accessos a les dades (lectures i escriptures) tenen una prioritat més alta que els accessos d'instruccions.

La interfície del bus extern (*External Bus Interface* a la figura 3) no està assignada a una regió específica d'espai d'adreces. En canvi, tots els accessos a memòria del processador que no s'orienten a un mòdul intern del processador es deleguen a la interfície de memòria externa. En resum, un accés de lectura/escriptura de la CPU es delega a la interfície del bus extern si:

1. No s'adreça a la memòria d'instruccions interna IMEM (si s'ha implementat)
2. i no s'adreça a la memòria de dades interna DMEM (si s'ha implementat)
3. i no s'adreça a la ROM interna del *bootloader* ni a cap dels dispositius d'entrada i sortida, independentment de si un o més d'aquests components estan implementats o no.

5.3 TEMPS D'EXECUCIÓ D'INSTRUCCIONS

A continuació, a la taula 2, podem observar els cicles d'execució de les diferents instruccions en el processador.

Classe	Instruccions	Cicles d'execució
ALU	addi slti sltiu xoriori andi add sub sltsltu xor or and lui auipc	2
Salts condicionals	beq bne blt bge bltu bgeu	*Taken* : 5 + (LM-1) *Not taken* : 3
Salts condicionals	jal jalr	5 + (LM-1)
Accés a memòria	lb lh lw lbu lhu sb sh sw	5 + (LM-1)

Taula 2. Cicles d'execució de les instruccions. LM = Latència de memòria. Font: https://stnolting.github.io/neorv32/#_instruction_timing.

6 DESCRIPCIÓ DEL MÒDUL D'ENCRIPTACIÓ

El mòdul que s'ha dissenyat en aquest projecte es connecta amb el processador a través d'una interfície de bus compatible amb *Wishbone B4*. Per tant a la hora de dissenyar el mòdul primer es van estudiar les senyals amb les que treballa el protocol *Wishbone*.

6.1 PROTOCOL WISHBONE

Wishbone és una arquitectura de bus de codi obert destinat a permetre que les parts d'un circuit integrat es comuniquin entre elles. L'objectiu és permetre la connexió de diferents nuclis entre si dins d'un xip. El *Wishbone* és utilitzat per molts dissenys del projecte *OpenCores* [12].

Aquest protocol consisteix de nodes màster i un nodes esclau. En aquest cas tenim un node màster que és el processador i un node esclau, el mòdul de memòria extern.

A la figura 4 es poden observar les senyals de cada node segons el informe oficial de *OpenCores* on es descriu l'arquitectura. Algunes de les senyals són opcionals i no totes són utilitzades en el processador seleccionat.

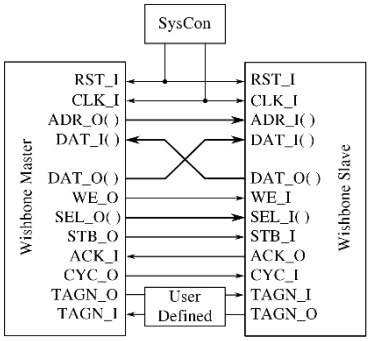

Figura 4. Diagrama senyals Wishbone. Font: https://upload.wikimedia.org/wikipedia/commons/thumb/7/71/Wishbone_Interface.svg/1200px-Wishbone_Interface.svg.png

D'aquestes senyals, les que proporciona la interfície del processador (node màster) són:

1. *wb_adr_o*: l'adreça de memòria a la que s'accedeix, és un vector sense signe de 32 bits.
2. *wb_dat_i*: la paraula que es llegeix de memòria en una lectura, és un vector sense signe de 32 bits.
3. *wb_dat_o*: la paraula que s'escriu a memòria en una escriptura, és un vector sense signe de 32 bits.
4. *wb_we_o*: bit que indica si es tracta d'una operació de lectura (0) o una d'escriptura (1).
5. *wb_sel_o*: indica a quins bytes de la paraula de 32 bits s'accedeix, és un vector sense signe de 4 bits, un per cada byte de la paraula. Si, per exemple, el bit 3 del vector està a 0, no es modificarà el contingut del byte de més pes de la paraula accedida de memòria.
6. *wb_stb_o*: aquesta senyal d'un bit està activada durant tota una transacció. Indica la validesa de les dades. El node esclau ha de respondre sempre a una senyal de *strobe*, i aquesta es negarà.
7. *wb_cyc_o*: quan s'activa, indica que hi ha un cicle de bus vàlid en curs. El senyal s'afirma durant la durada de tots els cicles de bus. Per exemple, durant un accés en bloc hi pot haver múltiples transferències de dades. El senyal s'activa durant la primera transferència de dades, i roman afirmada fins a la darrera transferència de dades. Aquest senyal és útil per a interfícies multiport (com ara memòries de dos ports). En el nostre cas té el mateix ús que el *strobe*.

8. *wb_lock_o*: quan s'activa indica que la transferència no es pot interrompre. Aquesta senyal no s'usarà en la nostre implementació degut a que totes les transaccions tenen màxima prioritat i no hi ha més d'un node màster en el bus.
9. *wb_ack_i*: a través d'aquesta senyal el node esclau indica al node màster que la transferència s'ha realitzat correctament.
10. *wb_err_i*: aquesta senyal indica si hi ha hagut algun error durant la transferència. Aquesta senyal no serà utilitzada ja que el nostre mòdul no consta de control d'errors.

Per tant, el processador, quan vulgui fer una accés a memòria activarà les senyals *wb_cyc_o* i *wb_stb_o*, ens indicarà el tipus d'operació a través de la senyal *wb_we_o* i l'adreça a la que s'accedeix a través de *wb_adr_o*.

6.2 ARQUITECTURA DEL MÒDUL

Les funcions que cal afegir són encriptació, desencriptació i memòria. Es va decidir implementar aquestes funcionalitats en 3 submòduls diferents, però tots dins el mateix mòdul. La decisió de fer un únic mòdul es deu a la simplicitat que això comporta a la hora de connectar-ho amb el processador, ja que si connectéssim els tres mòduls per separat al processador, hi hauria ports de l'arquitectura de bus *Wishbone* que anirien connectats a l'encriptador, altres a memòria i altres al desencriptador. El que aconseguim connectant només un mòdul al processador és que l'assignació de les senyals als diferents submòduls es pot fer sense modificar codi dels fitxers del processador.

A la figura 5 es pot observar un diagrama del mòdul dissenyat amb els ports d'entrada i sortida seguint l'arquitectura de bus *Wishbone* explicada anteriorment. La senyal d'adreça, tipus d'operació i selecció de bytes van directes a memòria, mentre que el bus amb les dades passa primer per el mòdul d'encriptació, i les senyals de *cycle* i *strobe* van directes a la màquina d'estats del mòdul. La senyal de sortida de dades surt del desencriptador i la senyal de *acknowledge* surt de la maquina d'estats, un cop s'ha finalitzat la operació d'escriptura o lectura. A continuació s'analitzen els diferents mòduls desenvolupats en el projecte que es veuen a la figura 5. El codi usat per definir aquest mòdul es troba a l'annex 1.

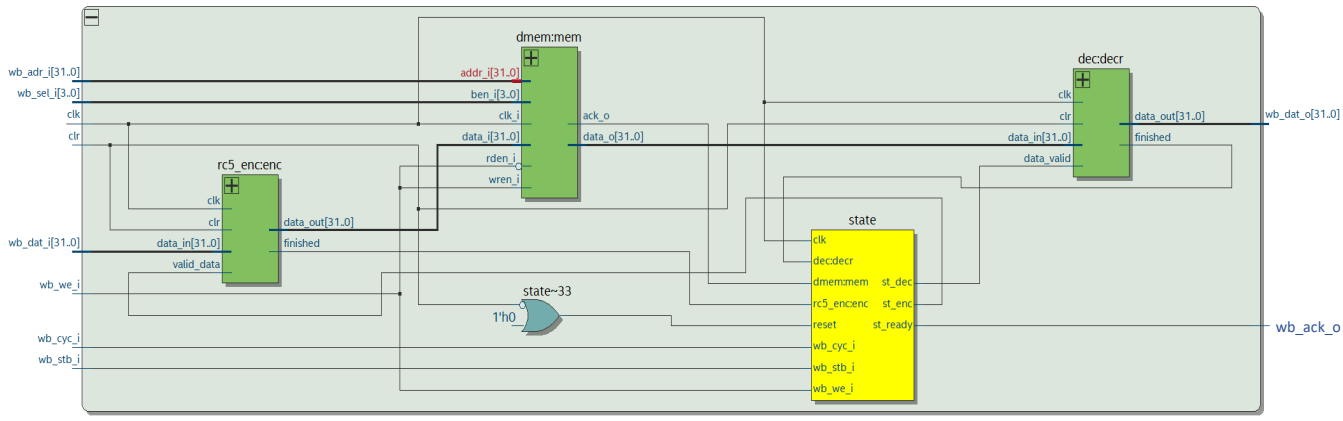

Figura 5. Arquitectura mòdul general. Font pròpia.

6.3 MÀQUINA D'ESTATS

La màquina d'estats consta de 6 estats:

1. *St_idle*: Aquest és l'estat de repòs o espera. Espera a que les senyals de *cycle* i *strobe* s'activin per començar la següent operació de lectura o escriptura en funció de la senyal *write_enable.*
2. *St_mem_load*: Estat d'accés de lectura a memòria. En aquest estat s'obtenen les dades de memòria que es volen llegir i s'envien al desencriptador. S'accedeix a aquest estat si la senyal *write_enable* val 0 quan s'activen les senyals *cycle* i *strobe*.
3. *St_enc*: Estat d'encriptació. En aquest estat es fa la encriptació de les dades. S'accedeix a aquest estat si la senyal *write_enable* val 1 quan s'activen les senyals *cycle* i *strobe*.
4. *St_dec*: Estat de desencriptació: En aquest estat s'efectua la desencriptació de les dades un cop s'han llegit de memòria en una operació de lectura.
5. *St_mem_store*: Estat d'accés d'escriptura a memòria. En aquest estat s'efectua l'operació d'escriptura de memòria un cop les dades han estat encriptades.
6. *St_ready*: Estat de finalització. Aquest estat indica que l'operació ja ha finalitzat i activa la senyal de *acknowledge* durant un cicle.

A la figura 7 es pot observar el diagrama d'estats i a la taula 3 la taula de transicions entre estats.

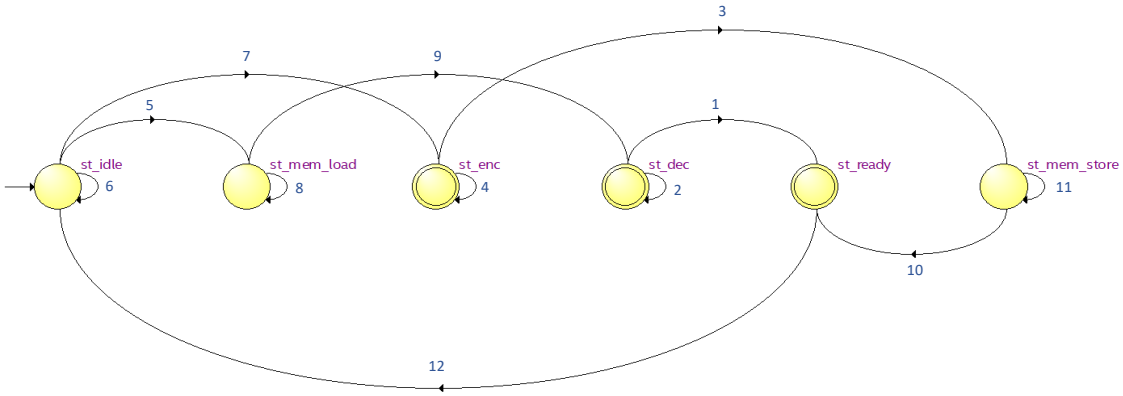

Figura 6. Diagrama d'estats. Font pròpia.

	Estat inici	Estat destí	Condició
1	st_dec	st_ready	dec_finished
2	st_dec	st_dec	!dec_finished
3	st_enc	st_mem_store	enc_finished
4	st_enc	st_enc	!enc_finished
5	st_idle	st_mem_load	(wb_stb_i).(wb_cyc_i).(!wb_we_i)
6	st_idle	st_idle	(!wb_stb_i) + (wb_stb_i).(!wb_cyc_i)
7	st_idle	st_enc	(wb_stb_i).(wb_cyc_i).(wb_we_i)
8	st_mem_load	st_mem_load	!mem_finished
9	st_mem_load	st_dec	mem_finished
10	st_mem_store	st_ready	mem_finished
11	st_mem_store	st_mem_store	!mem_finished
12	st_ready	st_idle	

Taula 3. Taula de transicions. Dec = Desencriptador, enc = encriptador, mem = memòria. Font pròpia.

6.4 MÒDUL D'ENCRIPTACIÓ

En una operació d'escriptura el primer que es fa es encriptar les dades a escriure a memòria, per això s'ha implementat un mòdul d'encriptació que segueix el algorisme RC5. Els motius de selecció ja s'han explicat anteriorment, per tant en aquesta secció es repassen breument els avantatges que proporciona aquest algorisme i s'explica en profunditat el funcionament d'aquest. També s'explica l'arquitectura del mòdul i la màquina d'estats interna. El codi que defineix aquest mòdul es troba a l'annex 2.

6.4.1 Arquitectura

Aquest mòdul té 4 ports d'entrada i 2 de sortida:

1. ***Clk***: La senyal de rellotge d'entrada indica el temps de cicle.
2. ***Clr***: La senyal de *clear* s'usa de *reset*, és *down-active*, és a dir que es fa *reset* quan la senyal val 0.
3. ***Data_in***: Per aquest port d'entrada es subministren les dades a encriptar. Un vector de 32 bits directament subministrat des del processador.
4. ***Valid_data***: Aquesta senyal d'entrada ve des de la màquina d'estats i indica que el port *data_in* conté dades vàlides i que es pot començar a encriptar.
5. ***Data_out***: Port de sortida que conté les dades ja encriptades. Un vector de 32 bits que s'envia al mòdul de memòria per a que es realitzi la operació de escriptura.
6. ***Finished***: Senyal de sortida que indica que s'ha finalitzat la encriptació i que, per tant, les dades que conté *data_out* ja són vàlides.

6.4.2 Característiques de l'algorisme

L'algorisme RC5 es caracteritza per la seva flexibilitat i simplicitat. RC5 es un algoritme de xifratge per blocs, i permet treballar blocs de dades de 32, 64 i 128 bits. Es pot decidir el nombre de rondes des de 0 fins a 255 i la mida de la clau en bytes des de 0 fins a 255 també. Aquest és un dels pocs algoritmes que permet treballar amb 32 bits i ens permetrà evitar tenir una memòria cau per tal de poder encriptar dades.

S'ha pres la decisió de utilitzar un nombre de rondes reduït, r = 10 concretament, per reduir la penalització en el rendiment del processador en els accessos a memòria. I s'ha decidit que la mida de la clau serà de b = 8 bytes, en total 64 bits, per tal de reduir l'espai de memòria a ocupar. Aquest algoritme es pot separar en dos parts, l'algorisme en sí i l'algorisme d'obtenció de la clau. A continuació veurem que no s'utilitza directament la clau per xifrar les dades si no un conjunt de valors que s'obtenen a partir de la clau. El procés per obtenir aquests valors és força costós i si a cada accés a memòria s'utilitza una clau diferent, s'ha d'executar aquest algorisme i afecta molt el rendiment del mòdul. A més a més, s'hauria de instal·lar un sistema d'etiquetes per saber quina clau se li ha assignat a cada posició de memòria, per poder desencriptar en la operació de lectura. És per això que s'ha decidit utilitzar una única clau per a tots els accessos.

Un cop s'han seleccionat aquests paràmetres es pot començar a implementar l'algorisme, començant per l'obtenció de la clau, que es divideix en diversos passos:

1. **Inicialitzar les dos constants P i Q.** En el nostre cas, amb un bloc d'entrada amb mida de 32 bits, P i Q valen b7e1 i 9e37 en hexadecimal respectivament.
2. **Convertir la clau secreta K de bytes a paraules.** La clau de mida b bytes s'utilitza per inicialitzar el vector L que conté c paraules, on c = b/u, u = w/8 i w = mida de la paraula,

en el nostre cas 16, ja que el algorisme RC5 divideix en dos el bloc d'entrada i treballa amb dos blocs per separat, per tant en la nostra instància de l'algorisme RC5, la mida de la de la paraula equival a 32/2 = 16 bits. Llavors, el vector L conté u = 16/8 = 2 i c =8/2 = 4 paraules.

Com que sempre utilitzarem la mateixa clau, per reduir els cicles d'execució del mòdul, el vector L sempre valdrà el mateix i serà la clau secreta de l'algorisme expressada en paraules. La clau escollida per aquest mòdul d'encriptació és 0001020304050607. I per tant, L[0] = 0100, L[1] = 0302, L[2] = 0504, L[3] = 0706.

3. **Inicialització de la subclau S.** La subclau S és un vector de mida t = 2(r+1) = 22. S'inicialitza usant les constants P i Q amb el següent algorisme:

```
S[0] = P
for i = 1 to 2(r+1)-1
  S[i] = S[i-1] + Q)
```

4. **Mescla de la subclau.** A partir del següent algorisme, mesclant S i L, obtenim a S la subclau definitiva.

```
i = j = 0
A = B = 0
do 3 * max(t, c) times:
  A = S[i] = (S[i] + A + B) <<< 3
  B = L[j] = (L[j] + A + B) <<< (A + B)
  i = (i + 1) % t
  j = (j + 1) % c
```

Fins aquest pas l'objectiu és la obtenció del vector S, que conté uns valors determinats a partir de la clau. Com que hem decidit utilitzar sempre la mateixa clau, aquests passos s'executaran només un cop, ja que el vector S serà sempre el mateix.

5. **Encriptació.** Com s'ha mencionat, es divideix el bloc de 32 bits en dos blocs A i B de mida w bits. Després de passar per el procés d'encriptació la unió dels dos blocs forma el bloc xifrat. L'algorisme d'encriptació és:
   a. Inicialització dels blocs de text A i B afegint S[0] i S[1] a A i B respectivament. Aquestes operacions són mod(2^w).
   b. XOR A i B.
   c. Desplaçament cíclic a l'esquerra del nou valor de A per B.
   d. Afegir S[2*i] a la sortida del pas anterior. Aquest és el nou valor de A.
   e. XOR B amb el nou valor de A i emmagatzemar a B.
   f. Desplaçament cíclic a l'esquerra del nou valor de B per A.
   g. Afegir S[2*i+1] a la sortida del pas anterior. Aquest és el nou valor de B.
   h. Repetir el procediment sencer (excepte la inicialització) r vegades (10 vegades).

   El pseudocodi per l'encriptació seria el següent:

```
A = A + S[0]
B = B + S[1]
for i = 1 to r do:
  A = ((A ^ B) <<< B) + S[2 * i]
  B = ((B ^ A) <<< A) + S[2 * i + 1]
return A, B
```

6.4.3 Màquina d'estats

El mòdul d'encriptació compte també d'una maquina d'estats, aquesta consta de 4 estats: *Idle*, *preround*, *oproud* i *ready*.

1. *Idle*: Estat d'espera, un cop *valid_data* és activada es passa a l'estat *preround*.
2. *Preround*: En aquest estat es porta a terme la ronda inicial en la qual s'executa el pas a. de l'encriptació explicat anteriorment.
3. *Oproud*: Es porta a terme les iteracions del algorisme d'encriptació, en aquest estat s'hi estarà 10 cicles, 1 per cada ronda.
4. *Ready*: Estat per indicar que s'ha completat la encriptació i la sortida ja és correcte. S'activa la senyal de *finished*.

A la figura 7 es pot observar el diagrama d'estats de l'encriptador i a la taula 4 la transició entre estats.

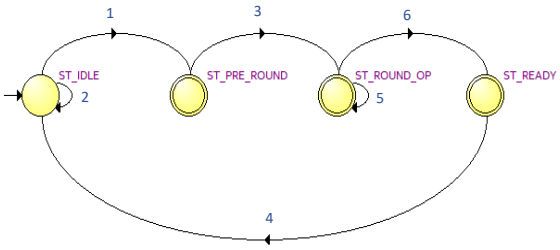

Figura 7. Diagrama d'estats del encriptador. Font pròpia.

	Estat inici	Estat destí	Condició
1	idle	preround	valid_data
2	idle	idle	!valid_data
3	preround	oproud	
4	ready	idle	
5	oproud	oproud	r<10
6	oproud	ready	r = 10

Taula 4. Taula de transició d'estats. r = ronda actual. Font pròpia.

6.5 MÒDUL DE DESENCRIPTACIÓ

El mòdul de desencriptació és molt similar al mòdul d'encriptació, ja que executen la mateixa tasca però a la inversa, per tant hi veurem moltes similituds. El codi que defineix aquest mòdul es troba a l'annex 3.

6.5.1 Arquitectura

Aquest mòdul també té 4 ports d'entrada i 2 de sortida:

1. ***Clk***: La senyal de rellotge d'entrada indica el temps de cicle.
2. ***Clr***: La senyal de *clear* s'usa de *reset*, és *down-active*, és a dir que es fa *reset* quan la senyal val 0.
3. ***Data_in*:** Per aquest port d'entrada es subministren les dades a desencriptar. Un vector de 32 bits directament subministrat des de memòria.
4. ***Valid_data***: Aquesta senyal d'entrada ve des de la màquina d'estats i indica que el port *data_in* conté dades vàlides i que es pot començar a desencriptar.
5. ***Data_out**:* Port de sortida que conté les dades ja desencriptades. Un vector de 32 bits que s'envia com a sortida del mòdul general cap al bus *Wishbone*.
6. ***Finished***: Senyal de sortida que indica que s'ha finalitzat la desencriptació i que, per tant, ha finalitzat també l'accés de lectura.

6.5.2 Algorisme

L'algorisme de desencriptació és bàsicament el algorisme de encriptació invertit, el seu pseudocodi seria el següent:

```
for i = r down to 1 do:
   B = ((B - S[2 * i + 1]) >>> A) ^ A
   A = ((A - S[2 * i]) >>> B) ^ B
B = B - S[1]
A = A - S[0]
return A, B
```

6.5.3 Màquina d'estats

El mòdul de desencriptació compte també d'una maquina d'estats, aquesta consta de 4 estats: *Idle*, *opround*, *postround* i *ready*.

1. *Idle*: Estat d'espera, un cop *valid_data* és activada es passa a l'estat *opround*.
2. *Opround*: Es porta a terme les iteracions del algorisme de desencriptació, en aquest estat s'hi estarà 10 cicles, un per cada ronda.
3. *Postround*: En aquest estat es porta a terme la operació inversa a la que es realitza en la ronda inicial de l'encriptació.
4. *Ready*: Estat per indicar que s'ha completat la desencriptació i la sortida ja és correcte. S'activa la senyal de *finished*.

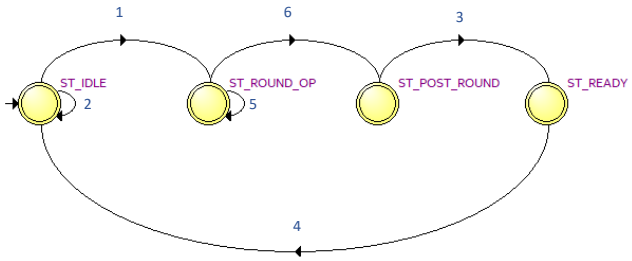

Figura 8. Diagrama de estats del mòdul de desxiframent. Font pròpia.

	Estat inici	Estat destí	Condició
1	idle	opround	data_valid
2	idle	idle	!data_valid
3	postround	ready	
4	ready	idle	
5	opround	opround	r>1
6	opround	postround	r=1

Taula 5. Taula de transicions entre estats. r = ronda. Font pròpia.

6.6 MÒDUL DE MEMÒRIA

El mòdul de memòria que s'ha utilitzat és molt similar al que ja incorporava el propi processador anteriorment com a memòria de dades interna. El codi que defineix aquest mòdul és a l'annex 4.

6.6.1 Arquitectura

Aquest mòdul té 6 ports d'entrada i 2 de sortida:

1. ***Clk***: La senyal de rellotge d'entrada indica el temps de cicle.
2. ***Data_in*****:** Per aquest port d'entrada es subministren les dades a escriure en una operació de escriptura. Un vector de 32 bits directament subministrat des de el mòdul d'encriptació.
3. ***Addr_in*****:** Port d'entrada on s'indica la adreça de memòria a llegir o escriure. És un vector de 32 bits que ve directament des del processador.
4. ***Ben_i*****:** Senyal de *Byte Enable* d'entrada, que indica a quins bytes de la paraula es porta a terme la operació d'accés a memòria. És un vector de 4 bits, un per cada byte de la paraula, on si el bit n és actiu indica que s'accedeix al byte n de la paraula.

5. ***Rden_i*:** Senyal de *Read Enable* d'entrada, un bit que indica si és un accés de lectura.
6. ***Wren_i*:** Senyal de *Write Enable* d'entrada, un bit que indica si és un accés de escriptura.
7. ***Ack_o*:** Senyal de sortida *acknowledge*. Un bit que indica que s'ha finalitzat l'accés a memòria.
8. ***Data_out***: Port de sortida que conté les dades de memòria de l'adreça indicada. Un vector de 32 bits que s'envien cap al mòdul desencriptador.

6.6.2 Espai d'adreces

La memòria consta de 8192 paraules de 32 bits, que equival a un total de 32 kB de memòria de dades. A la figura 9 es pot observar la organització de l'espai d'adreces del processador. A partir de la direcció 0x80000000[2] comença l'espai reservat per a la memòria de dades. La mida per defecte de la memòria de dades és de 2 GB, però en el nostre entorn l'hem modificat a 32 kB. Els demés espais d'adreces es mantenen en la configuració per defecte i seran interns al processador.

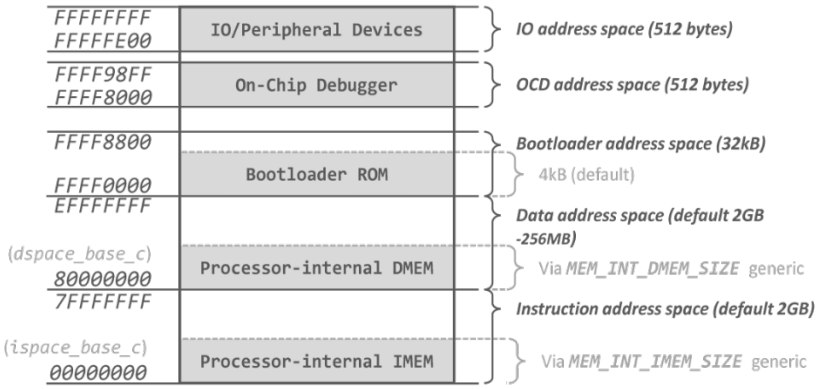

Figura 9. Espai d'adreces del processador. Font: https://stnolting.github.io/neorv32/img/address_space.png

Dins la memòria es compte amb un senyal anomenat *acc_en (access enable)* que s'activa quan l'accés és a una direcció dins del rang de adreces que pertoca a la espai de dades.

[2] 0x davant d'una xifra indica que aquesta es troba en base hexadecimal.

7 PROVES DE FUNCIONAMENT DEL MÒDUL

Per tal de comprovar i demostrar el correcte funcionament del mòdul general es porta a terme primer una simulació del comportament dels senyals externs, abans de introduir-lo dins el processador.

La primera prova que es realitza és una operació d'escriptura seguit d'una operació de lectura a la mateixa direcció de memòria. Es vol comprovar que la lectura retorna el valor que s'ha escrit a memòria, i que per tant el procés d'encriptació i desencriptació és correcte. Per aquesta prova s'han de simular les senyals que enviaria el processador a través del bus d'arquitectura *Wishbone*. Les senyals d'entrada actuen de la següent manera:

1. *Clock*: La senyal de rellotge es canvia de valor cada 5 nanosegons, el que suposa un temps de cicle de 10 nanosegons.
2. *Clr*: Aquesta senyal es posa a 1 als 10 nanosegons.
3. *Address*: L'accés a memòria es fa a la primera direcció, per tant l'adreça és 0x80000000.
4. *Data_in*: Per el bus de dades entrarà el valor 0x12345678, el qual s'escriurà a memòria encriptat, i s'espera llegir aquest valor per la sortida en la operació de lectura.
5. *Write Enable*: La senyal de escriptura val 1 tota la simulació fins als 190 nanosegons que passarà a valer 0, i s'efectuarà la lectura.
6. Selecció: El vector de selecció valdrà sempre 0b1111[3]. S'accedeix a tots els bytes de la paraula en totes les operacions.
7. *Strobe*: La senyal de *strobe* s'activa als 20 nanosegons, quan s'efectuarà la operació de escriptura i als 190 nanosegons, quan s'efectuarà la operació de lectura.
8. Cycle: La senyal de *cycle* valdrà 1 tota la simulació. Serà llavors la senyal de *strobe* la que indicarà el inici de cada operació.

Hem de tenir en compte que les senyals de *cycle* i *strobe* no funcionen seguint el protocol de l'arquitectura Wishbone, aquestes senyals es mantindrien actives durant tot l'accés fins que es rebés la senyal de *acknowledge* per part del mòdul, i llavors passarien a valdre 0. El comportament assignat a aquestes senyals no afecta al resultat de l'operació, ja que el mòdul només comprova les senyals al principi un cop, i prou.

En la figura 10 es pot observar el resultat de la primera part de la simulació, l'operació d'encriptació i escriptura de memòria, i el comportament dels senyals interns. S'han classificat les senyals en 5 grups diferents:

1. Senyals d'entrada del mòdul general,
2. Senyals de sortida del mòdul general,
3. Senyal d'estat del mòdul general,
4. Senyals internes del mòdul d'encriptació i
5. Senyals internes del mòdul de memòria.

[3] 0b davant d'una xifra indica que aquesta es troba en base binaria.

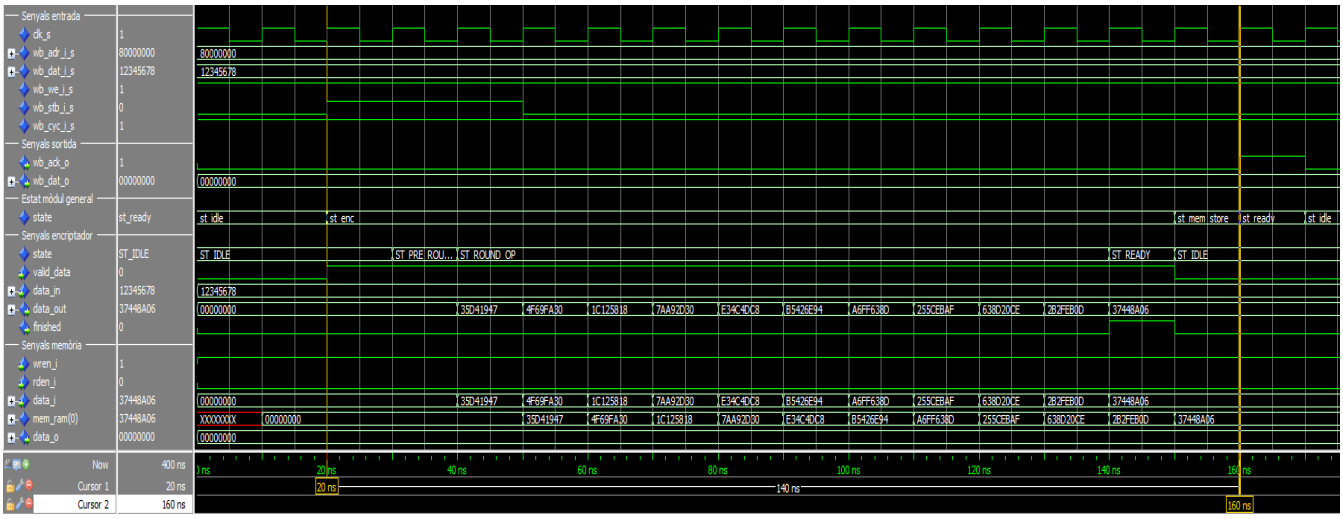

Figura 10. Diagrama en forma d'ona de les senyals de l'operació de escriptura. Font pròpia.

Comprovem el funcionament del mòdul analitzant pas a pas el que succeeix a la simulació que s'observa a la figura 11.

1. El mòdul es troba en estat de espera fins que la senyal d'entrada *strobe (wb_stb_i_s)* s'activa, als 20 nanosegons.
2. Quan la senyal de *strobe* és activada, s'activa la senyal de vàlid del encriptador i aquest mòdul comença a encriptar les dades que li arriben per l'entrada. A la senyal *data_out* dins la secció de senyals internes de l'encriptador trobem el resultat de cada ronda de l'algorisme d'encriptació.
3. Finalment, al cap de 10 rondes (11 si comptem la inicialització), el valor de la senyal de sortida *data_out* és correcte i el mòdul d'encriptació passa a estat *ready*, i activa la senyal *finished*, als 140 nanosegons.
4. Un cop l'encriptador té les dades encriptades, el mòdul general passa a l'estat *mem_store* i es realitza la escriptura de memòria, als 150 nanosegons.
5. Per acabar la operació d'escriptura, el mòdul general passa a estat *ready* i activa la senyal de sortida *wb_ack_o*, que indica *acknowledge* i per tant que la operació de escriptura finalitza correctament, als 160 nanosegons. Podem observar que la primera posició de memòria (senyal *mem_ram*(0)) conté el valor de sortida de l'encriptador, 0x37448A06, aquest es el valor 0x12345678 encriptat.

A continuació, a la figura 11, es pot veure el comportament del mòdul en l'operació de lectura, en aquest cas s'ha canviat la secció de senyals internes de l'encriptador per la secció de senyals internes del desencriptador, ja que en aquesta operació el mòdul de encriptació no actua.

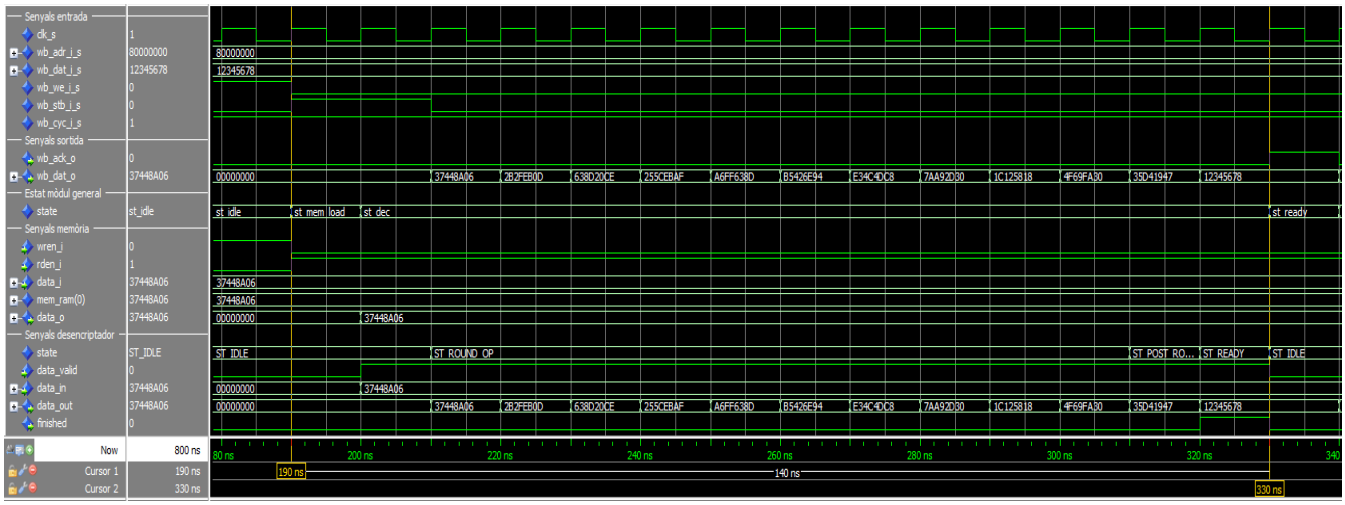

Figura 11. Diagrama de forma d'ona de les senyals de l'operació de lectura. Font pròpia.

Tornem a comprovar doncs el comportament correcte del mòdul, ara en un cas de operació de lectura, pas per pas.

1. S'activa als 190 nanosegons la senyal *strobe* de nou (*wb_stb_i_s*) indicant que comença el nou accés. A la vegada, la senyal de *write enable (wb_we_i_s)* es posa a 0 indicant que es tracta d'una operació de lectura. El mòdul passa de l'estat d'espera a l'estat de lectura de memòria (*mem_load*).
2. Durant l'estat de lectura de memòria, el mòdul de memòria accedeix a l'adreça indicada, que segueix sent 0x80000000. Als 200 nanosegons la dada a llegir encriptada es troba a la sortida de la memòria, com es pot veure a la senyal *data_o*. Es passa a l'estat de desencriptació (*st_dec*).
3. S'activa la senyal de *data_valid* del mòdul de desencriptació i aquest comença a desencriptar la dada llegida de memòria. Al cap de 11 rondes (incloent la ronda final, inversa a la inicial de l'algorisme de encriptació) el mòdul de desencriptació passa a l'estat *ready* i activa la senyal *finished*, indicant que la senyal *data_out* conté la dada a llegir ja desencriptada.
4. El mòdul general passa a estat *ready* i activa la senyal de *acknowledge*, indicant que la operació s'ha acabat i la dada a llegir és correcte, als 330 nanosegons.

Observem que el vector de bits que conté la senyal *data_out* als 320 nanosegons dins la secció de senyals internes del desencriptador és 0x12345678, igual al que s'havia escrit en l'operació anterior. De fet es pot observar que el port de sortida del mòdul de desencriptació conté els mateixos valors a cada ronda que la sortida del mòdul d'encriptació en la figura 10, però amb l'ordre invers. Efectivament la encriptació i desencriptació es portem a terme de manera correcte i es pot procedir a connectar el mòdul al processador. A continuació es fa un estudi de rendiment del mòdul.

8 ESTUDI DEL RENDIMENT DEL MÒDUL

Com s'ha presentat en la secció 7.3, el temps d'execució de les instruccions d'accés a memòria depenen de la latència de memòria, per tant es pot fer una comparació entre els accessos a memòria amb i sense encriptació. Òbviament, ens esperem un augment de latència en els accessos a memòria, ja que estem afegint la funcionalitat de encriptació i desencriptació en els accessos. En aquesta secció s'analitza l'augment del temps d'execució en cicles dels accessos a memòria a causa del nou mòdul de encriptació dissenyat.

A la figura 12 s'observa un accés a la memòria de dades interna. En el segon flanc ascendent del rellotge es veu com el valor del bus *addr_i* (senyal d'adreça d'entrada) conté la adreça 0x80001FF8 que es troba dins l'espai d'adreces de la memòria de dades, (0x80000000 – 0xEFFFFFFF). En el tercer flanc ascendent de rellotge el bus *data_o* (senyal de dades de sortida) conté el valor de memòria de la adreça indicada, en aquest cas 0. Per tant, la latència del mòdul de memòria interna al processador és de 1 cicle.

Llavors, seguint la taula 2, una instrucció d'accés a memòria tarda 5 cicles en el cas de que s'utilitzi la memòria interna del processador.

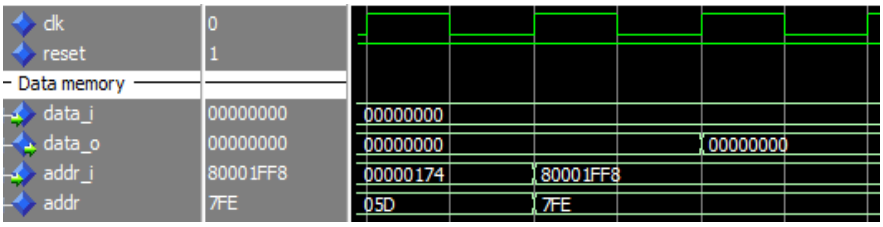

Figura 12. Diagrama d'ones de les senyals en un accés a memòria de dades. Font pròpia.

Observem ara la figura 13, on es mostra un accés de lectura, el que ja havíem vist anteriorment a la figura 11, al mòdul de memòria amb encriptació i desencriptació dissenyat en aquest projecte.

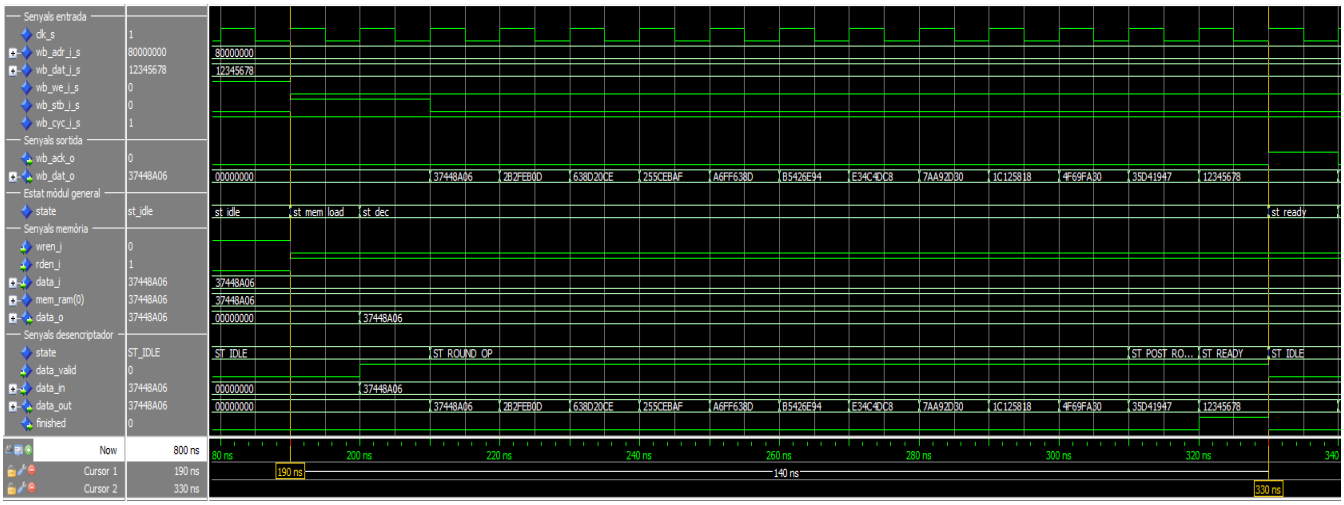

Figura 13. Diagrama de forma d'ona de les senyals de l'operació de lectura. Font pròpia.

Es pot observar que el temps d'accés a memòria són 140 nanosegons, que equivalen a 14 cicles, ja que el temps de cicle és 10 nanosegons. D'aquests 14 cicles, 11 es dediquen al procés de desencriptació, on a cada cicle es realitza les operacions de cada ronda de l'algorisme (10 rondes més la ronda d'inicialització), la memòria requereix d'un altre cicle per a accedir a la adreça determinada, i els altres dos cicles són destinats a l'estat de *idle* i l'estat de *ready* del mòdul desencriptador, on s'indica que les dades de sortida ja són correctes.

Per tant, la latència de memòria és ara 14 cicles. I seguint la formula utilitzada anteriorment, el temps d'execució d'una instrucció d'accés a memòria és de 5 + (14 – 1) = 18 cicles. Així doncs, la diferencia de temps d'execució en cicles d'un accés a memòria utilitzant el mòdul d'encriptació i desencriptació enfront de no utilitzar encriptació és de 13 cicles.

En la secció de observacions i millores del projecte s'observa la possibilitat de reduir el nombre de cicles en l'encriptació i desencriptació de memòria.

Es té en compte que tan la operació de lectura com la de escriptura tarden el mateix nombre de cicles, i per tant, el tipus de operació no afecta en la comparació de rendiment.

En un sistema real aquest mòdul se situaria entre el processador i la DRAM (fora del xip). En aquest cas, l'impacte sobre el rendiment seria molt menor ja que l'accés a la DRAM pot arribar a 250ns [13].

9 CONNEXIÓ DEL MÒDUL AL PROCESSADOR

Procedim ara a la connexió via software del mòdul dissenyat i presentat anteriorment al processador NEORV32 amb el que es vol treballar. S'ha creat una entitat superior per tal de connectar el processador al mòdul i portar les senyals correctament. El mòdul de encriptació i desencriptació de memòria és per tant totalment extern al processador. A la figura 14 es mostra la connexió entre el processador i el mòdul d'encriptació de memòria, eliminant senyals que no afecten a la connexió, obtingut a partir de *Quartus*. La entitat més alta del mòdul dissenyat rep el nom de *test_enc*, i la entitat més alta del processador rep el nom de *neorv32_top*. Com que el mòdul de memòria extern no té control d'errors s'envia sempre un 0 per el port d'entrada *wb_err_i* que notifica al node mestre (el processador) si hi ha error en l'operació. No s'utilitza la senyal *wb_tag_o* ja que no treballem amb accessos per blocs.

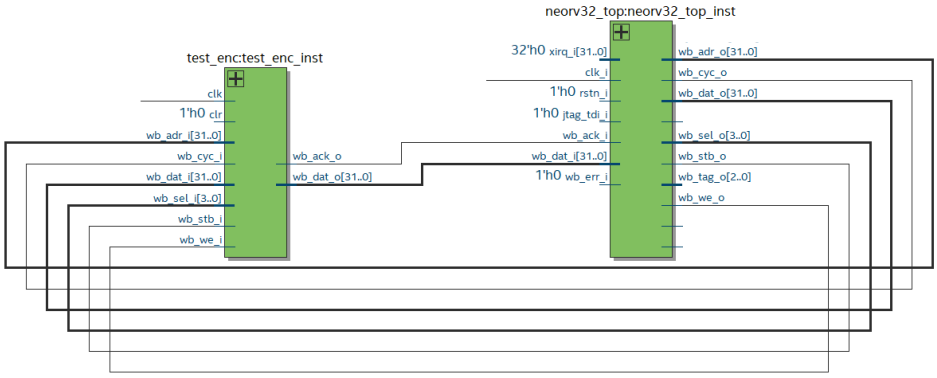

Figura 14. Diagrama de blocs de la connexió entre el processador i el mòdul de memòria. Font pròpia.

El processador consta de un mòdul intern anomenat *Wishbone* que bàsicament s'encarrega d'enviar i rebre les senyals de l'arquitectura *Wishbone* als mòduls externs. Aquest mòdul es mostra a la figura 15.

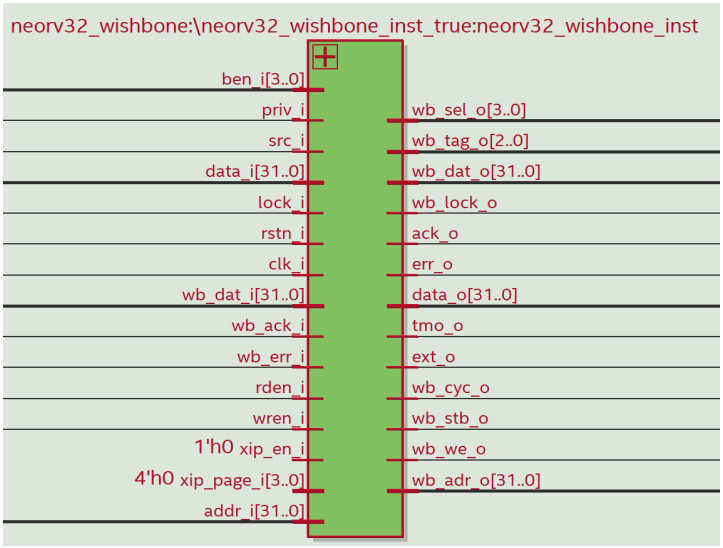

Figura 15. Mòdul intern del processador dedicat a mantenir l'arquitectura de bus Wishbone. Font pròpia.

Aquest mòdul és el que connecta el mòdul de memòria desenvolupat en aquest projecte i el processador en sí. Com es pot veure a la figura 15, el mòdul té varies senyals d'entrada i de sortida, i algunes són per el protocol *Wishbone* i d'altres no. A continuació s'analitzen i es descriuen les senyals més importants d'entrada i sortida del mòdul, en l'ordre de dalt a baix com apareixen en la figura.

SENYALS D'ENTRADA

Des del processador

1. *Ben_i*: Indica a quins bytes de la paraula de 32 bits determinada per l'adreça es volen accedir, és un vector sense signe de 4 bits, un per cada byte de la paraula. Si, per exemple, el bit 3 del vector està a 0, no es modificarà el contingut del byte de més pes de la paraula accedida a memòria. Aquesta senyal s'assigna directament a la senyal de sortida *wb_sel_o.*
2. *Data_i:* Conté els 32 bits de dades que es volen escriure, en cas d'operació d'escriptura, a memòria. Aquesta senyal d'entrada s'assigna directament a la senyal de sortida *wb_dat_o*.
3. *Rden_i:* Senyal que indica que es tracta d'una operació de lectura quan es posa a 1.
4. *Wren_i:* Senyal que indica que es tracta d'una operació de escriptura quan es posa a 1.
5. *Addr_i:* Indica l'adreça de memòria a la qual s'accedeix. Aquesta senyal s'assigna directament a la senyal de *wb_adr_o*.

Des del mòdul extern de memòria

1. *Wb_dat_i:* Conté les dades llegides de memòria en cas d'una operació de lectura. Aquesta senyal s'assigna directament a la senyal de sortida *data_o*.
2. *Wb_ack_i:* A través d'aquesta senyal, el mòdul de memòria extern indica que la operació d'accés a memòria s'ha portat a terme i s'ha executat, aquesta senyal s'assigna directament a la senyal de sortida *ack_o*.
3. *Wb_err_i:* A través d'aquesta senyal, el mòdul de memòria extern indica si la operació d'accés a memòria ha generat algun error. Aquesta senyal s'assigna directament a la senyal de sortida *err_o*.

SENYALS DE SORTIDA

Cap al processador

1. *Ack_o:* Equival a la senyal d'entrada *wb_ack_i.*
2. *Err_o:* Equival a la senyal d'entrada *wb_err_i.*
3. *Data_o:* Equival a la senyal d'entrada *wb_dat_i.*

Cap al mòdul extern de memòria

1. *Wb_sel_o:* Equival a la senyal d'entrada *ben_i.*
2. *Wb_dat_o:* Equival a la senyal d'entrada *data_i.*
3. *Wb_cyc_o*: Quan s'activa, indica que hi ha un cicle de bus vàlid en curs. El senyal s'afirma durant la durada de tots els cicles de bus. En aquest cas equival a la senyal *wb_stb_o.*
4. *Wb_stb_o:* Aquesta senyal d'un bit està activada durant tota una transacció. Indica la validesa de les dades. S'activa quan hi ha un accés a memòria i no és a la memòria d'instruccions, i la senyal de *wren_i* o *rden_i* estan activades.

5. *Wb_we_o:* Indica de quin tipus d'accés és l'operació actual. Si val 1 significa que es tracta d'un accés de escriptura, si val 0, es tracta d'un accés de lectura.
6. *Wb_adr_o:* Equival a la senyal d'entrada *addr_i*.

9.1 SIMULACIÓ D'ACCESSOS DES DEL PROCESSADOR

Un dels objectius d'aquest projecte és poder connectar i simular el funcionament del mòdul d'encriptació amb un processador RISC-V.

Malauradament, no s'ha pogut accedir al mòdul dissenyat al llarg d'aquest projecte des del processador per a poder fer simulacions d'operacions d'accés a memòria. Tot i això, el mòdul funciona correctament i compleix amb l'arquitectura de bus *Wishbone*.

S'ha intentat fer un accés a la memòria interna del processador i tampoc s'executa correctament, cosa que indica que el problema no resideix en el mòdul d'encriptació dissenyat si no que pot ser degut a una mala configuració del processador. És per això que es va contactar amb el dissenyador del processador.

Al realitzar les simulacions, el codi entra en un *trap handler*[4], cosa que indica que salta alguna excepció en l'execució del codi. El problema és que des de l'entorn en el que es treballa, ModelSim dins de Windows, no es pot llegir el motiu de l'excepció, i per tant dificulta molt la resolució d'aquesta. El dissenyador del processador va recomanar mantenir la memòria de dades interna, i fer accessos a la memòria externa a partir de l'adreça 0xF0000000 amb un codi com el que es pot observar a continuació:

```
int main() {

  // inicialitzar el entorn d'execució de neorv32
  // això s'encarregarà de gestionar totes les "traps" de la CPU (interrupcions
i excepcions)
  neorv32_rte_setup();

// configurar UART0 a la velocitat de transmissió predeterminada, sense bits
//de paritat, sense control de flux
  neorv32_uart0_setup(19200, PARITY_NONE, FLOW_CONTROL_NONE);

  int* vector = (int*)0xF0000000UL; // Definir vector a 0xF0000000
  for (int i = 0; i < 4; i++) {
    vector[i] = i +1;
  }

  for (int i = 0; i < 4; i++) {
    neorv32_uart0_printf("vector[%i] @ 0x%x = %i\n", i, (uint32_t)(&vector[i]),
vector[i]);
  }

  return 0;
}
```

I va aconsellar també utilitzar les eines que proporciona el processador. A través d'un entorn Linux, es pot habilitar un mode de simulació d'UART0 imprimint totes les sortides (des de neorv32_uart0_printf en aquest exemple) per la consola.

[4] Aquesta rutina s'executa quan apareix una excepció en l'execució del codi, i s'encarrega de fer la crida a sistema.

Es va procedir per tant a la instal·lació d'una màquina virtual Linux, en concret la imatge virtual proporcionada per la Facultat d'Informàtica de Barcelona *OpenSUSE 15.3.* Es va descarregar tot el codi i les eines del processador i el mòdul d'encriptació dissenyat. Malauradament, el simulador amb el que es treballa per defecte amb el processador, el GHDL no està disponible per a la versió de Linux de la que es disposa, i no s'ha pogut instal·lar cap altre simulador, en especial es va provar amb *ModelSim*. La comunicació amb el dissenyador del mòdul i la busca de diferents solucions ha ocupat 10 dies extra que no es tenien en compte, i degut a la falta de temps s'ha hagut de deixar a part aquest objectiu.

10 PLANIFICACIÓ TEMPORAL

La data d'inici del projecte és el 21 de febrer de 2022 i la data de finalització és el dia 27 de juny de 2022, la data en que es preveu fer la lectura del treball. Per tant, el projecte es portarà a terme durant 18 setmanes, el que equival a 126 dies. Es preveu que aquest projecte ocupi 450 hores de feina (18 crèdits * 25 hores/crèdit). La dedicació diària estimada és de 3,5 hores.

10.1 RECURSOS

10.1.1 Recursos humans

Aquest projecte requereix de 6 rols diferents: director del projecte, investigador, dissenyador de hardware, programador, tester i criptògraf. Tot i això, el projecte es porta a terme només a través d'una sola persona, la qual haurà de assumir tots els rols (menys el de director), segons la tasca a realitzar. Al final d'aquesta secció, a la taula 6 es pot observar les assignacions de tasques a cada rol.

10.1.1.1 Director del projecte (DP)

És l'encarregat de supervisar el projecte.

10.1.1.2 Investigador (I)

És el responsable de la investigació del projecte, fa recerca de les diferents tècniques possibles a utilitzar i redacta la documentació del treball. És l'encarregat de la planificació del projecte. Gestiona els recursos i el personal del projecte per tal d'optimitzar el temps. Participa a la redacció de la documentació.

10.1.1.3 Dissenyador de hardware (DH)

És l'encarregat de dissenyar els mòduls de hardware des d'un punt de vista teòric, no els especifica amb codi.

10.1.1.4 Programador (Pr)

És l'encarregat de implementar en codi els dissenys del dissenyador i l'encarregat de preparar i seleccionar l'entorn de treball juntament amb l'investigador.

10.1.1.5 Tester (T)

L'encarregat de realitzar les proves per validar el funcionament dels diferents mòduls dissenyats i implementats per el dissenyador i el programador i de redactar un informe de errors en cas que es no es compleixin els objectius.

10.1.1.6 Criptògraf (Cr)

És l'encarregat de dissenyar els algorismes de encriptació i adaptar-los a cada entorn. En aquest projecte no es dissenya un algorisme des de zero però s'ha d'entendre el algorisme que s'utilitzarà i s'ha d'adaptar al mòdul que s'introduirà, treballarà conjuntament amb el dissenyador.

10.1.2 Recursos materials

10.1.2.1 Microsoft Office

L'entorn office s'utilitzarà per a la creació de la documentació, això inclou Microsoft Word, Excel, PowerPoint i OneDrive. El OneDrive és el núvol de Microsoft, que s'utilitzarà per guardar la documentació de manera segura i compartida amb el director del projecte i l'investigador.

10.1.2.2 Programari d'especificació de hardware

Serà l'entorn de treball en l'especificació de hardware al llarg del projecte. El programari específic es determinarà més endavant durant la tasca de selecció i estudi de l'entorn.

10.1.2.3 Gantter

Eina de software que s'utilitzarà per dissenyar un diagrama de Gantt en la planificació temporal del projecte.

10.1.2.4 Ordinador

Eina física que s'utilitzarà per treballar amb el programari seleccionat i per a redactar la documentació.

10.1.2.5 Google meets

Eina de software que serveix per mantenir reunions de manera online. S'utilitzarà per la comunicació entre el director i l'investigador del projecte.

10.1.2.6 Manual del llenguatge de especificació

Servirà per a que el programador agafi els coneixements necessaris per a poder implementar els dissenys amb el nou llenguatge de especificació seleccionat.

10.1.2.7 Diagrams.net

Eina de dibuix online, permetrà fer els esquemes dibuix dels mòduls a dissenyar.

10.2 DESCRIPCIÓ DE LES TASQUES

Per aquest projecte s'han dividit les tasques en 6 grups diferents. El primer grup inclou les tasques relacionades amb la selecció i comprensió de l'entorn de treball (E). El segon grup considera la implementació d'una memòria principal al processador (MP). El tercer grup la implementació del mòdul d'encriptació (ME). El quart, inclou les tasques relacionades amb la realització de proves (P) del treball fet. El cinquè grup de tasques agrupa tot el que té relació amb l'elaboració de la documentació (D) i l'últim grup de tasques engloba totes les reunions de seguiment (RS) que es portaran a terme amb el director del projecte. Al final d'aquesta secció, a la figura 16 s'observa un diagrama de Gantt on es representen totes les tasques i en la taula 6 es fa un resum del cost estimat en hores de cada tasca i els recursos usats per cadascuna.

10.2.1 Estudi previ de l'entorn (E)

10.2.1.1 Selecció del processador (E1)

Aquest projecte presenta la introducció d'un mòdul de encriptació en l'accés a memòria, però necessitem un processador ja definit amb el qual treballar i poder implementar-li el mòdul. Per això s'ha de buscar un processador ja creat i definit a Internet. Hi ha moltes opcions, l'objectiu d'aquesta tasca és buscar diferents opcions interessants, comparar-les i escollir el processador que millor s'adapti al projecte i el que sigui més còmode per a la introducció del mòdul posteriorment.

Hi ha varies pàgines web on es poden obtenir processadors ja definits per altres usuaris i descarregar-los de forma segura. S'ha de tenir en compte que el processador ha de ser *open source,* és a dir que sigui legal treballar amb ell de forma lliure i gratuïta.

Aquesta tasca és costosa, ja que per tal de fer una bona selecció s'han de entendre bé els processadors, això vol dir comprendre què fa cada senyal i fer-se una idea de com es podria implementar el mòdul de encriptació amb el sistema de senyals de cada processador. La tasca és molt important, si es troba un processador i s'entén bé tot el funcionament des d'un principi, pot facilitar molt el projecte. S'estima que aquesta tasca requerirà 50 hores.

Aquesta tasca requereix del dissenyador i investigador com a recurs humà per a portar a terme la selecció adient.

10.2.1.2 Estudi del llenguatge de descripció (E2)

Aquesta tasca està molt relacionada amb la anterior. Per tal de definir els processadors a través de software s'utilitza un llenguatge de descripció de hardware. Hi ha varis, els més populars són VHDL i *Verilog*. A la hora de seleccionar un processador, també s'ha de tenir en compte el llenguatge de descripció de hardware que s'utilitza. Es té en compte que pot donar-se el cas que el dissenyador no tingui experiència amb el llenguatge seleccionat. Si més no, tots s'assemblen i no hauria de suposar gaire temps entendre un nou llenguatge de descripció de hardware. Aquesta tasca té com a objectiu l'estudi i el repàs de conceptes del llenguatge de descripció seleccionat. Si es té un bon coneixement del llenguatge de programació que s'usa, el temps de implementació serà menor. S'estima que aquesta tasca requerirà de 15 hores.

Aquesta tasca requereix de manuals de llenguatge com a recurs material i del programador com a recurs humà per poder portar a terme un estudi òptim del llenguatge seleccionat.

10.2.1.3 Selecció i estudi del programari (E3)

Un cop s'ha seleccionat el processador i es té coneixement del llenguatge que s'utilitzarà, s'ha de seleccionar un programa amb el qual treballar. No tots els programaris accepten tots els llenguatges de descripció de software, i hi ha alguns que estan més ben dissenyats per treballar amb certes llenguatges que d'altres. Aquesta tasca té com a objectiu seleccionar un programari adequat al llenguatge de descripció de hardware que s'utilitzarà i entendre les possibilitats que ofereix i com funciona, en general. Aquesta tasca no requereix de gaire temps, ja que és força senzilla. S'estima un temps de 10 hores per a la realització d'aquesta tasca. Una vegada s'ha escollit processador, aquesta tasca es pot efectuar paral·lelament a E2.

Aquesta tasca requereix del programador i el tester per portar a terme la selecció i instal·lació adequada.

10.2.2 Mòdul memòria principal (MP)

10.2.2.1 Disseny del mòdul de memòria (MP1)

Un cop s'ha seleccionat un processador i l'entorn amb el que es treballarà el primer que s'ha de fer és afegir-li un mòdul de memòria. El processador definit que seleccionarem no incorporarà un mòdul de memòria principal, el qual es necessari per simular els accessos a memòria que volem encriptar. Aquesta memòria no cal que sigui gran, ha de ser suficient per poder llegir i escriure-hi dades i provar el funcionament del nostre mòdul d'encriptació que implementarem més endavant.

Aquesta tasca té com a objectiu dissenyar un mòdul que simuli una memòria principal. Això inclou definir els senyals d'entrada i sortida que es necessitaran i el funcionament de la memòria. En aquesta tasca no es tocarà codi, sinó que es farà un esquema dibuix que representarà el mòdul a implementar.

Per aquesta tasca s'ha d'entendre molt bé totes les senyals del processador per tal de fer una bona implementació del mòdul, que no tingui errors que puguin afectar més endavant al desenvolupament del mòdul d'encriptació. S'estima un total de 40 hores per la realització d'aquesta tasca.

Aquesta tasca té com a recurs material el software diagrams.net que s'usarà per realitzar l'esquema dibuix amb el disseny del mòdul. I el dissenyador com a recurs humà.

10.2.2.2 Implementació del mòdul de memòria (MP2)

Després tenir el mòdul de memòria dissenyat es procedeix a la seva implementació. L'objectiu d'aquesta tasca és implementar amb codi el mòdul dissenyat a la tasca MP1. Si el disseny de la tasca MP1 és correcte i està ben fet, la implementació en codi no hauria de ser complicada, tot i que pot requerir de cert temps depenent de la destresa del programador amb el llenguatge de descripció de hardware que s'hagi seleccionat. Es calcula que aquesta tasca requerirà de 25 hores.

Aquesta tasca requereix del programari com a recurs material, on es desenvoluparà el codi del mòdul de memòria. I el del programador, que implementarà el codi de especificació del mòdul, com a recurs humà.

10.2.3 Mòdul d'encriptació (ME)

10.2.3.1 Disseny de les senyals del mòdul d'encriptació (ME1)

Un cop tenim el processador i el mòdul de memòria ben implementats i es comuniquen correctament, ja es pot portar a terme el desenvolupament del mòdul d'encriptació en els accessos a memòria. El primer pas és el disseny d'aquest. Aquesta tasca té com a objectiu definir els senyals d'entrada i sortida a nivell teòric. En aquesta tasca no es tocarà codi, sinó que es farà un esquema dibuix que representarà el mòdul a implementar. Aquesta tasca es on es defineix el mòdul principal del projecte. Es calcula que requerirà de 50 hores de feina.

Aquesta tasca té com a recurs material el software diagrams.net que s'usarà per realitzar l'esquema dibuix amb el disseny del mòdul. I el dissenyador com a recurs humà.

10.2.3.2 Disseny del algorisme de encriptació (ME2)

Un cop tenim dissenyat el mòdul en quant a senyals dins l'entorn processador-memòria en el qual serà implementat es necessita dissenyar el algorisme d'encriptació que s'utilitzarà. Aquesta tasca té com a objectiu dissenyar el algorisme que s'usarà dins el mòdul i estudiar si calen més senyals per tal de que el mòdul realitzi la tasca desitjada correctament. Es calcula que es requeriran 45 hores de feina per a la realització d'aquesta tasca.

Aquesta tasca té com a recurs material el software diagrams.net que s'usarà per modificar l'esquema dibuix del disseny del mòdul en cas que sigui necessari. I el criptògraf com a recurs humà.

10.2.3.3 Implementació del mòdul d'encriptació (ME3)

Després tenir el mòdul d'encriptació objectiu clar es procedeix a la seva implementació. L'objectiu d'aquesta tasca és implementar amb codi el mòdul dissenyat a la tasca ME1. Si el disseny de la tasca ME1 és correcte i està ben fet, la implementació en codi no hauria de ser complicada, tot i que pot requerir de cert temps depenent de la destresa del programador amb el llenguatge de descripció de hardware que s'hagi seleccionat. S'estima que per aquesta tasca es requereixen 50 hores.

Aquesta tasca requereix del programari com a recurs material, on es desenvoluparà el codi del mòdul d'encriptació. I el del programador, que implementarà el codi de especificació del mòdul, com a recurs humà.

10.2.4 Proves (P)

10.2.4.1 Proves de mòdul de memòria (P1)

Aquesta tasca té com a objectiu el disseny i la realització de proves sobre el mòdul de memòria principal implementat. El joc de proves que s'usarà ha de ser complet, que tingui en compte tots els casos límit possibles. La realització d'aquest joc de proves ha de demostrar que el mòdul de memòria està ben implementat i assegurar que es pot procedir amb les tasques posteriors sense dubtar del correcte funcionament d'aquest mòdul. De manera que si sorgeixen errors futurs, es pugui confirmar que no sorgeixen d'una mala implementació de la memòria principal. S'estima un total de 10 hores per la realització d'aquesta tasca.

Aquesta tasca té com a recurs material el programari seleccionat prèviament, on es dissenyaran els jocs de prova i el tester com a recurs humà.

10.2.4.2 Proves del mòdul d'encriptació (P2)

Aquesta tasca té com a objectiu el disseny i la realització de proves sobre el mòdul de encriptació en els accessos a memòria principal implementat. El joc de proves que s'usarà ha de ser complet, que tingui en compte tots els casos límit possibles. La realització d'aquest joc de proves ha de demostrar que el mòdul està ben implementat i que no té cap mena d'error. És el que determinarà l'èxit del projecte, si el joc de proves és robust i es complex, demostrarà que la introducció del mòdul ha estat exitosa. S'estima un total de 20 hores per la realització d'aquesta tasca.

Aquesta tasca té com a recurs material el programari seleccionat prèviament, on es dissenyaran els jocs de prova i tester com a recurs humà.

10.2.5 Documentació (D)

10.2.5.1 Documentació de gestió de projecte (D1)

Aquesta tasca té com a objectiu documentar tot el procés de gestió del projecte que s'ha realitzat abans de començar a treballar en el objectiu principal d'aquest i s'ha dividit en 4 parts on es defineixen diferents aspectes del treball.

10.2.5.1.1 Abast (D11)

Aquesta tasca té com a objectiu principal definir l'abast i context del treball. Definint els objectius, requeriments i possibles obstacles. Prèviament a la definició de l'abast es requereix d'una reunió amb el director per tal de sincronitzar-se i acotar conjuntament el projecte. Aquesta tasca requereix també d'una recerca prèvia de l'estat de l'art per tal de estudiar quines tècniques s'han utilitzat fins ara i amb quins entorns s'ha treballat per la introducció de mòduls de encriptació. Aquesta tasca ha suposat un treball de 20 hores.

10.2.5.1.2 Planificació (D12)

Per complir amb els objectius proposats dins del marge de temps del que es disposa és necessari definir i planificar tasques, tenint en compte els possibles obstacles. El objectiu d'aquesta tasca és definir les tasques en les que es dividirà el projecte, les dependències entre elles i la realització d'un diagrama de Gantt per tal de organitzar el temps de treball de forma òptima. S'estima que aquesta tasca requerirà de 20 hores.

Els recursos materials que s'utilitzaran en aquesta tasca és Gantter, un software que serveix per crear diagrames de Gantt.

10.2.5.1.3 Pressupost (D13)

Es realitzarà un pressupost per quantificar el cost del projecte, es farà una partida per cada tasca definida anteriorment, associada als costos de personal per les persones que hi treballen. S'inclouran costos genèrics. Es realitzaran també partides per els imprevistos i obstacles considerats. S'estima que aquesta tasca requerirà de 15 hores.

Aquesta tasca requerirà d'Excel com a recurs material que servirà com a fulla de càlcul per el pressupost.

10.2.5.1.4 Informe de sostenibilitat (D14)

S'analitzarà a partir d'un informe l'impacte mediambiental, social i econòmic del projecte, en concret les fases de planificació i desenvolupament. Es calcula que aquesta tasca requereixi de 5 hores.

10.2.5.2 Memòria (D2)

La memòria final del projecte és el document que descriu tota la feina feta i els mètodes i tècniques emprades. És una tasca que es portarà a terme al llarg del semestre durant el desenvolupament del projecte, però al final del treball se li dedicarà la major quantitat del temps. Es calcula dedicar-li un total de 60 hores.

10.2.5.3 Documentació entorn de treball (D3)

Aquesta documentació és extraoficial, s'utilitzarà com a mètode de comunicació entre el dissenyador i el director. L'objectiu d'aquesta tasca és facilitar la informació de les opcions seleccionades per l'entorn de treball del dissenyador i investigador al director, per tal de que aquest estigui al dia del progrés del investigador i reduir el temps de les reunions, ja que el director disposarà de la informació en el mateix temps en que és desenvolupada per el investigador, donat que serà un document compartit. Aquesta tasca es portarà a terme concurrentment amb les tasques de selecció de l'entorn. Es calcula que aquesta tasca requereixi de 10 hores de manera intermitent durant el procés de selecció de l'entorn.

Els recurs material d'aquesta tasca és el Microsoft Word i OneDrive, que s'usaran per redactar la documentació i per compartir el document entre els dos participants, respectivament. Els recursos humans són el investigador, el dissenyador i el director, el investigador s'encarregarà de redactar el procés de selecció de l'entorn i el director ho utilitzarà com a guia per a seguir el progrés del dissenyador.

10.2.5.4 Presentació (D4)

Un cop la memòria està redactada i presentada s'ha de preparar la lectura del treball de final de grau que es realitzarà davant del tribunal. Aquesta tasca té com a objectiu la preparació de les diapositives i posar en pràctica tècniques de expressió oral i corporal per a fer una presentació correcte. Es calcula que aquesta tasca requerirà de 20 hores.

10.2.6 Reunions de seguiment (RS)

10.2.6.1 Reunió inicial (RS1)

Aquesta tasca té com a objectiu definir l'abast i el context del projecte entre el dissenyador i el director del projecte per tal d'assegurar que ambdós van per el mateix camí i amb els mateixos objectius. A aquesta reunió se li dedicarà un temps de 45 minuts.

10.2.6.2 Reunions de seguiment ordinari (RS2)

El director i el investigador dedicaran 30 minuts de reunions de manera setmanal. Hi ha una reunió programada cada dos setmanes i es deixa oberta la possibilitat per el investigador de sol·licitar una reunió si es troba necessari. Aquesta tasca té com a objectiu mantenir el investigador i el director del projecte a la mateixa pàgina, i que el director pugui mantenir control del progrés que fa el investigador i que el investigador pugui obtenir consells i guies en el desenvolupament del projecte. S'estima un total de 7 reunions.

10.2.6.3 Reunió entorn (RS3)

Una vegada s'ha seleccionat l'entorn i el processador sobre el qual es treballarà durant el projecte, es convocarà una reunió per part del dissenyador per tal de informar al director de la selecció realitzada. Aquesta reunió té un temps estimat de 30 minuts.

10.2.6.4 Reunió disseny del mòdul d'encriptació (RS4)

Per tal de prevenir errors, abans de la implementació del mòdul de encriptació es mostrarà el disseny teòric amb el esquema dibuix del mòdul realitzat per el dissenyador al director del projecte, per tal que aquest últim pugui donar la seva opinió i algun consell o apunt en cas que es trobi necessari. Aquesta reunió té un temps estimat de 60 minuts, 1 hora.

10.2.6.5 Reunió final (RS5)

Un cop acabada la implementació del mòdul d'encriptació i s'hagi comprovat el seu correcte funcionament, es programarà una reunió amb el investigador i el director del projecte per que el director pugui comprovar els resultats i pugui aportar algun comentari. Aquesta reunió també te com a objectiu preparar la presentació oral del treball. Aquesta tasca se li ha assignat una estimació de temps de 1 hora.

ID	Nom	Temps (h)	Dependències	Tipus dependència	Recursos	Rols
E	**Estudi de l'entorn**	**75**	-	-	-	-
E1	Selecció del processador	50	D1	De fi a inici	Ordinador, Microsoft Office	DH, I
E2	Estudi del llenguatge	15	E1	De fi a inici	Ordinador, manual	Pr, I
E3	Selecció i estudi del programari	10	E1	De fi a inici	Ordinador, programari	Pr, I, T
MP	**Mòdul de memòria principal**	**65**	-	-	-	-
MP1	Disseny	40	RS4	De fi a inici	Ordinador, Diagrams.net	DH
MP2	Implementació	25	MP1	De fi a inici	Ordinador, programari	Pr
ME	**Mòdul d'encriptació**	**145**	-	-	-	-
ME1	Disseny de les senyals	50	P1	De fi a inici	Ordinador, Diagrams.net	DH
ME2	Disseny del algorisme	45	ME1	De fi a inici	Ordinador, Diagrams.net	Cr, DH
ME3	Implementació	50	ME2	De fi a inici	Ordinador, programari	Pr
P	**Proves**	**30**	-	-	-	-
P1	Mòdul de memòria principal	10	MP2	De fi a inici	Ordinador, programari	T
P2	Mòdul d'encriptació	20	ME3	De fi a inici	Ordinador, programari	T
D	**Documentació**	**150**	-	-	-	-
D1	Gestió de projecte	60	-	-		
D11	Abast	20	RS1	De fi a inici	Ordinador, Microsoft Office	I
D12	Planificació	20	D11	De fi a inici	Ordinador, Microsoft Office, Gantter	I
D13	Pressupost	15	D12	De fi a inici	Ordinador, Microsoft Office	I
D14	Informe de sostenibilitat	5	D13	De fi a inici	Ordinador, Microsoft Office	I
D2	Memòria	60	-	-	Ordinador, Microsoft Office	I
D3	Entorn de treball	10	E	De inici a inici	Ordinador, Microsoft Office	DP, I, DH
D4	Presentació	20	D2	De fi a inici	Ordinador, Microsoft Office	I
RS	**Reunions de seguiment**	**6,75**	-	-	-	-
RS1	Inicial	0,75	-	-	Ordinador, Google Meets	DP, I
RS2	Ordinàries	3,5	-	-	Ordinador, Google Meets	DP, I
RS3	Entorn de treball	0,5	D3	De fi a inici	Ordinador, Google Meets	DP, I
RS4	Disseny mòdul encriptació	1	ME2	De fi a inici	Ordinador, Google Meets	DP, I
RS5	Final	1	P2	De fi a inici	Ordinador, Google Meets	DP, I
-	**Total**	**471**	-	-	-	-

Taula 6. Taula resum de les tasques del projecte. Les abreviatures de la columna rols corresponen a les assignades a la secció recursos. Font pròpia.

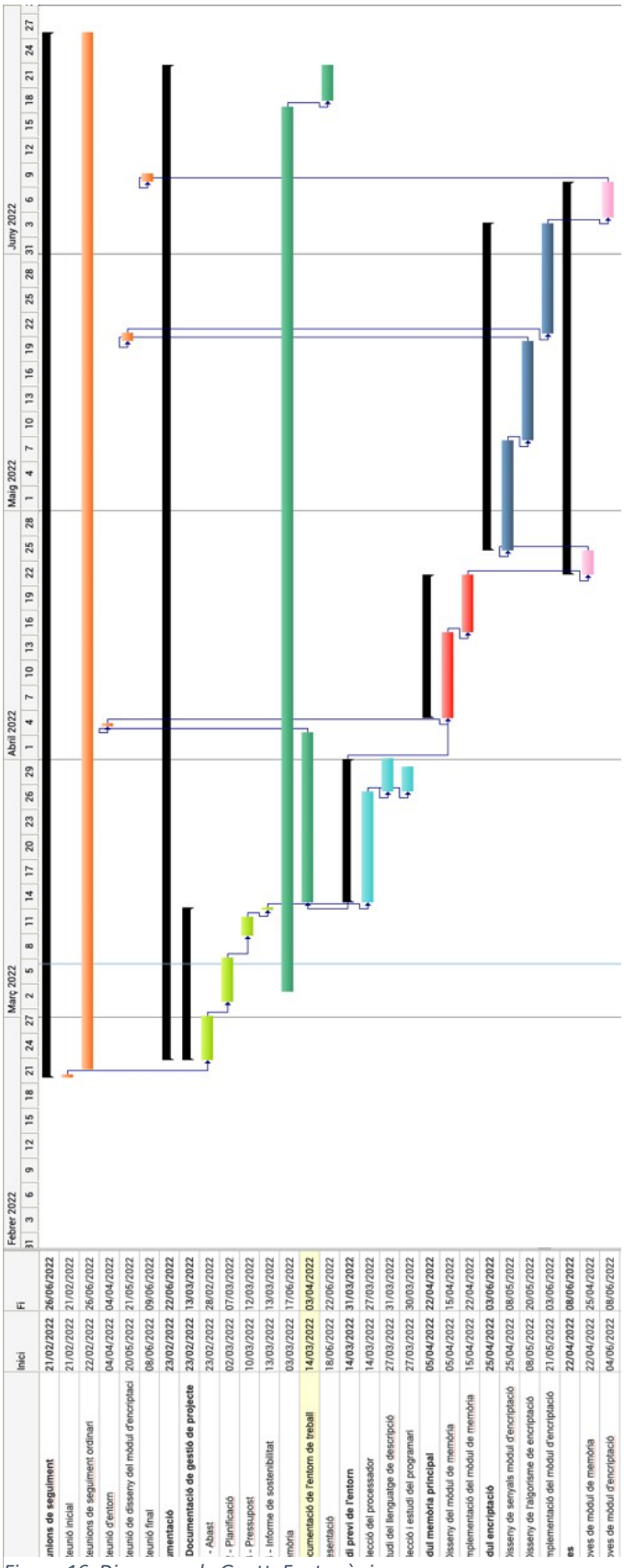

Figura 16. Diagrama de Gantt. Font pròpia.

10.3 METODOLOGIA I SEGUIMENT

10.3.1 Metodologia

Aquest és un projecte complex, on es fàcil que es cometin errors que en un principi passin desapercebuts, però que puguin afectar al resultat final de manera rellevant si no es depuren a temps. Per tant, seguint també amb la metodologia seguida en els projectes de especificació de hardware durant el grau, la metodologia de treball amb què es treballarà durant el desenvolupament del projecte serà incremental. El model incremental té com a objectiu un creixement progressiu de la funcionalitat. El producte va evolucionant en cada una de les entregues previstes fins que arriba a ser el producte desitjat, sent funcional en totes les entregues. Uns dels avantatges d'aquest mètode és la facilitat de provar i depurar el projecte i la simplificació de la gestió de riscos.

10.3.2 Seguiment

Es preveu mantenir un contacte setmanal, fins a dos cops per setmana les primeres setmanes per tal de assegurar que el dissenyador i el director tenen les mateixes idees. L'objectiu és aprofitar la iteració del mètode incremental per tal de poder obtenir *feedback* per part del director a cada fase del projecte. El punt d'inflexió és decidir els límits i objectius de cada incrementació o iteració, tan de temps com de disseny, per tal de mantenir un ritme òptim del desenvolupament del projecte.

10.4 GESTIÓ DE RISCOS

És necessari preveure possibles riscos i obstacles i avaluar els possibles costos que poden comportar durant el desenvolupament del treball. En aquesta secció s'expliquen els obstacles, les tasques alternatives i com afectarien al desenvolupament del treball.

10.4.1 Estimacions de temps errònies

Tot i que el temps requerit per les tasques s'ha aproximat tirant cap amunt, és probable que alguna de les estimacions temporals que s'han realitzat no es compleixi i s'acabi requerint més temps. Per això s'ha deixat un marge de 10 dies abans del dia límit, perquè la prolongació d'una tasca no afecti el temps de dedicació a les posteriors.

10.4.2 Mala selecció del entorn

Es considera el cas en que la selecció del processador o llenguatge no sigui el correcte o que acabi suposant un obstacle, encara que s'ha dotat de suficient temps per a poder fer una bona decisió. Aquest obstacle afectaria com a molt tard en el moment d'implementació del mòdul de memòria. Entre d'altres, l'objectiu de la documentació de la selecció de l'entorn és de deixar registrades les diferents opcions estudiades, en cas que sigui necessari s'hauria de tornar enrere i seleccionar la segona opció. La tasca de disseny de memòria és la que s'hauria de repetir, la qual s'ha estimat que requereix 40 hores, donat que part del raonament del disseny del mòdul de memòria ja s'hauria realitzat en el primer intent, es considera que aquest obstacle suposaria un endarreriment de 20 a 30 hores. S'hauria de tornar a fer una reunió de seguiment del nou entorn seleccionat per tal de mantenir el director del projecte informat, aquest també pot ser d'ajuda en cas de prendre una decisió final en la selecció de l'entorn.

10.4.3 Falta de temps

També pot donar-se el cas que les estimacions de temps per cada tasca sigui correcte però que el temps de dedicació diari sigui massa alt i que hi hagi dies o setmanes a les quals no s'hi pugui arribar. Això també estaria "protegit" per el temps addicional de 10 dies al final del projecte, però es podria solucionar també augmentat els temps de dedicació de les setmanes posteriors per tal de mantenir el ritme adient a la planificació realitzada.

11 GESTIÓ ECONÒMICA

A continuació es fa un estudi dels costos del projecte. Tenint en compte els recursos materials, personal i inconvenients especificats en la secció prèvia i un pla de contingència per a cobrir imprevistos.

11.1 COSTOS RECURSOS HUMANS

En la taula 6, queden especificats els rols que participen en el projecte identificats en la planificació temporal i el cost per hora de cada rol, s'han obtingut les dades a través de "talent.com", una pàgina web que calcula el sou mitjà d'una posició de feina a partir de diferents ofertes d'Internet.

Rol	Cost per hora (€/h)
Director projecte	35,9
Investigador	12,92
Dissenyador de hardware	15,38
Programador	13,85
Tester	12,31
Criptògraf	18,59

Taula 7. Costos de personal amb dades obtingudes de https://es.talent.com/. Font pròpia.

A partir de la taula 7 i la distribució de tasques que s'ha realitzat en la planificació temporal obtenim la taula 8 amb les partides per tasca.

ID	Nom	Temps (h)	Rols	Cost(€/h)	Cost SS(€/h)
E	**Estudi de l'entorn**	**75**	**-**	**2.207,35**	**2.869,56**
E1	Selecció del processador	50	DH, I	1.415,00	1.839,50
E2	Estudi del llenguatge	15	Pr, I	401,55	522,02
E3	Selecció i estudi del programari	10	Pr, I, T	390,80	508,04
MP	**Mòdul de memòria principal**	**65**	**-**	**961,45**	**1.249,89**
MP1	Disseny	40	DH	615,20	799,76
MP2	Implementació	25	Pr	346,25	450,13
ME	**Mòdul d'encriptació**	**145**	**-**	**2.990,15**	**3.887,20**
ME1	Disseny de les senyals	50	DH	769,00	999,70
ME2	Disseny del algorisme	45	Cr, DH	1.528,65	1.987,25
ME3	Implementació	50	Pr	692,50	900,25
P	**Proves**	**30**	**-**	**369,30**	**480,09**
P1	Mòdul de memòria principal	10	T	123,10	160,03
P2	Mòdul d'encriptació	20	T	246,20	320,06
D	**Documentació**	**150**	**-**	**2.450,80**	**3.186,04**
D1	Gestió de projecte	60		-	-
D11	Abast	20	I	258,40	335,92
D12	Planificació	20	I	258,40	335,92
D13	Pressupost	15	I	193,80	251,94
D14	Informe de sostenibilitat	5	I	64,60	83,98
D2	Memòria	60	I	775,20	1.007,76
D3	Entorn de treball	10	DP, I, DH	642,00	834,60
D4	Presentació	20	I	258,40	335,92
RS	**Reunions de seguiment**	**6,75**	**-**	**329,54**	**428,40**
RS1	Inicial	0,75	DP, I	36,62	47,60
RS2	Ordinàries	3,5	DP, I	170,87	222,13
RS3	Entorn de treball	0,5	DP, I	24,41	31,73
RS4	Disseny mòdul encriptació	1	DP, I	48,82	63,47
RS5	Final	1	DP, I	48,82	63,47
-	**Total**	**471**	**-**	**9.308,59**	**12.101,16**

Taula 8. Taula de partides per tasca. Cost SS es el cost de cada tasca tenint en compte la seguretat social. Rols: DP - Director de projecte, I - Investigador, DH - Dissenyador Hardware, Pr - Programador, T - Tester, Cr - Criptògraf. Font pròpia.

En la taula 8 es poden observar els costos de personal incloent també els costos de la seguretat social que s'estimen multiplicant per 1,3 el sou brut. El cost de personal del projecte és de 12.102€ aproximadament.

11.2 COSTOS GENÈRICS

Per els costos genèrics es consideren els costos en els recursos materials en el document anterior. Dividirem aquests costos en hardware, software i costos indirectes.

11.2.1 Costos Software

Recurs	Preu	Unitats	Vida útil	Amortització
Microsoft Office	0 €	1	-	0 €
Programari especificació hardware	0 €	1	-	0 €
Gantter	0 €	1	-	0 €
Google Meets	0 €	1	-	0 €
Diagrams.net	0 €	1	-	0 €
TOTAL	0 €			0 €

Taula 9. Pressupost de eines de software. Font pròpia.

Tot i no tenir encara un programari d'especificació de hardware escollit, una de les premisses és que aquest sigui gratuït, és per això que ja s'inclou al cost a la taula 9. Totes les eines utilitzades o bé són gratuïtes o bé no requereixen del servei de pagament per a la tasca a realitzar, per això els costos en software son 0 euros.

11.2.2 Costos Hardware

Com a eina de hardware només tenim l'ordinador, el qual s'utilitza en totes les hores de feina del treball. Considerant que l'any té 220 dies hàbils i cada dia té 8 hores laborables el cost per hora del dispositiu és Cost Dispositiu/(Vida Útil $*$ 220 $*$ 8). Si això ho multipliquem per les hores d'us obtenim l'amortització del producte, el qual es pot observar a la taula 10.

Recurs	Preu	Unitats	Vida útil (anys)	Hores d'ús	Amortització
Ordinador	1.000 €	1	4	450	64 €
TOTAL	1.000 €	-	-	-	64 €

Taula 10. Costos de les eines hardware. Font pròpia.

11.2.3 Costos indirectes

En aquesta secció es tenen en compte altres costos que no queden inclosos en els altres grups però que representen una despesa en el projecte, es pot observar a la taula 11.

En el cas de l'electricitat es calcula que el ordinador consumeix 0,75 kWh en una hora de treball i sumant-li el consum de llum i de la pantalla s'arrodoneix a 1,25 kWh, multiplicant per el temps d'ús (450 hores) resulta en 563 kWh.

Recurs	Preu	Unitats	Cost
Electricitat	0,38 €/kWh	563 kWh	213,94 €
Internet	30€/mes	4 mesos	120,00 €
Total			**333,94 €**

Taula 11. Costos indirectes. Font pròpia.

11.3 CONTINGÈNCIA

És important afegir un sobre cost per tal de cobrir possibles imprevistos que no es poden tenir en compte. Tot i que el projecte se situa en l'àmbit informàtic, no es considera que les eines tinguin probabilitats altes de error, per això s'ha decidit fixar un 10% de sobre cost. En la taula 12 podem observar la contingència aplicada als diferents costos i la total del projecte.

Tipus	Cost	Contingència
Personal	12.106,16 €	1.210,62 €
Genèrics	429,94 €	42,99 €
Total	**12.536,10 €**	**1.253,61 €**

Taula 12. Taula contingència del 10% per tipus de cost. Font pròpia.

11.4 IMPREVISTOS

Tenint en compte els possibles obstacles considerats a la planificació temporal, a continuació es calcula el cost que aquests comportarien i la probabilitat de que succeeixin. A la taula 13 es mostra el cost.

1. Mala selecció de l'entorn - Es considera que si es fa una mala selecció de l'entorn pot suposar un endarreriment d'unes 25 hores que s'haurien de recuperar per part del investigador, el dissenyador de hardware i el programador, en total això equivaldria a 1053,75€ de despesa en personal. El risc de que això succeeixi és baix, es considera una probabilitat del 10%.
2. Augment de temps de treball - És possible que es necessiti augmentar el temps de dedicació en alguna de les tasques degut a que l'estimació del temps necessari per la tasca feta a la planificació temporal no s'ajusti a la realitat. Per això s'ha afegit un marge de 10 dies extra de feina, el qual equivaldria a 35 hores de feina. La mitja de sous és de 18€/h. Tenint en compte que la mitjana de rols per tasca és de 1,5 és probable que més d'un rol hagi de participar a la vegada per resoldre el problema, per tant s'augmenta el cost de l'hora a 27€/h. Aquest inconvenient per tant suposaria una despesa de 35h x 27€/h que equivaldria a 945€. És molt plausible que això succeeixi donada la poca experiència del investigador en planificacions de projectes d'aquestes dimensions, per tant s'estima un risc del 20%.

Imprevist	Cost	Risc	Cost total
Mala selecció de l'entron	1.053,75 €	10%	105,38 €
Augment del temps de treball	945,00 €	20%	189,00 €
Total	**1.998,75 €**		**294,38 €**

Taula 13. Taula de despeses per imprevistos. Font pròpia.

11.5 COST TOTAL

A partir dels diferents tipus de cost estudiats a la taula 14 s'observa el cost final estimat per el projecte.

Tipus	Cost
Personal	12.106,16 €
Genèric	429,94 €
Contingència	1.253,61 €
Imprevistos	294,38 €
TOTAL	**14.084,09 €**

Taula 14. Pressupost final del projecte. Font pròpia.

11.6 CONTROL DE GESTIÓ

Un cop tenim el pressupost fixat, en aquest apartat es defineixen els mecanismes de control que utilitzarem per comparar i avaluar les desviacions, si existeixen, entre el pressupost i els costos reals del projecte un cop acabat completament. Els costos reals que aniran incurrent s'aniran reportant al sistema de costos.

A continuació es presenten els descriptors numèrics per el control:

1. Desviació de la mà d'obra en cost:

(cost estimat – cost real) * hores reals

2. Desviació de la mà d'obra en tasques:

(hores estimades – hores reals)*cost real

3. Desviació total del cost de personal:

cost estimat – cost total

4. Desviació de recursos en preu (Software, Hardware i costos indirectes):

(cost estimat – cost real)

5. Desviació total del cost d'imprevistos:

(cost estimat – cost real)

6. Desviació d'hores per tasca:

temps estimat tasca – temps real tasca

7. Desviació total d'hores:

temps estimat – temps total

12 CANVIS PLANIFICACIÓ

12.1 JUSTIFICACIÓ

Una de les tasques principals del projecte és l'estudi de l'entorn. Aquesta tasca es basa en la selecció del processador, l'estudi del llenguatge de descripció de hardware i la selecció i estudi del programari amb el que es treballarà. Per a la realització d'aquesta tasca es calculava que es necessitarien 15 dies de feina. Finalment, aquesta tasca s'ha demorat 10 dies més de feina, 25 dies en total.

L'estudi del llenguatge i la selecció del programari van ser força ràpides per què s'han pogut utilitzar uns amb els que el dissenyador ja hi està familiaritzat. La selecció del processador es va portar a terme dins del temps estimat en un principi. El problema ha aparegut en la instal·lació d'aquest.

El processador es va seleccionar a través del procés explicat en la secció d'anàlisi d'alternatives dins aquest document. Un cop seleccionat es va haver de instal·lar el processador i totes les eines que requereix. Semblava una tasca senzilla, ja que un dels punts a favor del processador seleccionat era la claredat de la documentació, però es va acabar demorant més de l'esperat.

El principal objectiu abans de començar a dissenyar components propis era intentar fer accessos a la memòria interna del processador que ja ve implementada, per tal d'entendre el funcionament de les senyals en els accessos a memòria. Aquesta tasca no es va tenir en compte en la planificació inicial, el dissenyador va calcular que seria ràpid i no afectaria la planificació, però es va acabar allargant. El procés de prova dels accessos a memòria que es va fer és el següent:

El primer problema és el mètode amb el qual el codi a executar és inserta dins la memòria del processador. Per escriure codi que sigui executat pel processador hi ha dos opcions:

1. La primera opció (no oficial) és modificar el document amb el codi màquina d'inicialització del processador. Aquesta va ser la primera opció que es va provar per tal d'evitar instal·lar Linux o una màquina virtual. El punt a favor és que és molt més senzill i clar fer el codi d'accés a memòria en codi maquina que en C, ja que el codi en llenguatge màquina seria el següent:
    a. addi reg1, reg0, 3
    b. addi reg2, reg0, 8
    c. store 0(reg2), reg1

    El qual escriu un 3 en la posició 8 de memòria. El problema és que és molt complicat afegir codi màquina nou en un d'antic, ja que la dificultat de comprensió del codi és alta.

    Per tant al cap d'uns dies de intentar-ho es va procedir a la segona opció, instal·lant Linux.

2. La segona opció era la opció recomanada per el dissenyador del processador, però que requereix de més temps, ja que s'ha d'instal·lar una maquina virtual amb Linux i les eines proporcionades per el dissenyador del processador. Aquestes eines tenen la funció de traduir codi de C a codi maquina. Això fa que la tasca sigui més senzilla un cop tens tot el programari instal·lat. Tot i això, malauradament no es va aconseguir fer un codi que fes accessos a memòria de manera correcte al processador. Es va contactar amb el

dissenyador del processador directament per sol·licitar ajuda sobre el tema, però es va decidir deixar el tema a part i començar a dissenyar els mòduls d'encriptació i desencriptació i el de la memòria externa.

12.2 PLANIFICACIÓ DEFINITIVA

Degut a l'estancament amb els accessos a memòria del processador s'ha canviat la planificació del projecte. En la primera planificació, primer es dissenyava el mòdul de memòria, es feien proves d'accés des del processador a la memòria externa, i un cop es comprovés el correcte funcionament de la memòria externa, es començava a dissenyar el mòdul d'encriptació. Per evitar tornar a estancar-nos amb els accessos a memòria des del processador i començar a avançar feina, s'ha decidit dissenyar i implementar primer el mòdul d'encriptació, després el mòdul de desencriptació, comprovar el correcte funcionament d'aquest mòdul i finalment connectar això a una memòria externa. Un cop tot el conjunt funcioni, es provarà de connectar al processador.

En la taula 15, podem observar els canvis en la planificació temporal respecte la planificació inicial i la planificació definitiva.

	Durada	
Tasca	Plan. Inicial	Plan. Definitiva
EE	14/3 - 31/3	14/3 - 11/4
MP	5/4 - 22/4	25/5 - 30/5
ME	25/4 - 3/6	12/4 - 13/5
CP	-	31/5 - 10/6

Taula 15. Dates inici i final de les tasques en cada planificació (inicial i definitiva). EE = Estudi de l'entorn, MP = mòdul memòria principal, ME = mòdul d'encriptació, CP = connexió amb processador. Font pròpia.

A la taula 15 es veuen els canvis ja mencionats com els 10 dies d'afegit en la tasca de l'estudi de l'entorn, el canvi d'ordre entre les tasques de memòria principal i el mòdul de encriptació i la nova tasca afegida on es tractarà la connexió del mòdul d'encriptació i memòria amb el processador, que abans no es contemplava.

A més a més, s'ha reduït la duració de la tasca de disseny i implementació de la memòria principal de 17 a 6 dies, això és degut a que es calcula que sigui una tasca més senzilla de l'esperat, ja que es podria utilitzar com a base la memòria interna que ja incorpora el processador seleccionat.

S'observa també una escletxa de 12 dies entre la tasca de disseny del mòdul de encriptació i la tasca de disseny del mòdul de memòria principal. Aquesta escletxa es dedicarà a les proves de funcionament del mòdul de encriptació i a la redacció de la documentació per a la fita de seguiment, documentació que no es va tenir en compte en la planificació inicial.

12.3 OBJECTIUS

Com es pot observar, això no afecta desenvolupament del projecte, només s'ha prioritzat la creació del mòdul d'encriptació i memòria per sobre de la connectivitat amb el processador. L'objectiu del projecte segueix sent que aquest mòdul es pugui connectar al processador. De moment, el que es farà serà que el mòdul creat compleixi amb el protocol que utilitza el processador seleccionat per a connectar-se amb memòries externes (en aquest cas Wishbone), un cop compleixi el protocol i funcioni correctament, s'intentarà connectar el mòdul al processador seleccionat.

12.4 AFECTACIÓ EN COSTOS

Els canvis de planificació no generen cap afectació significativa en costos, ja que el nombre d'hores de feina no varia notòriament en quantitat. El possible augment de pressupost a causa de les variacions en hores de feina de cada actor ja es tenen en compte en el càlcul del pressupost inicial, per tant, no hi ha afectació en costos.

12.5 SITUACIÓ ACTUAL

Seguint la planificació definitiva presentada anteriorment, ara mateix estem en l'escletxa entre el disseny del mòdul d'encriptació i el disseny de la memòria. Aquesta escletxa esta dedicada a redactar aquest document i a testejar el mòdul de encriptació. El mòdul d'encriptació per tant ja està acabat. Amb les proves realitzades fins ara es pot assegurar que funciona correctament, indicant que el disseny ja podria ser definitiu i per tant, podríem començar ja amb el disseny de la memòria.

13 METODOLOGIA I RIGOR

No hi ha motius per a canviar la metodologia del treball, tot i haver canviat la planificació el mètode incremental es segueix aplicant al llarg del projecte. El model incremental té com a objectiu un creixement progressiu de la funcionalitat. El producte va evolucionant en cada una de les proves o entregues previstes fins que arriba a ser el producte desitjat, sent funcional en totes les entregues.

14 SOSTENIBILITAT

En aquesta secció es realitza un estudi de sostenibilitat a nivell social, econòmic i mediambiental del projecte. A continuació, es realitza una autoavaluació per part del autor sobre el seu domini de la competència de sostenibilitat, i seguidament s'analitzen els tres àmbits de la sostenibilitat mencionats prèviament.

14.1 AUTOAVALUACIÓ

Durant tot el procés d'educació es dona molta importància a la sostenibilitat mediambiental i econòmica principalment, i menys a la social. Els coneixements apresos es basen en el sentit comú, mai no s'expliquen mètodes per a reduir l'impacte d'un producte que es dissenyi. Realment no soc capaç de dissenyar un producte i poder confirmar que he observat totes les alternatives, he realitzat diferents mètodes i que he reduït al màxim el seu impacte, tant mediambiental, com social i com econòmic.

Si més no, en el camp en què em sento més còmode és el de sostenibilitat mediambiental, ja que tinc alguns coneixements en enginyeria sostenible, conec les lleis que controlen els impactes ambiental, mètodes per a reduir l'impacte ambiental d'un producte i les tècniques que existeixen per augmentar la vida útil d'un producte, tot i que tot això a un nivell força teòric, i dins de l'àmbit de l'enginyeria industrial, no informàtica. Trobo que la sostenibilitat és molt important i com a enginyer hauria de conèixer molt millor les bases per dissenyar un producte sostenible i tenir experiència en l'àmbit.

En general, crec que a la sostenibilitat econòmica se li dona molta importància i es valora molt a la hora de prendre decisions. A la sostenibilitat mediambiental també se li dona molt de valor però no es prioritza davant dels diners, i l'impacte social del producte passa més desapercebut.

Crec que aquest projecte m'ajudarà a entendre com aplicar les competències de la meva especialitat en un cas real, i aprendre a gestionar un projecte més complex comparat amb els treballs que he fet fins ara.

14.2 ÀMBIT ECONÒMIC

En l'àmbit econòmic del projecte s'ha realitzat ja un pressupost inicial on s'estimen unes despeses de 14.000 € aproximadament. En aquest pressupost s'han tingut en compte els recursos humans, els recursos materials i la electricitat i internet com a despeses indirectes. A més a més d'una contingència del 10% i un sobre cost per imprevistos.

Actualment existeixen processador amb mòduls d'encriptació en els accessos a memòria sota llicència com el de Intel o AMD per l'arquitectura x86, els quals no són d'accés lliure i gratuït. El

que s'aconsegueix amb la implementació d'aquest mòdul en l'arquitectura RISC-V, és que sigui accessible de manera gratuïta per tothom.

14.3 ÀMBIT AMBIENTAL

La realització d'aquest projecte no té un gran impacte ambiental, ja que no s'usa hardware, només s'especifica a través de software. Només l'ús d'electricitat per part de l'ordinador de treball i el materials d'aquest tindran un impacte negatiu per el medi ambient.

Tot i que no s'ha especificat com a objectiu, es podria intentar minimitzar el consum de energia del mòdul en comparació amb els que existeixen per reduir l'impacte al medi i atraure interès dels sectors interessats.

Els sistemes de encriptació per hardware presenten una clara avantatge respecte les solucions per software ja que són molt més eficients. En el cas de encriptació per hardware es disposa de un xip dedicat a la encriptació que pot reduir molt el consum respecte un software que no està dissenyat per cap processador en concret i que comparteix recursos amb els altres processos.

14.4 ÀMBIT SOCIAL

Des de que vaig començar la carrera sempre m'he interessat per el funcionament dels processadors i la comprensió de cada senyal d'aquest. Aquest projecte requereix d'una alta comprensió de totes les senyals d'un processador i modificar-les de forma eficient per a afegir una funcionalitat. Trobo que és un treball que em pot ajudar molt per a projectes futurs on es requereixi introduir un mòdul que implementi una nova funcionalitat en un processador.

En quant al benefici de la societat, com ja mencionat en l'àmbit econòmic, estem aconseguint la introducció d'un mòdul d'encriptació per hardware més accessible a tothom, amb llicència oberta. La solució per hardware és la solució més segura, que potser requereix de més inversió però que pot ser un requeriment per a dades que requereixen molta seguretat. Amb la encriptació per hardware ens protegim dels atacs cibernètics a les dades i es requereix d'un atac físic per obtenir la informació (robatori de la memòria), cosa que facilita el control i la protecció.

15 CONCLUSIÓ

L'objectiu principal d'aquest projecte és el disseny i implementació d'un mòdul d'encriptació en l'accés a memòria en un entorn RISC-V. Es pot dir que s'ha complert la primera part, el disseny i la implementació d'un mòdul d'encriptació, però malauradament no s'ha aconseguit implementar-ho en un processador RISC-V i que funcioni correctament, com ja s'ha explicat a la secció 9.1.

Es podria concloure que la selecció del processador no es va fer de manera correcte, ja que aquest ha comportat moltes complicacions en els accessos a memòria que es podrien haver detectat des d'un principi. De fet, durant la fita de seguiment, es menciona que el projecte es va endarrerir perquè no s'aconseguia fer un accés a la memòria de dades interna, però es va decidir seguir i provar-ho més endavant directament amb la memòria externa. Si la prova d'accés a memòria s'hagués fet des d'un principi abans de seleccionar el processador NEORV32 com a finalista, es podria haver provat l'accés a memòria amb altres processadors (com per exemple amb el REONV, el segon candidat), molt abans de estancar-se i perdre tants dies per provar l'accés a memòria des del NEORV32.

Tot i això, el mòdul d'encriptació dissenyat en aquest projecte és correcte i s'ha assolit una gran part de l'objectiu inicial, l'encriptació de dades en l'accés a memòria. El encriptador final utilitza l'algorisme RC5 de 10 rondes i amb una clau de 64 bits que permet encriptar blocs de 32 bits. A més a més el mòdul conté un desencriptador, que executa l'algorisme invers a l'encriptador, i una memòria de 32 kB. Aquest mòdul compleix amb l'arquitectura de bus *Wishbone*. I per tant pot connectar-se a qualssevol processador que accepti aquest tipus de connexió.

15.1 POSSIBLES MILLORES I OBSERVACIONS

15.1.1 Reducció de la latència

Com s'ha analitzat a la secció 8, el mòdul requereix de 14 cicles per a portar a terme un accés a memòria. D'aquests 14 cicles, 11 es dediquen al procés de desencriptació, on a cada cicle es realitza les operacions de cada ronda de l'algorisme (10 rondes més la ronda d'inicialització), la memòria requereix d'un altre cicle per a accedir a la adreça determinada, i els altres dos cicles són destinats a l'estat de *idle* i l'estat de *ready* del mòdul desencriptador, on s'indica que les dades de sortida ja són correctes.

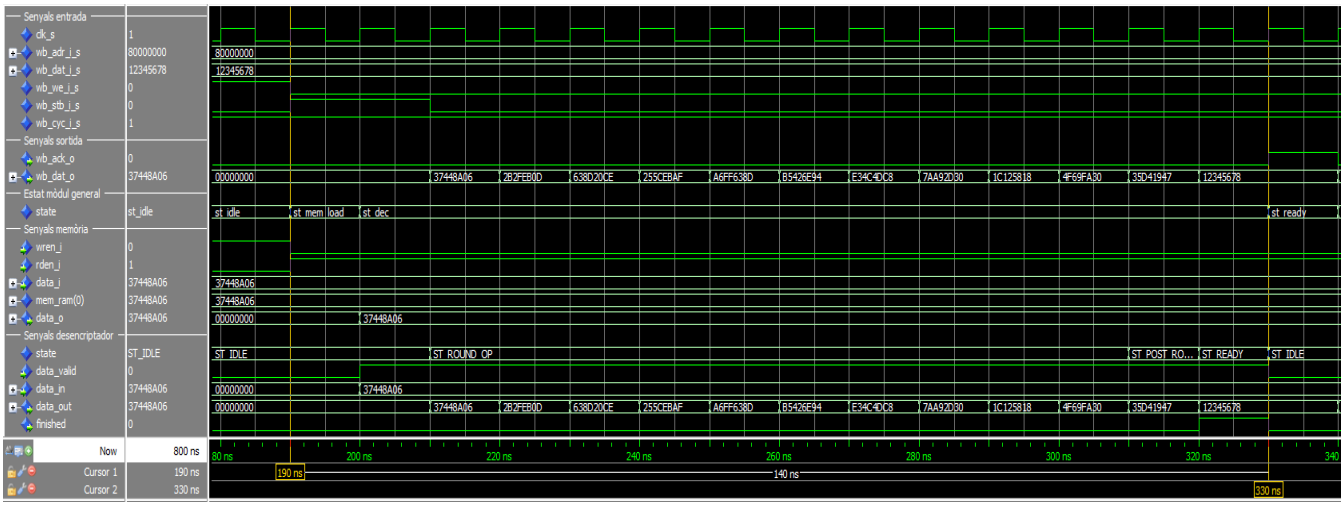

Figura 17. Diagrama de forma d'ona de les senyals de l'operació de lectura. Font pròpia.

A la figura 17 es pot observar que el valor de la sortida del mòdul és correcte durant dos cicles, i no es llegeix fins el segon cicle. És a dir, en el diagrama de la figura 17, el valor de sortida es podria llegir als 320 nanosegons en comptes de llegir-ho als 330.

Per fer això, el que s'hauria de fer es que el mòdul general passés a l'estat de *ready* a la vegada que el mòdul de desencriptació es troba en estat *ready*. Per fer això, la senyal de *finished* del mòdul de desencriptació s'hauria d'activar un cicle abans, a l'estat *st_post_round*. D'aquesta manera es podria reduir 1 cicle la latència de memòria en un accés de lectura.

15.1.2 Connexió processador

En un futur, es podria buscar un altre processador RISC-V que pogués connectar-se a través del protocol de bus *Wishbone* i provar de fer accessos a memòria a través d'aquest.

16 LLEIS I REGULACIONS

16.1 PROPIETAT INTEL·LECTUAL

Tal com indica la normativa del Treball de Fi de Grau de la FIB:

“La propietat industrial i intel·lectual dels TFG de modalitat A està regulada per la normativa aprovada pel Consell de Govern de la UPC, per la qual s'aprova la confidencialitat, responsabilitat patrimonial i propietat industrial i intel·lectual a la UPC.

- Correspondrà a la UPC la titularitat sobre les invencions desenvolupades exclusivament pels estudiants si s’ha desenvolupat en el marc d’una activitat acadèmica que hagi estat dirigida i/o coordinada pel professorat de la UPC.

- En el cas que el desenvolupament de l'obra intel·lectual hagi estat dirigida i/o coordinada pel professorat de la UPC, correspondrà a la UPC la titularitat dels drets d’explotació sobre aquesta obra i l'estudiant i el professor seran considerats coautors de la mateixa.

- En cas d’explotació de l'obra per part de la UPC que li suposi un benefici econòmic, l'autor o conjunt d'autors tindran dret a una participació del 50% dels beneficis nets obtinguts.”

16.2 LLICÈNCIES

En aquest treball s’utilitza codi de tercers, el del processador NEORV32. Aquest processador té una llicència BSD 3-clause la qual indica que es permet la redistribució i l'ús en forma font i binària, amb o sense modificació, sempre que es compleixin les condicions següents [14]:

1. Les redistribucions del codi font han de conservar l'avís de drets d'autor anterior, aquesta llista de condicions i l'exempció de responsabilitat següent.

2. Les redistribucions en forma binària han de reproduir l'avís de drets d'autor anterior, aquesta llista de condicions i l'exempció de responsabilitat següent a la documentació i/o altres materials subministrats amb la distribució.

3. Ni el nom del titular dels drets d'autor ni els noms dels seus col·laboradors no es poden utilitzar per avalar o promocionar productes derivats d'aquest programari sense un permís previ específic per escrit.

“THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.”

17 BIBLIOGRAFIA

[1] A. Cilardo, «Memory Encryption Support for an FPGA-based RISC-V Implementation,» IEEE, 2021.

[2] WikiChip, «Total Memory Encription (TME) - x86,» 17 Diciembre 2017. [En línea]. Available: https://en.wikichip.org/wiki/x86/tme.

[3] WikiChip, «Secure Memory Encryption (SME) -x86,» 7 Mayo 2020. [En línea]. Available: https://en.wikichip.org/wiki/x86/sme.

[4] R. Dekker, «What's the Difference Between VHDL, Verilog, and SystemVerilog?,» 17 9 2014. [En línea]. Available: https://www.electronicdesign.com/resources/whats-the-difference-between/article/21800239/whats-the-difference-between-vhdl-verilog-and-systemverilog.

[5] C. Riley, «RPU,» 11 Septiembre 2020. [En línea]. Available: https://github.com/Domipheus/RPU/commits?author=Domipheus.

[6] J. V. R. Chrisóstomo, «Maestro,» 23 3 2020. [En línea]. Available: https://github.com/Artoriuz/maestro.

[7] S. Nolting, «NeorV32,» 23 Marzo 2022. [En línea]. Available: https://github.com/stnolting/neorv32.

[8] L. Castro, «ReonV,» 7 Octubre 2018. [En línea]. Available: https://github.com/lcbcFoo/ReonV.

[9] R. Swarbrick, «lowRisc/Ibex,» 15 Mayo 2022. [En línea]. Available: https://github.com/lowRISC/ibex.

[10] National Institute of Standards and Technology, «Advanced Encryption Standard,» 26 Noviembre 2001. [En línea]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[11] S. Upadhyay, «Data encryption standard (DES),» 15 Mayo 2022. [En línea]. Available: https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/.

[12] OpenCores, «Wishbone B4,» 2010.

[13] M. Hassan, «On the Off-chip Memory Latency of Real-Time Systems: Is DDR DRAM Really the Best Option?,» 2018. [En línea]. Available: https://www.ece.mcmaster.ca/faculty/hassan/assets/publications/hassan2018off.pdf. [Último acceso: 3 6 2022].

[14] OpenSource, «The 3-Clause BSD License,» [En línea]. Available: https://opensource.org/licenses/BSD-3-Clause.

18 ANNEX

18.1 CODI MÒDUL GENERAL

```
entity mod_gen is
        port(
                clk  : inSTD_LOGIC;        -- system clock
                clr   : in        STD_LOGIC := '0';

                wb_adr_i   : in std_ulogic_vector(31 downto 0); -- address
                wb_dat_i   : in  std_ulogic_vector(31 downto 0); -- write data (store)
                wb_dat_o   : out std_ulogic_vector(31 downto 0); -- read data (load)
                wb_we_i    : in std_ulogic; -- read/write
                wb_sel_i   : in std_ulogic_vector(03 downto 0); -- byte enable
                wb_stb_i   : in std_ulogic; -- strobe
                wb_cyc_i   : in std_ulogic; -- valid cycle
                wb_ack_o   : out  std_ulogic -- transfer acknowledge
                                );
        end entity;
        architecture arch of mod_gen is

        component rc5_enc
        port(
                clk                 : in   STD_LOGIC;         -- system clock
                clr                 : in   STD_LOGIC := '0';
                data_in             : in   std_logic_vector(31 downto 0);
                valid_data          : in   STD_LOGIC;
                data_out            : out STD_LOGIC_VECTOR(31 downto 0);
                finished   : out STD_LOGIC  -- output valid
                );
end component;

        component dec
        port(
          clr: IN STD_LOGIC := '0';
          clk: IN STD_LOGIC;  -- Clock signal
          data_in: IN STD_LOGIC_VECTOR(31 DOWNTO 0);--
          data_valid: IN STD_LOGIC;  -- input is valid
          data_out: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);--
          finished: OUT STD_LOGIC := '0' --answer is ready when '1'
          );
 end component;

 component dmem
 port (
    clk_i  : in  std_ulogic; -- global clock line
    rden_i : in  std_ulogic; -- read enable
    wren_i : in  std_ulogic; -- write enable
    ben_i  : in  std_ulogic_vector(03 downto 0); -- byte write enable
    addr_i : in  std_ulogic_vector(31 downto 0); -- address
    data_i : in  std_ulogic_vector(31 downto 0); -- data in
    data_o : out std_ulogic_vector(31 downto 0); -- data out
    ack_o  : out std_ulogic -- transfer acknowledge
  );
end component;

        TYPE  StateType IS (st_idle, --
                    st_enc, --
                    st_mem_load, --
                           st_mem_store,
                    st_dec,
                           st_ready);

  SIGNAL  state : StateType := ST_IDLE; --store the state in a variable called 'state'
```

```
        signal clk_main: std_logic := '0';
        signal enc_clr: std_logic := '0';
        signal dec_clr: std_logic := '0';

        signal wb_adr_i_s:   STD_uLOGIC_VECTOR(31 downto 0); -- address
        signal wb_dat_i_s:   STD_LOGIC_VECTOR(31 downto 0); -- write data (store)
        signal wb_dat_o_s:   STD_LOGIC_VECTOR(31 downto 0); -- read data (load)
        signal wb_we_i_s:    std_ulogic; -- read/write
        signal wb_sel_i_s:   STD_uLOGIC_VECTOR(03 downto 0); -- byte enable

        signal data_in_dec :STD_LOGIC_VECTOR(31 downto 0);
        signal valid_data_enc: std_logic := '0';
        signal data_valid_dec: std_logic := '0';

        signal wren, rden: std_ulogic;

  signal data_in_mem : STD_uLOGIC_VECTOR(31 downto 0); -- data in
        signal data_out_enc, data_out_dec: STD_LOGIC_VECTOR(31 downto 0);
        signal data_out_mem: STD_uLOGIC_VECTOR(31 downto 0);

        signal enc_ack, dec_ack: std_logic;
        signal mem_ack: std_ulogic;

        begin

        enc: rc5_enc      port    map(clk=>clk,    clr=>clr,    data_in=>wb_dat_i_s,
valid_data => valid_data_enc, data_out =>data_out_enc, finished => enc_ack);

        decr: dec port map(clr=>clr, clk=>clk,   data_in=>data_in_dec, data_valid =>
data_valid_dec , data_out=>data_out_dec, finished => dec_ack);

        mem: dmem port map (clk_i=>clk, rden_i=>rden, wren_i=> wren, ben_i =>
wb_sel_i_s, addr_i => wb_adr_i_s, data_i => data_in_mem, data_o => data_out_mem, ack_o
=> mem_ack);

         wb_adr_i_s<= wb_adr_i;
         wb_dat_i_s<= std_logic_vector(wb_dat_i);
         wb_dat_o<= std_ulogic_vector(data_out_dec);
         wb_sel_i_s<=wb_sel_i;

         wren <= wb_we_i;
         rden <= not wb_we_i;
         data_in_mem<= STD_uLOGIC_VECTOR(data_out_enc);
         data_in_dec<= STD_LOGIC_VECTOR(data_out_mem);

                  --dout ready
        WITH state SELECT
        wb_ack_o<= '1' WHEN ST_READY,
                  '0' WHEN OTHERS;

                 -- valid data encryption
        WITH state SELECT
        valid_data_enc<= '1' WHEN st_enc,
                           '0' WHEN OTHERS;

                 -- valid data decryption
        WITH state SELECT
        data_valid_dec<= '1' WHEN st_dec,
                           '0' WHEN OTHERS;
         --state machine
PROCESS(clr, clk)
   BEGIN
      IF(clr='0') THEN  --start idle
         state<=ST_IDLE;
                 ELSIF(clk'EVENT AND clk='1') THEN
```

```
         CASE state IS
            WHEN ST_IDLE=>
                  IF(wb_cyc_i='1' and wb_stb_i='1' and wb_we_i = '0') THEN
                           state<=st_mem_load;  --if data valid and load
                  ELSIF(wb_cyc_i='1' and wb_stb_i='1' and wb_we_i = '1') THEN
                           state<=st_enc;  --if data valid and store
                  END IF;

            WHEN st_mem_load=>
                  IF (mem_ack = '1') THEN
                           state<=st_dec;
                  END IF;

                  WHEN st_dec=>
                  IF (dec_ack = '1') THEN
                           state<=st_ready;
                  END IF;

            WHEN st_enc=>
                           IF (enc_ack= '1') THEN
                                    state <= st_mem_store;
                           END IF;
                  WHEN st_mem_store =>
                           IF (mem_ack = '1') THEN
                                    state <= st_ready;
                           END IF;
                  WHEN ST_READY=> state <= ST_IDLE;

         END CASE;
      END IF;
END PROCESS;
end architecture;
```

18.2 CODI DEL MÒDUL D'ENCRIPTACIÓ

```
entity rc5_enc is
         port(clk             : in   STD_LOGIC;         -- system clock
         clr                  : in   STD_LOGIC := '0';
         data_in              : in   STD_LOGIC_VECTOR(31 downto 0);
         valid_data           : in   STD_LOGIC;
         data_out             : out STD_LOGIC_VECTOR(31 downto 0);
         finished             : out STD_LOGIC  -- output valid
                  );
end rc5_enc;

architecture Behavioral of rc5_enc is

type exp_key_table is array(0 to 21) of STD_LOGIC_VECTOR(15 downto 0);
type aux_key_table is array (0 to 3) of STD_LOGIC_VECTOR(15 downto 0);

signal w : integer := 16; -- mida de la paraula (dades de 32 bits)
signal r : integer := 10; -- numero de rondes
signal b_int : integer := 8; --keysize en bytes (64 bits)
signal t : integer := 22; -- mida de expanded key table (S).
signal u : integer := 2; -- u = w/8
signal c : integer := 4;
signal P : STD_LOGIC_VECTOR(15 downto 0) := x"b7e1";
signal Q : STD_LOGIC_VECTOR(15 downto 0) := x"9e37";

--signal L : aux_key_table := (x"0100", x"0302", x"0504", x"0706");

signal S : exp_key_table := (x"23a0", x"c2cf", x"05d3", x"9d84", x"66b9", x"bf8d",
x"7065", x"caeb", x"8bb3", x"80e1", x"3094", x"8c69", x"e992", x"7f58", x"ccae",
x"d742", x"7c14", x"cfca", x"5a5f", x"651d", x"df40", x"c569");
```

```
SIGNAL i_cnt: STD_LOGIC_VECTOR(3 DOWNTO 0);  -- round counter
SIGNAL a_reg: STD_LOGIC_VECTOR(15 DOWNTO 0);  --register to store value A
SIGNAL b_reg: STD_LOGIC_VECTOR(15 DOWNTO 0);  --register to store value B
SIGNAL ab_xor: STD_LOGIC_VECTOR(15 DOWNTO 0);  --AB internal signals
SIGNAL ab_rot: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL a_pre: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL ba_xor: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL ba_rot: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL b_pre: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL a: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(15 DOWNTO 0);

--maquina d'estats
  TYPE  StateType IS (ST_IDLE, --
                      ST_PRE_ROUND,
                      ST_ROUND_OP,
                      ST_READY);

  SIGNAL  state : StateType := ST_IDLE;

begin

partA : process(ab_xor, a_reg, b_reg, ab_rot, i_cnt, data_in)--A=((A XOR B)<<<B)+S[2×i];
begin

ab_xor <= (a_reg) XOR (b_reg);

case b_reg(3 DOWNTO 0) is
         WHEN "0001" => ab_rot <= ab_xor(14 DOWNTO 0) & ab_xor(15);
         WHEN "0010" => ab_rot <= ab_xor(13 DOWNTO 0) & ab_xor(15 DOWNTO 14);
         WHEN "0011" => ab_rot <= ab_xor(12 DOWNTO 0) & ab_xor(15 DOWNTO 13);
         WHEN "0100" => ab_rot <= ab_xor(11 DOWNTO 0) & ab_xor(15 DOWNTO 12);
         WHEN "0101" => ab_rot <= ab_xor(10 DOWNTO 0) & ab_xor(15 DOWNTO 11);
         WHEN "0110" => ab_rot <= ab_xor(9 DOWNTO 0) & ab_xor(15 DOWNTO 10);
         WHEN "0111" => ab_rot <= ab_xor(8 DOWNTO 0) & ab_xor(15 DOWNTO 9);
         WHEN "1000" => ab_rot <= ab_xor(7 DOWNTO 0) & ab_xor(15 DOWNTO 8);
         WHEN "1001" => ab_rot <= ab_xor(6 DOWNTO 0) & ab_xor(15 DOWNTO 7);
         WHEN "1010" => ab_rot <= ab_xor(5 DOWNTO 0) & ab_xor(15 DOWNTO 6);
         WHEN "1011" => ab_rot <= ab_xor(4 DOWNTO 0) & ab_xor(15 DOWNTO 5);
         WHEN "1100" => ab_rot <= ab_xor(3 DOWNTO 0) & ab_xor(15 DOWNTO 4);
         WHEN "1101" => ab_rot <= ab_xor(2 DOWNTO 0) & ab_xor(15 DOWNTO 3);
         WHEN "1110" => ab_rot <= ab_xor(1 DOWNTO 0) & ab_xor(15 DOWNTO 2);
         WHEN "1111" => ab_rot <= ab_xor(0) & ab_xor(15 DOWNTO 1);
         WHEN OTHERS => ab_rot <= ab_xor;
END CASE;

--store new a.

a<=ab_rot + S(CONV_INTEGER(i_cnt & '0')); --S[2×i]
a_pre <= data_in(31 DOWNTO 16) + S(0); -- A = A + S[0]. Pre-round

END PROCESS partA;

partB : process(a, ba_xor, ba_rot, i_cnt, data_in)
begin

ba_xor <= b_reg XOR a;




case a(3 DOWNTO 0) is
       WHEN "0001" => ba_rot <= ba_xor(14 DOWNTO 0) & ba_xor(15);
       WHEN "0010" => ba_rot <= ba_xor(13 DOWNTO 0) & ba_xor(15 DOWNTO 14);
```

```
       WHEN "0011" => ba_rot <= ba_xor(12 DOWNTO 0) & ba_xor(15 DOWNTO 13);
       WHEN "0100" => ba_rot <= ba_xor(11 DOWNTO 0) & ba_xor(15 DOWNTO 12);
       WHEN "0101" => ba_rot <= ba_xor(10 DOWNTO 0) & ba_xor(15 DOWNTO 11);
       WHEN "0110" => ba_rot <= ba_xor(9 DOWNTO 0) & ba_xor(15 DOWNTO 10);
       WHEN "0111" => ba_rot <= ba_xor(8 DOWNTO 0) & ba_xor(15 DOWNTO 9);
       WHEN "1000" => ba_rot <= ba_xor(7 DOWNTO 0) & ba_xor(15 DOWNTO 8);
       WHEN "1001" => ba_rot <= ba_xor(6 DOWNTO 0) & ba_xor(15 DOWNTO 7);
       WHEN "1010" => ba_rot <= ba_xor(5 DOWNTO 0) & ba_xor(15 DOWNTO 6);
       WHEN "1011" => ba_rot <= ba_xor(4 DOWNTO 0) & ba_xor(15 DOWNTO 5);
       WHEN "1100" => ba_rot <= ba_xor(3 DOWNTO 0) & ba_xor(15 DOWNTO 4);
       WHEN "1101" => ba_rot <= ba_xor(2 DOWNTO 0) & ba_xor(15 DOWNTO 3);
       WHEN "1110" => ba_rot <= ba_xor(1 DOWNTO 0) & ba_xor(15 DOWNTO 2);
       WHEN "1111" => ba_rot <= ba_xor(0) & ba_xor(15 DOWNTO 1);
       WHEN OTHERS => ba_rot <= ba_xor;
END CASE;

--store new b
b<=ba_rot+S(CONV_INTEGER(i_cnt & '1'));--S[2×i+1]

b_pre <= data_in(15 DOWNTO 0) + S(1);  -- B = B + S[1]

END PROCESS partB;

--output
data_out<=a_reg & b_reg;

--dout ready
WITH state SELECT
finished<= '1' WHEN ST_READY,
         '0' WHEN OTHERS;

-- a_reg
PROCESS(clr, clk)  BEGIN
  IF(clr='0') THEN a_reg <=(OTHERS=>'0');
  ELSIF(clk'EVENT AND clk='1') THEN
           IF(state=ST_PRE_ROUND) THEN
                           a_reg<=a_pre;
           ELSIF(state=ST_ROUND_OP) THEN
                           a_reg<=a; END IF;
  END IF;
END PROCESS;


-- b_reg
PROCESS(clr, clk)  BEGIN
  IF(clr='0') THEN b_reg<=(OTHERS=>'0');
  ELSIF(clk'EVENT AND clk='1') THEN
           IF(state=ST_PRE_ROUND) THEN   b_reg<=b_pre;
           ELSIF(state=ST_ROUND_OP) THEN   b_reg<=b; END IF;
  END IF;
END PROCESS;

--state machine
PROCESS(clr, clk)
   BEGIN
      IF(clr='0') THEN  --start idle
         state<=ST_IDLE;
                           i_cnt<= "0001";
                  ELSIF(clk'EVENT AND clk='1') THEN
         CASE state IS
            WHEN ST_IDLE=> IF(valid_data='1') THEN state<=ST_PRE_ROUND;

         END IF;
            WHEN ST_PRE_ROUND=> state<=ST_ROUND_OP;
            WHEN ST_ROUND_OP=>
                  IF(i_cnt="1010") THEN
```

```
                        state<=ST_READY;
                        i_cnt<= "0001";  --if round, move to ready when done
                ELSE
                        i_cnt<=i_cnt+'1';
                END IF;
        WHEN ST_READY=> state <= ST_IDLE;

        END CASE;
     END IF;
END PROCESS;

end architecture Behavioral;
```

18.3 CODI DEL MÒDUL DE DESENCRIPTACIÓ

```
ENTITY dec IS
 PORT  (
  clr: IN STD_LOGIC := '0';
  clk: IN STD_LOGIC;
  data_in: IN STD_LOGIC_VECTOR(31 DOWNTO 0);--
  data_valid: IN STD_LOGIC;  -- input is valid
  data_out: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);--
  finished: OUT STD_LOGIC := '0'
  );
END dec;

ARCHITECTURE arch OF dec IS

type exp_key_table is array(0 to 21) of STD_LOGIC_VECTOR(15 downto 0);
type aux_key_table is array (0 to 3) of STD_LOGIC_VECTOR(15 downto 0);

signal S : exp_key_table := (x"23a0", x"c2cf", x"05d3", x"9d84", x"66b9", x"bf8d",
x"7065", x"caeb", x"8bb3", x"80e1", x"3094", x"8c69", x"e992", x"7f58", x"ccae",
x"d742", x"7c14", x"cfca", x"5a5f", x"651d", x"df40", x"c569");

  SIGNAL i_cnt: STD_LOGIC_VECTOR(3 DOWNTO 0);

  --register to store value A
  SIGNAL a_reg: STD_LOGIC_VECTOR(15 DOWNTO 0);

  --register to store value B
  SIGNAL b_reg: STD_LOGIC_VECTOR(15 DOWNTO 0);

  SIGNAL ab_sub: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL ab_rot: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL a_pre: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL ba_sub: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL ba_rot: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL b_pre: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL a: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL b: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL a_ini: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL b_ini: STD_LOGIC_VECTOR(15 DOWNTO 0);

  --state machine (4 states)
  TYPE  StateType IS (ST_IDLE, --
                      ST_ROUND_OP,
                      ST_POST_ROUND,
                   ST_READY);

  SIGNAL  state : StateType;
```

```
  begin
    a_ini<=data_in(31 downto 16);
    b_ini<=data_in(15 downto 0);

--start with B

partB : process(a_reg, b, ba_sub, ba_rot, i_cnt)
begin

--pre
ba_sub <= b_reg - S(CONV_INTEGER(i_cnt & '1'));--S[2×i+1]

case a_reg(3 DOWNTO 0) is

WHEN "1111" => ba_rot <= ba_sub(14 DOWNTO 0) & ba_sub(15);
WHEN "1110" => ba_rot <= ba_sub(13 DOWNTO 0) & ba_sub(15 DOWNTO 14);
WHEN "1101" => ba_rot <= ba_sub(12 DOWNTO 0) & ba_sub(15 DOWNTO 13);
WHEN "1100" => ba_rot <= ba_sub(11 DOWNTO 0) & ba_sub(15 DOWNTO 12);
WHEN "1011" => ba_rot <= ba_sub(10 DOWNTO 0) & ba_sub(15 DOWNTO 11);
WHEN "1010" => ba_rot <= ba_sub(9 DOWNTO 0) & ba_sub(15 DOWNTO 10);
WHEN "1001" => ba_rot <= ba_sub(8 DOWNTO 0) & ba_sub(15 DOWNTO 9);
WHEN "1000" => ba_rot <= ba_sub(7 DOWNTO 0) & ba_sub(15 DOWNTO 8);
WHEN "0111" => ba_rot <= ba_sub(6 DOWNTO 0) & ba_sub(15 DOWNTO 7);
WHEN "0110" => ba_rot <= ba_sub(5 DOWNTO 0) & ba_sub(15 DOWNTO 6);
WHEN "0101" => ba_rot <= ba_sub(4 DOWNTO 0) & ba_sub(15 DOWNTO 5);
WHEN "0100" => ba_rot <= ba_sub(3 DOWNTO 0) & ba_sub(15 DOWNTO 4);
WHEN "0011" => ba_rot <= ba_sub(2 DOWNTO 0) & ba_sub(15 DOWNTO 3);
WHEN "0010" => ba_rot <= ba_sub(1 DOWNTO 0) & ba_sub(15 DOWNTO 2);
WHEN "0001" => ba_rot <= ba_sub(0) & ba_sub(15 DOWNTO 1);
WHEN OTHERS => ba_rot <= ba_sub;
END CASE;

--store new b
b<=ba_rot xor a_reg;--S[2×i+1]

--same for b
b_pre <= b_reg - S(1);
END PROCESS;

--PART A
partA : process(ab_sub, a_reg, b, b_reg, ab_rot) --A=((A XOR B)<<<B) + S[2×i];
begin

--pre
ab_sub <= a_reg - S(CONV_INTEGER(i_cnt & '0'));

--rotate
case b(3 DOWNTO 0) is

         WHEN "1111" => ab_rot <= ab_sub(14 DOWNTO 0) & ab_sub(15);
         WHEN "1110" => ab_rot <= ab_sub(13 DOWNTO 0) & ab_sub(15 DOWNTO 14);
         WHEN "1101" => ab_rot <= ab_sub(12 DOWNTO 0) & ab_sub(15 DOWNTO 13);
         WHEN "1100" => ab_rot <= ab_sub(11 DOWNTO 0) & ab_sub(15 DOWNTO 12);
         WHEN "1011" => ab_rot <= ab_sub(10 DOWNTO 0) & ab_sub(15 DOWNTO 11);
         WHEN "1010" => ab_rot <= ab_sub(9 DOWNTO 0) & ab_sub(15 DOWNTO 10);
         WHEN "1001" => ab_rot <= ab_sub(8 DOWNTO 0) & ab_sub(15 DOWNTO 9);
         WHEN "1000" => ab_rot <= ab_sub(7 DOWNTO 0) & ab_sub(15 DOWNTO 8);
         WHEN "0111" => ab_rot <= ab_sub(6 DOWNTO 0) & ab_sub(15 DOWNTO 7);
         WHEN "0110" => ab_rot <= ab_sub(5 DOWNTO 0) & ab_sub(15 DOWNTO 6);
         WHEN "0101" => ab_rot <= ab_sub(4 DOWNTO 0) & ab_sub(15 DOWNTO 5);
         WHEN "0100" => ab_rot <= ab_sub(3 DOWNTO 0) & ab_sub(15 DOWNTO 4);
         WHEN "0011" => ab_rot <= ab_sub(2 DOWNTO 0) & ab_sub(15 DOWNTO 3);
         WHEN "0010" => ab_rot <= ab_sub(1 DOWNTO 0) & ab_sub(15 DOWNTO 2);
         WHEN "0001" => ab_rot <= ab_sub(0) & ab_sub(15 DOWNTO 1);
         WHEN OTHERS => ab_rot <= ab_sub;
END CASE;
```

```
--store new a.
a<=ab_rot xor b; --S[2×i]

a_pre <= a_reg - S(0);
end process;

--output
data_out<=a_reg & b_reg;

--dout ready
WITH state SELECT
finished<= '1' WHEN ST_READY,
           '0' WHEN OTHERS;


-- a_reg.
PROCESS(clr, clk)
BEGIN
  IF(clr='0') THEN a_reg<=(OTHERS=>'0');
  ELSIF(clk'EVENT AND clk='1') THEN
     IF(state=ST_ROUND_OP) THEN
                            a_reg<=a;
           ElSIF(state=ST_POST_ROUND) THEN
                            a_reg<=a_pre;
           ELSIF (state=ST_IDLE) THEN a_reg <= a_ini;
           END IF;
  END IF;
END PROCESS;


-- b_reg
PROCESS(clr, clk)
BEGIN
  IF(clr='0') THEN b_reg<=(OTHERS=>'0');
  ELSIF(clk'EVENT AND clk='1') THEN
           IF(state=ST_ROUND_OP) THEN  b_reg<=b;
           ELSIF(state=ST_POST_ROUND) THEN b_reg<=b_pre;
           ELSIF(state=ST_IDLE) THEN  b_reg <= b_ini;
           END IF;
  END IF;
END PROCESS;


--4bit downcounter (starts at 12 and rolls over at 1)
PROCESS(clr, clk)
BEGIN
  IF(clr='0') THEN --async clr/reset
         i_cnt<="1010";
  ELSIF(clk'EVENT AND clk='1') THEN
           IF(state=ST_IDLE) THEN
                  i_cnt<="1010";
     ELSIF(state=ST_ROUND_OP) THEN
       IF(i_cnt="0000") THEN
         i_cnt<="1010";
       ELSE
         i_cnt<=i_cnt-'1';
       END IF;
     END IF;
   END IF;
END PROCESS;

-- state machine
PROCESS(clr, clk)
BEGIN
      IF(clr='0') THEN  --best to start idle
```

```
         state<=ST_IDLE;
      ELSIF(clk'EVENT AND clk='1') THEN
         CASE state IS
            WHEN ST_IDLE =>
                  IF(data_valid='1') THEN
                           state<=ST_ROUND_OP;  --if data valid change to round
                  END IF;
            WHEN ST_ROUND_OP=>
                  IF(i_cnt="0001") THEN
                           state<=ST_POST_ROUND;
                   END IF;
           WHEN ST_POST_ROUND=> state<=ST_READY;  --if postround change to ready
           WHEN ST_READY=> state <= ST_IDLE;
         END CASE;
      END IF;
END PROCESS;
end arch;
```

18.4 CODI DEL MÒDUL DE MEMÒRIA EXTERN

```
entity dmem is
  port (
    clk_i  : in  std_ulogic; -- global clock line
    rden_i : in  std_ulogic; -- read enable
    wren_i : in  std_ulogic; -- write enable
    ben_i  : in  std_ulogic_vector(03 downto 0); -- byte write enable
    addr_i : in  std_ulogic_vector(31 downto 0); -- address
    data_i : in  std_ulogic_vector(31 downto 0); -- data in
    data_o : out std_ulogic_vector(31 downto 0); -- data out
    ack_o  : out std_ulogic -- transfer acknowledge
  );
end dmem;

architecture arch of dmem is
         type ram is array (0 to 8191) of std_ulogic_vector(31 downto 0);

  -- local signals --
  signal acc_en : std_ulogic;
  signal rdata  : std_ulogic_vector(31 downto 0);
  signal rden   : std_ulogic;
  signal addr   : std_ulogic_vector(28 downto 0);


  signal mem_ram: ram;
  --signal mem_ram_b0 : mem8_t(0 to DMEM_SIZE/4-1);
  --signal mem_ram_b1 : mem8_t(0 to DMEM_SIZE/4-1);
  --signal mem_ram_b2 : mem8_t(0 to DMEM_SIZE/4-1);
  --signal mem_ram_b3 : mem8_t(0 to DMEM_SIZE/4-1);

  -- read data –
Signal mem_ram_b0_rd, mem_ram_b1_rd, mem_ram_b2_rd, mem_ram_b3_rd : std_ulogic_vector
(7 downto 0);

begin

  acc_en <= '1' when addr_i(31 downto 16) = x"F000" else '0';
  addr   <= addr_i(30 downto 2); -- word aligned


  mem_access: process(clk_i)
  begin
    if rising_edge(clk_i) then

      if (acc_en = '1') then
```

```
        if (wren_i = '1') and (ben_i(0) = '1') then -- byte 0
          mem_ram(to_integer(unsigned(addr)))(7 downto 0) <= data_i(07 downto 00);
        else
          mem_ram_b0_rd <= mem_ram(to_integer(unsigned(addr)))(7 downto 0);
        end if;
        if (wren_i = '1') and (ben_i(1) = '1') then -- byte 1
          mem_ram(to_integer(unsigned(addr)))(15 downto 8) <= data_i(15 downto 08);
        else
          mem_ram_b1_rd <= mem_ram(to_integer(unsigned(addr)))(15 downto 8);
        end if;
        if (wren_i = '1') and (ben_i(2) = '1') then -- byte 2
          mem_ram(to_integer(unsigned(addr)))(23 downto 16) <= data_i(23 downto 16);
        else
          mem_ram_b2_rd <= mem_ram(to_integer(unsigned(addr)))(23 downto 16);
        end if;
        if (wren_i = '1') and (ben_i(3) = '1') then -- byte 3
          mem_ram(to_integer(unsigned(addr)))(31 downto 24) <= data_i(31 downto 24);
        else
          mem_ram_b3_rd <= mem_ram(to_integer(unsigned(addr)))(31 downto 24);
        end if;
      end if;
    end if;
  end process mem_access;


 bus_feedback: process(clk_i)
  begin
    if rising_edge(clk_i) then
      rden  <= acc_en and rden_i;
      ack_o <= acc_en and (rden_i or wren_i);
    end if;
  end process bus_feedback;

  -- pack --
  rdata <= mem_ram_b3_rd & mem_ram_b2_rd & mem_ram_b1_rd & mem_ram_b0_rd;

  -- output gate --
  data_o <= rdata when (rden = '1') else (others => '0');

end arch;
```