UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA TÈCNICA SUPERIOR ENGINYERIA INDUSTRIAL VALÈNCIA

TREBALL FINAL DE MASTER EN ENGINYERÍA INDUSTRIAL

DISSENY I IMPLEMENTACIÓ D'UN ARBRE DE CODIFICACIÓ HUFFMAN EN TEMPS REAL EN FPGA PER A L'OPTIMITZACIÓ DE LA COMPRESSIÓ DE DADES DEL PLA D'ENERGIA DE L'EXPERIMENT NEXT

AUTOR: Guillem Giménez Aparisi

TUTOR: Raúl Esteve Bosch

Selecció

Curs Acadèmic: 2019-20

Documents continguts al TFM

Índex de la Memòria

Índex del Pressupost

Índex de l'Annex 1

Índex de l'Annex 2

Índex de l'Annex 3

Índex de l'Annex 4

DISSENY I IMPLEMENTACIÓ D'UN ARBRE DE CODIFICACIÓ HUFFMAN EN TEMPS REAL EN FPGA PER A L'OPTIMITZACIÓ DE LA COMPRESSIÓ DE DADES DEL PLA D'ENERGIA DE L'EXPERIMENT NEXT

Document núm. 1: Resum

RESUM

El projecte consisteix en el desenvolupament d'un circuit digital per a l'obtenció en temps real de l'arbre de codis Huffman per a la compressió de dades del pla d'energia de l'experiment NEXT. L'objectiu és l'optimització de la compressió de dades ja implementada que utilitzen aquests codis, però que s'han obtingut mitjançant simulació i no estan optimitzats per a l'activitat del detector que usa diferents fonts radioactives i que pot generar al llarg del temps diferents topologies de senyal. L'optimització de la compressió a través de l'ús de codis Huffman que s'adapten a l'activitat del detector suposa una menor taxa en la transmissió de la informació d'interès del detector cap al sistema de processat offline de dades. El projecte implica l'estudi dels codi Huffman i l'obtenció de l'arbre de codis que s'usen posteriorment per a fer la compressió, així com la implementació del circuit que obté l'arbre de codis en temps real en FPGA per a ser integrat en els mòduls ja existents del sistema d'adquisició de dades. Aquest procés implica també una fase de verificació del disseny.

Paraules clau: Circuit digital; Compressió de dades; Codis Huffman; Verilog; FPGA.

RESUMEN

El proyecto consiste en el desarrollo de un circuito digital para la obtención en tiempo real del árbol de códigos Huffman para la compresión de datos del plano de energía del experimento NEXT. El objetivo es la optimización de la compresión de datos ya implementada que utilizan estos códigos, pero que se han obtenido mediante simulación y no están optimizados para la actividad del detector que usa diferentes fuentes radiactivas y que puede generar a lo largo del tiempo diferentes topologías de señal. La optimización de la compresión a través del uso de códigos Huffman que se adaptan a la actividad del detector supone una menor tasa en la transmisión de la información de interés del detector hacia el sistema de procesado offline de datos. El proyecto implica el estudio de los códigos Huffman y la obtención del árbol de códigos que se usan posteriormente para hacer la compresión, así como la implementación del circuito que obtiene el árbol de códigos en tiempo real en FPGA para ser integrado en los módulos ya existentes del sistema de adquisición de datos. Este proceso implica también una fase de verificación del diseño.

Palabras clave: Circuito digital; Compresión de datos; Códigos Huffman; Verilog; FPGA.

ABSTRACT

The project consists in the development of a real time digital circuit that generates dynamically the Huffman coding tree for the data compression of the energy plane of the NEXT experiment. The objective is the optimisation of the data compression already implemented that uses this type of codes, but that have been obtained by means of simulation. These codes are not optimised for the activity of the detector because it uses different radioactive sources that generates different signal topologies. The online optimisation of the Huffman codes leads to data throughput reduction between the data acquisition and the offline processing system. The project involves the evaluation of the Huffman codes and the obtaining of the coding tree. These codes are later used to do the data compression. It also involves the implementation of the circuit that obtains the coding tree in real time in an FPGA in order to be integrated in the already existent modules of the data acquisition system. This last process involves a phase of verification of the design.

Key words: Digital circuit; Data compression; Huffman Coding; Verilog; FPGA.

DISSENY I IMPLEMENTACIÓ D'UN ARBRE DE CODIFICACIÓ HUFFMAN EN TEMPS REAL EN FPGA PER A L'OPTIMITZACIÓ DE LA COMPRESSIÓ DE DADES DEL PLA D'ENERGIA DE L'EXPERIMENT NEXT

Document núm. 2: Memòria

1. OBJECTE DEL PROJECTE

L'objectiu és la programació i verificació d'un circuit electrònic digital mitjançant llenguatge Verilog que permet la creació d'una taula dinàmica amb codis Huffman que permeten la compressió de dades d'un detector de física de partícules. Aquestes dades provenen de l'experiment de física NEXT, ubicat a Laboratori Soterrani de Canfranc. La compressió de dades serà realitzada mitjançant la codificació de Huffman, la qual construeix una estructura en forma d'arbre segons la probabilitat que té una determinada dada en aparèixer i proporciona una codificació per a cadascun dels nodes d'aquest arbre. La codificació creada té menys bits quan una dada és més freqüent.

El programa dissenyat es compondrà de tres parts diferenciables: l'adquisició de dades, la construcció de l'arbre segons l'algorisme de Huffman i la codificació d'aquest. El conjunt del projecte serà simulat tenint en compte diversos casos per tal de verificar el seu correcte funcionament. Amb açò, serà possible comprimir en gran mesura les dades per a la seua posterior transmissió.

El projecte podrà formar part de l'actual sistema de compressió de dades de l'experiment ja present a una FPGA de la marca Xilinx. Serà també important optimitzar la programació per reduir la quantitat de recursos utilitzats, ja que la targeta actual està a la meitat de la seua capacitat.

Per tant, els objectius concrets del projecte són:

- L'estudi dels codi Huffman i l'obtenció de l'arbre de codis que s'utilitzaran posteriorment per a fer la compressió mitjançant l'ús del llenguatge de programació Python.
- Definició del funcionament del circuit.
- Implementació del circuit en Verilog per a una FPGA de Xilinx.
- Estimació dels recursos utilitzats en FPGA per a l'estudi de la viabilitat de la seua integració en els mòduls ja existents del sistema d'adquisició de dades.
- Verificació del disseny.

2. MOTIVACIÓ

És innegable la importància de l'electrònica al món actual. No sols és ben present al nostre dia a dia, sinó que aquesta presència va en augment cada any que passa amb l'aparició de nous productes "intel·ligents" al mercat [1] [2]. Pel que respecta al món de la indústria, l'entrada en escena de, per exemple, sistemes d'automatització a les fàbriques i les necessitats de connexió amb altres parts del món fan de l'electrònica una necessitat vital per dur a terme les activitats quotidianes.

Es pot extraure d'açò que no sols es necessària per crear nous aparells que ens faciliten la vida, les necessitats de transmissió de la informació cada volta són majors i els aparells electrònics tenen capacitats majors per adaptar-se. No obstant, a l'ecosistema de les transmissions també s'inclouen les infraestructures que les permeten. És ací on radica la importància de la compressió, la millora i abaratiment de les memòries actuals permet emmagatzemar gran quantitat d'informació, però les infraestructures no estan preparades [3].

La millora en la velocitat de transmissió de les dades implicaria directament una reducció de la taxa de transmissió i, per tant, dels temps morts. El programa d'adquisició conté dos buffers en els que s'emmagatzema i s'empaqueten les dades per ser enviades. Si aquests estan plens quan es produeix un esdeveniment a l'experiment, la informació es perd. Aquest temps en què s'està processant i enviant les dades i el sistema de compressió no admet més esdeveniments és el temps mort.

3. JUSTIFICACIÓ

La creixent quantitat d'informació que s'ha d'enviar amb les evolucions que està tenint l'experiment NEXT fan necessària la compressió de les dades per dur a terme les transmissions pertinents. Els mòduls de compressió ja han sigut desenvolupats en un treball anterior, però emprant una taula fixa. En aquest treball es pretén millorar aquesta compressió adaptant-la a les dades rebudes mitjançant una taula dinàmica, millorant així la transmissió de les dades on han de ser posteriorment processades.

Una optimització de la compressió mitjançant una taula dinàmica permetria la reducció del temps mort (temps en el que l'experiment no pot gestionar nous esdeveniments); d'aquesta manera podria augmentar la quantitat d'esdeveniments que s'envien, i per tant, derivat d'açò, l'augment de la freqüència de funcionament de l'experiment, mantenint un temps mort acceptable per a l'experiment. Cal destacar que no és imperatiu reduir el temps mort a un mínim; hi ha un màxim assumible i pot convindre mantindre's en aquest si escau. A NEXT està fixat un temps mort màxim del 2 %.

El perquè de la taula dinàmica ve donat pels diferents tipus de dades que hi poden aplegar. Per dur a terme l'experiment s'han de realitzar diversos calibratges amb diferents fonts radioactives. Cadascuna d'aquestes pot proporcionar diferents tipus de senyal amb diverses topologies. I és per això que una taula estàtica no seria capaç d'adaptar-se a tots els casos que poden tenir-hi lloc.

4. INTRODUCCIÓ

4.1. L'experiment NEXT

L'experiment NEXT (Neutrino Experiment with a Xenon Time Projection Chamber), ubicat al Laboratori Soterrani de Canfranc, tracta de demostrar que el neutrí és la seua pròpia antipartícula, un fet amb grans implicacions en el món de la física. Una forma de demostrar açò seria mitjançant l'observació d'una desintegració doble beta sense neutrins. Sense entrar en molt de detall, en la desintegració doble beta dos neutrons d'un mateix nucli atòmic passen a ser dos protons, emetent-se en el procés dos electrons i dos antineutrins. L'aparició d'una desintegració com aquesta sense neutrins implicaria que aquests han sigut aniquilats mútuament, fet que sols seria possible si l'antineutrí fora la seua pròpia antipartícula, passant a ser considerat una partícula de Majorana [4] [5] [6] [7].

Per tal d'aconseguir açò, s'empra una cambra de projecció temporal amb Xenó a alta pressió, un esquema de la qual es pot veure a la Figura 1. A l'interior es col·loca un càtode un poc separat d'un dels costats, per suavitzar la caiguda de tensió d'aquest a terra, i un ànode a l'altre costat. Entre ells es produeix un fort camp elèctric que accelera els electrons, els quals al seu torn ionitzen el gas. On es troba l'ànode, hi ha un pla de seguiment amb fotomultiplicadors de silici (SiPM) que és capaç de fer un seguiment de les partícules a l'interior en els tres eixos (x, y i z) al mateix temps [5] [8].

A la cambra se li suma l'amplificació per electroluminescència a l'ànode, proporcionant una gran quantitat de fotons que seran posteriorment captats per fotomultiplicadors (PMT), presentats a la Figura 2. És aquesta agrupació de fotomultiplicadors l'anomenat pla d'energia. Els senyals detectats a aquest i posteriorment processats són les entrades del circuit digital programat al present projecte. La combinació de tot açò dóna com a resultat un sistema capaç de mesurar amb molta resolució l'energia de l'esdeveniment que, en cas de ser desintegració doble beta sense neutrins, l'energia alliberada durant aquest serà pràcticament constant [5] [8].

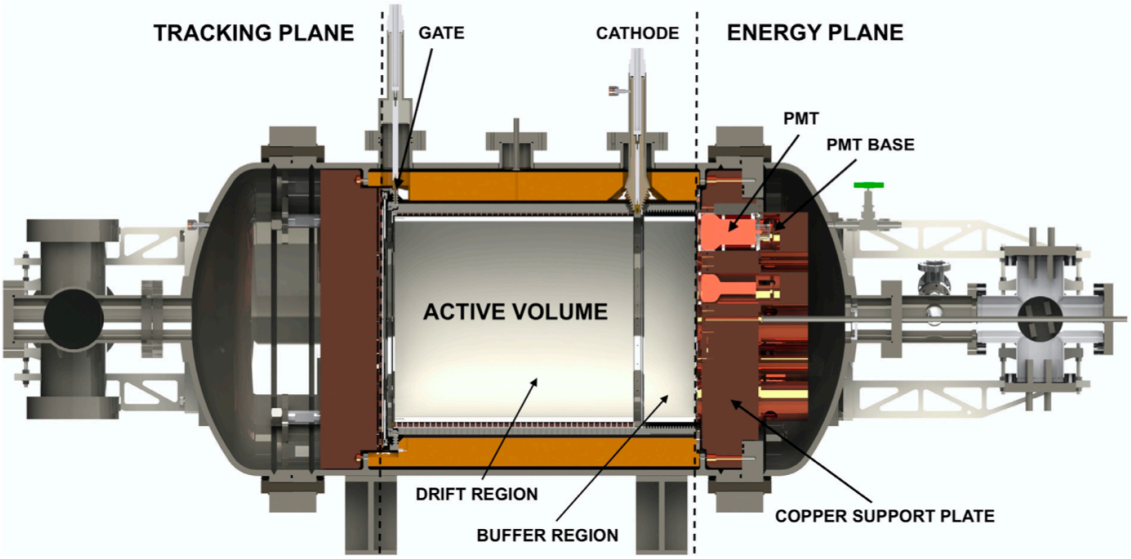

Figura 1. Cambra de Projecció Temporal de l'experiment NEXT [9]

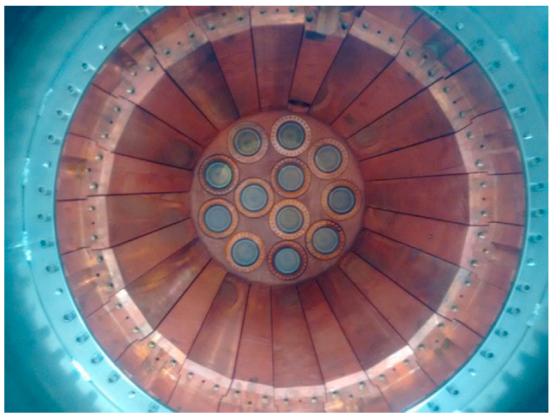

Figura 2. Pla d'energia vist de de l'ànode [9]

4.2. Sistema d'adquisició de dades de l'experiment

El sistema d'adquisició de dades està basat en mòduls d'FPGA que estan contínuament emmagatzemant informació en un buffer circular [9]. Per facilitar l'anàlisi posterior offline, s'ha incorporat un sistema de trigger per discriminar les dades importants de la resta. El mòdul d'adquisició ja genera els possibles candidats per al mòdul de trigger, els quals poden ser acceptats o no. En cas de ser acceptats, les dades són enviades a una granja d'ordinadors [10].

Les imatges presentades a continuació mostren uns exemples de les formes d'ona de l'experiment. La Figura 3 representa l'adquisició de dades dels senyals amplificats per electroluminescència dels PMTs, mentre que la Figura 4 mostra la suma dels senyals dels PMTs ja corregits sense cap ampliació.

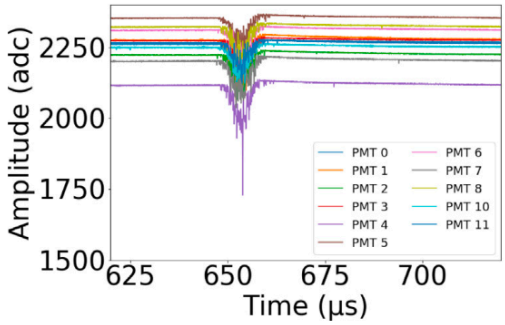

Figura 3. Exemple de senyal EL amplificat que activa el trigger [9]

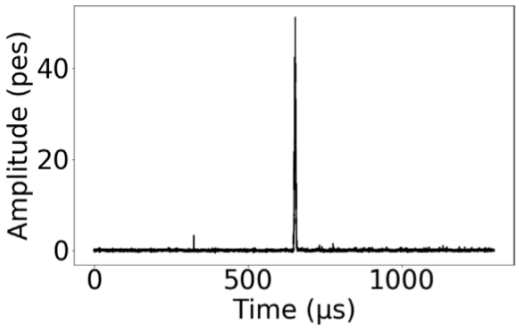

Figura 4. Forma d'ona corregida adquirida [9]

Tot aquest sistema d'adquisició està compost per targetes digitals anomenades Front-End Cards (FEC), fent d'interfície de dos tipus diferents de targetes. A més, targetes que actuen com a buffers de senyals digitals, DTC (Direct to Card) s'utilitzen per a connectar dades i configuració entre FEC i les targetes digitals que discretitzen i processen els senyals dels sensors SiPMs, anomenades FE cards, i targetes per a la discretització de dades dels PMTs, anomenades ADC (Analog to Digital Converter) cards. Més amunt en la jerarquia d'adquisició es troben els PCs, als quals els fa d'interfície una connexió per GbE.

Les dades utilitzades a aquest projecte són les provinents del PMT, els quals utilitzen uns ADC de 12 bits que mostregen a 40 MHz el senyal analògic. Una vegada digitalitzada i processada la informació per a l'enviament, aquest es produeix a 10 MBytes/s per als 12 PTMs amb una freqüència de mostreig màxima de 10 Hz amb una grandària estàndard de buffer de memòria de 1300 µs sense compressió de dades.

Aquestes dades rebudes de l'experiment no són exactament les que es rebrà al programa del projecte. La compressió és més elevada si les dades es tracten de forma diferencial; és a dir, si el procés s'aplica sobre la diferència entre la dada anterior i l'actual. Aquest format és el que s'ha tingut en compte per a realitzar el treball. Les dades continuen enviant-se en format de 12 bits.

Una representació del sistema d'adquisició de dades de l'experiment pot veure's a la Figura 5.

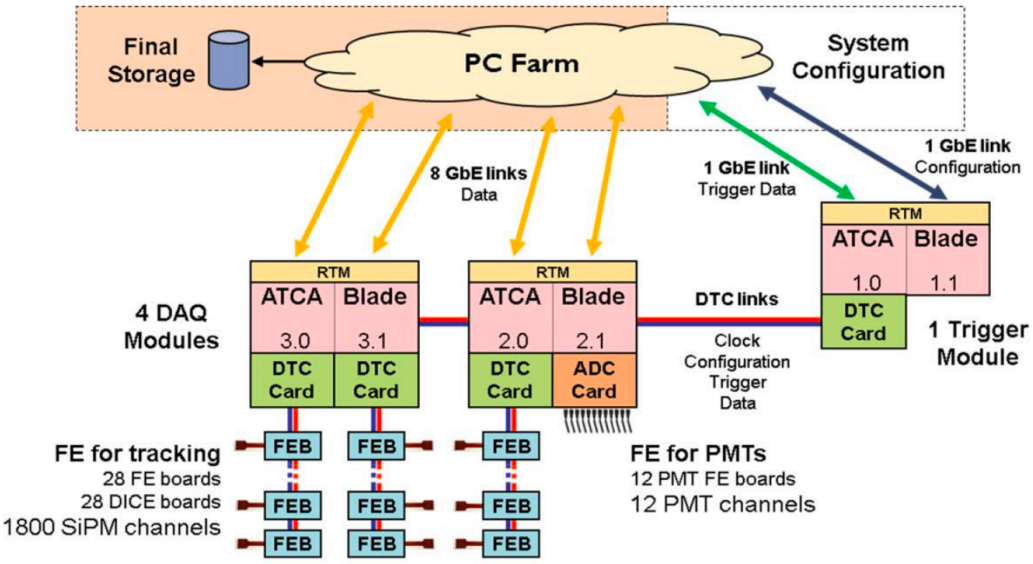

Figura 5. Sistema NEXT-NEW [11]

4.3. Antecedents

Aquest projecte forma part del sistema d'adquisició i processament de dades de l'experiment NEXT. Dins d'aquest processament es troba la compressió de dades, la qual utilitzarà el mòdul programat en aquest projecte. Part del programa de compressió forma part de [12], on s'estudien diversos mètodes, quedant-se finalment la compressió mitjançant l'arbre de Huffman amb dades diferencials. En aquest programa, la compressió utilitza una codificació fixa per als valors diferencials més probables a l'experiment. Amb el present projecte, aquesta taula fixa passarà a ser-ne una dinàmica calculada a partir de dades estadístiques.

5. MATERIAL UTILITZAT AL PROJECTE

Aquest projecte ha sigut plenament desenvolupat amb un ordinador i amb els programes necessaris per al disseny del codi i les seues proves pertinents per veure el correcte funcionament. Aquests programes són l'ISE Design Suite de Xilinx amb el que s'ha programat el circuit digital i ModelSim, amb el que s'han anat realitzant les simulacions dels diferents mòduls.

Malgrat que no s'ha utilitzat, s'inclourà també al present apartat la targeta FPGA on serà enviat el programa. S'ha de tindre en compte que aquesta targeta està inclosa a l'experiment, ja conté programats circuits d'adquisició i de compressió de dades i no es pot utilitzar amb plena llibertat.

5.1. Targeta FPGA

L'FPGA (Field Programmable Gate Array) utilitzada al projecte és la Virtex-6 de Xilinx, model XC6VLX240T. Aquests dispositius electrònics estan dissenyats a partir de matrius de Blocs de Lògica Configurable (Configurable Logic Blocks en anglès, CLBs) interconnectades de manera programable. Al seu torn, les CLB estan formades a partir de "Slices", que són aproximadament dos Cèl·lules Lògiques (LC, de l'anglès Logic Cell). Dins d'un "Slice", les LC comparteixen senyals de rellotge, com l'habilitació d'aquest. Un LC es el bloc més bàsic d'un dispositiu programable, i conté algun mòdul capaç de generar funcions lògiques, com una LUT (Look-Up Table) més un registre [13].

Disposen a més de recursos ja incorporats com memòries RAM, microprocessadors, estàndards de comunicació, etc. Resulten d'aquesta manera en dispositius programables que poden adaptar-se a aplicacions molt diverses [13] [14].

Aquesta filosofia pot recordar als ASICs (Application Specific Integrated Circuits) , els quals són dissenyats i fabricats específicament per a una feina concreta. No obstant això, la majoria d'FPGAs són reprogramables, el que implica que el sistema pot evolucionar amb l'aplicació en qüestió (dins dels seus límits tècnics) [14].

Es presenten a continuació (Taula 1) les especificacions del dispositiu on hi anirà el programa. Cal mencionar que a dintre d'aquest no sols va el present projecte. El programa formarà part del sistema d'adquisició de dades i de compressió ja existents.

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices(2)	Block RAM Blocks			MMCMs(4)	Interface Blocks for PCI Express(5)	Ethernet MACs(6)	Maximum Transceivers		Total I/O Banks(7)	Max User I/O(8)
		Slices(1)	Max Distributed RAM (Kb)		18 Kb(3)	36 Kb	Max (Kb)				GTX	GTH		
XC6VLX75T	74,496	11,640	1,045	288	312	156	5,616	6	1	4	12	0	9	360
XC6VLX130T	128,000	20,000	1,740	480	528	264	9,504	10	2	4	20	0	15	600
XC6VLX195T	199,680	31,200	3,040	640	688	344	12,384	10	2	4	20	0	15	600
XC6VLX240T	241,152	37,680	3,650	768	832	416	14,976	12	2	4	24	0	18	720
XC6VLX365T	364,032	56,880	4,130	576	832	416	14,976	12	2	4	24	0	18	720
XC6VLX550T	549,888	85,920	6,200	864	1,264	632	22,752	18	2	4	36	0	30	1200
XC6VLX760	758,784	118,560	8,280	864	1,440	720	25,920	18	0	0	0	0	30	1200

Taula 1. Especificacions tècniques la família d'FPGA Virtex-6 [15]

5.2. ISE Xilinx

Com bé s'ha comentat en apartats anteriors, el llenguatge de programació utilitzat al present projecte és el Verilog. Existeixen diversos entorns de programació per a aquest, tals com Quartus II d'Altera (pertanyent a Intel) o Vivado Design Suite de Xilinx. Sent ambdós fabricants els més importants del mercat [13].

En aquest cas, s'ha d'utilitzar l'ISE Design Suite de Xilinx. És l'entorn de programació anterior de Xilinx, que fou substituït el 2013 pel Vivado Design Suite; però que suporta fins als dispositius Spartan-6, Virtex-6 (la utilitzada al projecte) i CoolRunner. Cal afegir que hi ha tres versions disponibles: Embedded Edition, System Edition i WebPACK Edition, de les quals la utilitzada ha sigut l'última d'aquestes [16].

La WebPACK Edition és la versió més bàsica de les tres. No obstant això, té totes les funcionalitats necessàries per al projecte. Ofereix suport tant per a FPGA com per a CPLD amb llenguatges HDL i inclou, entre altres, el CORE Generator, que permet la generació de mòduls estàndard per ser utilitzats als projectes, des de mòduls d'aritmètica bàsics fins a blocs de memòria configurables. El resultat és una ferramenta prou potent per a aquest nivell de complexitat amb molt bones capacitats per a l'anàlisi posterior a la programació [17]. Entrant en aquesta anàlisi posterior esmentada, es veurà en apartats següents que el programa proporciona esquemes RTL del circuit programat.

La Figura 6 mostra la interfície del programa, on es pot observar una finestra gran on es fa la programació, unes xicotetes finestres a l'esquerra per a la navegació entre mòduls i una finestra baix per a missatges informatius.

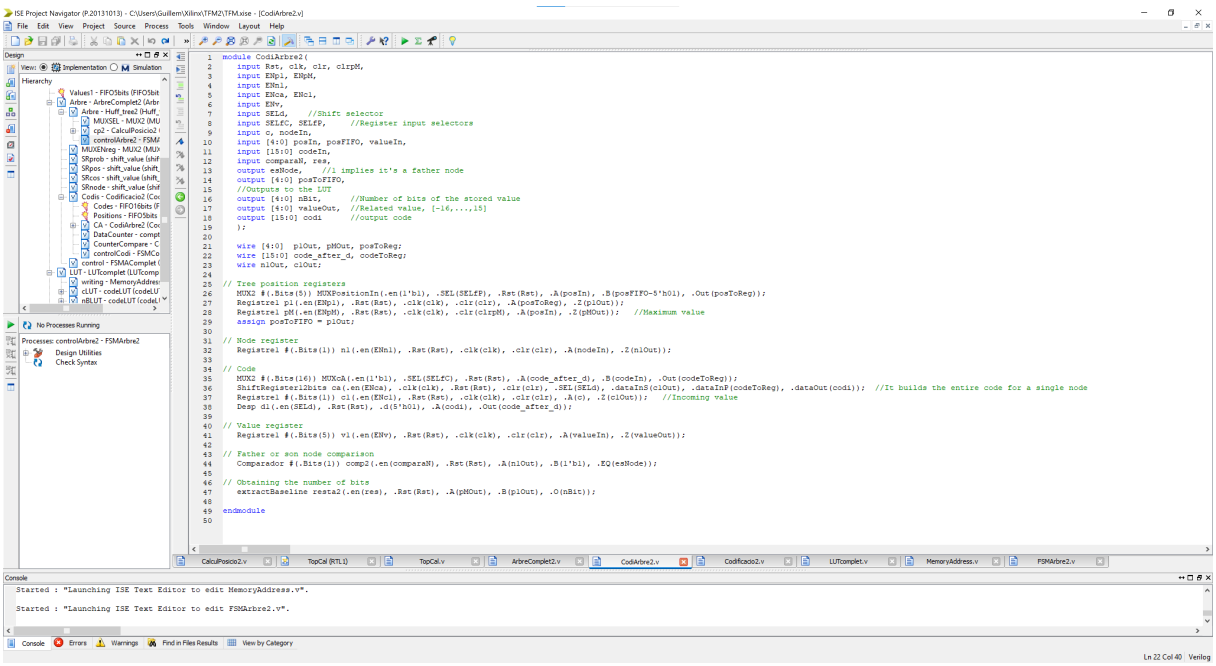

Figura 6. Interfície de l'ISE Design Suite de Xilinx

5.3. ModelSim

Malgrat que el programa anterior té algunes ferramentes de simulació, per a aquesta tasca s'ha emprat el ModelSim PE Student Edition. Suporta llenguatges estàndard HDL (VHDL i Verilog) sense barrejar, encara que la versió completa sí que permet dissenys amb els dos llenguatges. La simulació pot ser tant de programes comportamentals ("behavioural"), com RTL. També permet l'edició dels codis i la compilació dels programes al seu entorn per a la posterior simulació [18]. La interfície d'usuari és prou intuïtiva, permetent per exemple una fàcil addició de variables a la simulació, ja que per defecte la simulació sols mostra les entrades i eixides del circuit. A més, opera conjuntament amb l'ISE de Xilinx, cosa que facilita molt la feina a l'hora de compilar tots els mòduls al ModelSim.

A la Figura 7 es presenta l'entorn d'aquest programa. En aquesta, pot apreciar-se una pantalla principal on es troben algunes de les entrades, eixides i altres variables del circuit representades. La finestra de navegació de l'esquerra de la imatge permet la fàcil addició de les diferents variables del circuit.

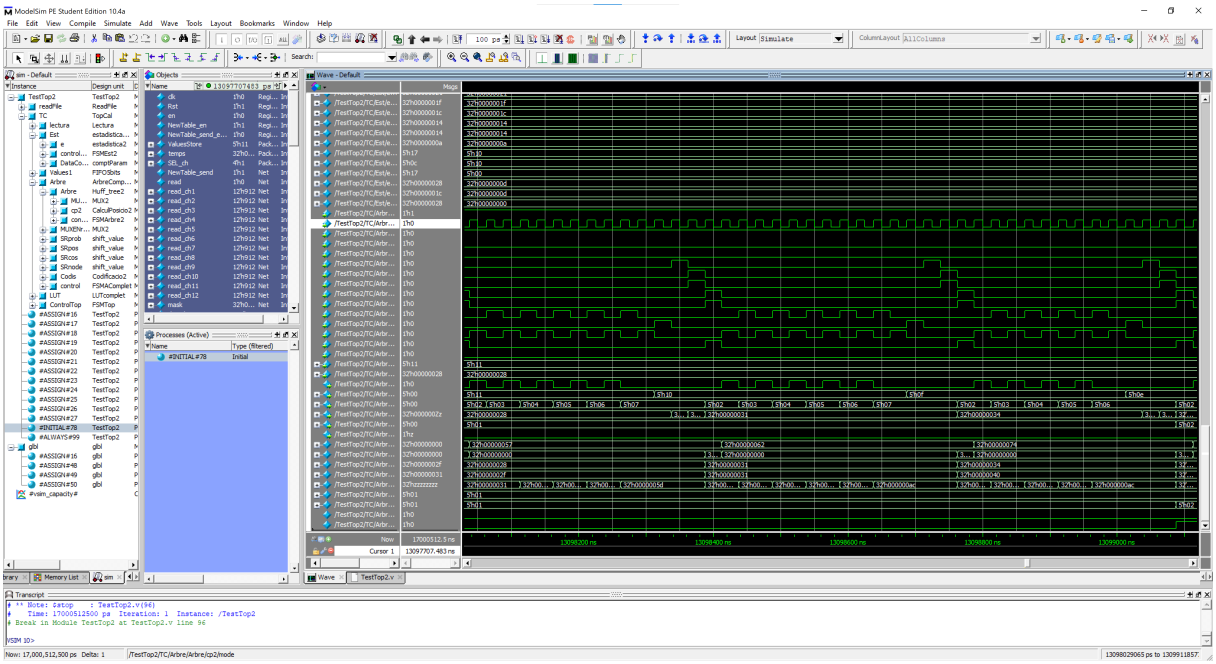

Figura 7. Interfície del ModelSim PE Student Edition d'Altera

5.4. Jupyter Notebook

Jupyter és un projecte per crear programes de codi obert per a computació interactiva mitjançant diversos llenguatges de programació. El Notebook en concret és una aplicació web que permet crear i compartir codis d'una manera molt intuïtiva. Cal destacar les eixides interactives de què disposa, tals com HTML, vídeos, etc. Suporta uns 40 llenguatges de programació, dels quals s'ha utilitzats el Python 3 [19].

Junt a aquest programa es disposava d'uns scripts on hi era programat l'algorisme treballat en aquest projecte[1]. Gràcies a aquest, s'ha pogut primerament obtindre el rang de dades més probables d'entre totes les possibles de l'experiment. A més, ha servit per, una volta finalitzat el programa del projecte, verificar el seu funcionament comparant els arbres codificats d'aquest amb els del programa en Python. També s'ha utilitzat un codi per passar dels arxius d'eixida de l'experiment a fitxers de text. A la Figura 8 es pot veure un exemple de la interfície d'aquest programa.

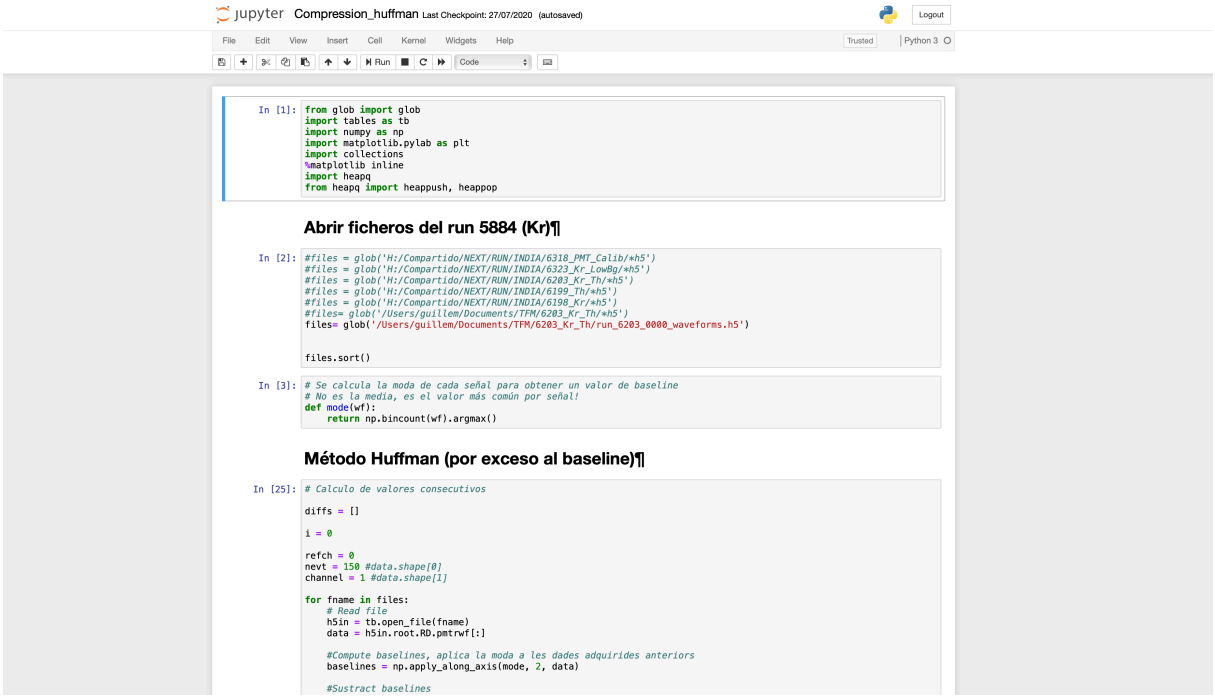

Figura 8. Interfície del Jupyter Notebook al navegador web

[1] Més detalls poden veure's a l'Annex núm. 3

6. DESENVOLUPAMENT DE LA SOLUCIÓ

6.1. L'arbre de Huffman

El mètode de Huffman, proposat al 1952 per David A. Huffman, es basa en assignar codis amb un nombre de bits relacionat amb la probabilitat que tinga una mostra d'aparèixer, major freqüència implica menys quantitat de bits. L'arbre es construeix de manera que cada node "pare" té una probabilitat igual a la suma de les dels nodes "fills". En tot l'arbre s'ha de mantindre el mateix ordre, bé a l'esquerra va el "fill" amb menys probabilitat o bé a la dreta [3].

L'algorisme de construcció comença amb la suma dels dos valors menys probables, afegint la suma als valors existents i procedint de nou amb els dos menys probables. Si es contemplen la totalitat de les dades a l'arbre, la probabilitat de l'últim node pare ("arrel") serà del 100%. S'exemplifica aquest algorisme amb la Figura 9.

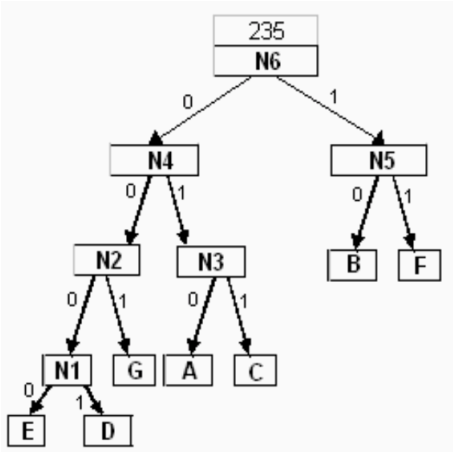

Figura 9. Exemple d'arbre de Huffman [3]

Una volta construït l'arbre, es procedeix a assignar a cada branca un 1 o un 0 de forma coherent a tota l'estructura (el que s'anomenarà bit de "branca" a partir d'aquest punt al projecte). El codi de cada node serà el resultat de l'addició del bit corresponent segons les branques que s'han seguit per arribar a aquest. El primer bit assignat serà el MSB (Most Significant Bit) i l'últim, el de la branca prèvia al node, el LSB (Less Significant Bit). Per exemple, per a la Figura 9, el codi per a "A" seria "010". Cal esmentar que l'arbre no és únic; l'aparició de nodes amb la mateixa probabilitat fa que la codificació puga diferir.

6.2. Consideracions prèvies

El pla d'energia actual a l'experiment consta de 12 PMTs i, per tant, 12 canals diferents dels quals s'obtenen les dades utilitzades en aquest projecte. Per fer un càlcul més complet de la probabilitat de cadascuna de les dades seria lògic pensar que s'han d'analitzar tots els canals. No obstant, com es detallarà a l'apartat 8, els recursos disponibles a la targeta FPGA actual són escassos i fer un circuit que incloga la informació de tots els PMT podria traduir-se en un augment considerable dels recursos que empraria el programa. Per tant, tenint açò en consideració, sols s'utilitzaran les dades d'un sol canal mitjançant un multiplexor.

Les conseqüències d'aquesta decisió sobre l'experiment són menors, ja que, en principi, tots els sensors "veuen" el mateix. És cert que amb més dades recollides es millora la fidelitat de l'experiment, però s'ha de tindre en compte també que amb la compressió òptima poden adquirir-se més esdeveniments per unitat de temps, al reduir la quantitat de dades transmeses per esdeveniment.

El programa de compressió ja incorporat a l'FPGA està dissenyat per a una taula estàtica de 17 valors. Com es veurà a l'apartat VERIFICACIÓ DEL FUNCIONAMENT DEL PROJECTE, es comprova sobretot que el programa funciona amb aquesta quantitat. Malgrat això, el disseny del present projecte contempla taules amb més o menys valors (des d'1 fins a 32), per la qual cosa es podria, si fóra necessari, modificar el programa de compressió per admetre una quantitat diferent de dades.

6.3. Descripció de la seqüència de funcionament

Es procedeix a continuació a fer un breu resum sobre el funcionament del sistema i el flux de dades que té lloc a dintre a mode d'introducció a tota l'explicació del programa. Per una major claredat a l'hora d'explicar els processos, es presenta la Figura 10. Tot allò introduït en aquest apartat serà esplaiat posteriorment als seus apartats corresponents.

Primerament, s'observa a la figura següent que quan el programa es posa en funcionament, s'espera fins que no s'estiga produint cap esdeveniment a l'experiment. Una volta el sistema comença, fa la lectura de les dades que li arriben i les va comptant. S'ha de tindre en compte que no poden comptar-se totes les dades que puguen donar-se; és per això que s'ha fet una selecció mitjançant un programa de Python[2] de les 32 dades més probables.

Com s'ha comentat a l'apartat anterior, el sistema actual està dissenyat per a una taula de 17 valors. Per tant, d'aquestes 32 dades se'n seleccionaran les 17 més probables. Aquesta quantitat és variable i pot modificar-se per futures modificacions del programa de compressió.

Una vegada seleccionats els valors que compondran l'arbre de Huffman, els seus recomptes són enviats a la part del programa que construirà aquest arbre, per després codificar-lo i actualitzar la taula de codis ja implementada a l'FPGA.

Durant tot el procés d'aquest programa, s'envia un senyal als altres programes de l'FPGA per indicar que s'està creant i enviant una nova taula. Senyal que finalitzarà una volta aplegue al seu destí.

[2] Una explicació d'aquest pot veure's a l'Annex núm. 3.

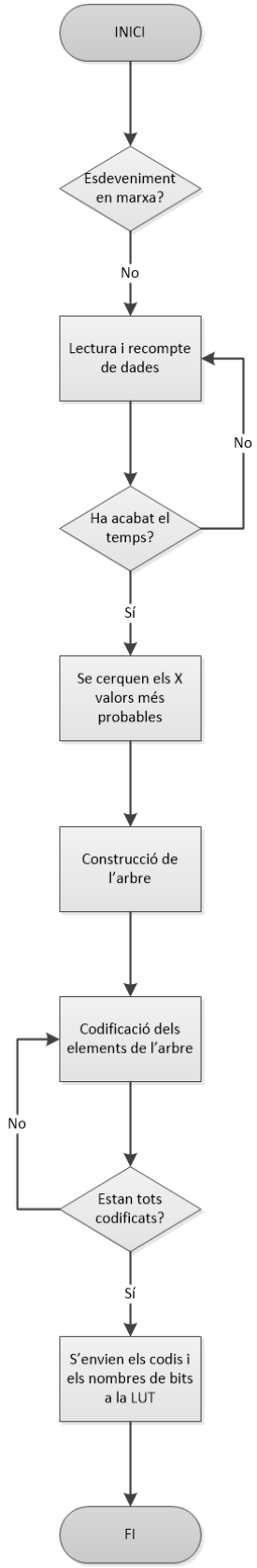

Figura 10. Diagrama de flux del programa complet

6.4. Descripció del circuit digital implementat

El circuit general generador de codis Huffman s'ha estructurat en tres blocs: el sistema d'adquisició de dades (Mòdul d'Estadística), l'estructuració d'aquestes en un arbre de Huffman (Mòdul de l'Arbre) i la codificació dels elements rellevants de l'arbre (Mòdul de Codificació), els anomenats nodes fulla d'aquest. En la Figura 11 pot apreciar-se la connexió entre aquests blocs, així com el flux de dades que es transfereixen entre ells. Al programa entren dades ja processades per altres mòduls que es troben implementats a l'FPGA. De la mateixa manera, l'eixida de dades va a una LUT ja implementada que servirà per a la compressió/descompressió.

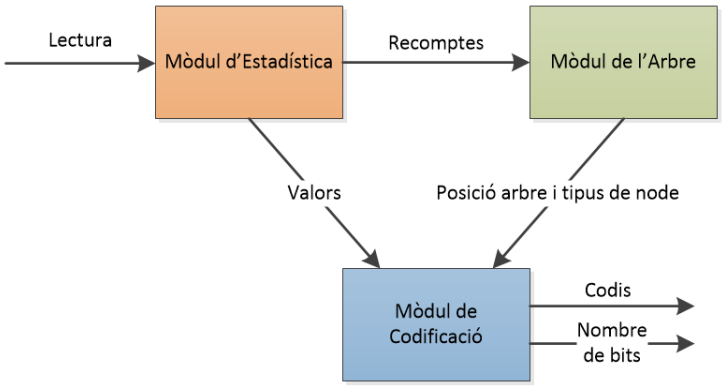

Figura 11. Diagrama dels tres blocs del programa

En aquest apartat, s'explicarà sense entrar en detall de programació el funcionament del programa. Una anàlisi més exhaustiva serà realitzada a l'apartat DETALL DEL CIRCUIT DIGITAL IMPLEMENTAT, on s'explicarà com estan dissenyats cadascun dels blocs; incloent tant els mòduls que els componen com les connexions entre aquests i les màquines d'estats que hi ha darrere perquè tot funcione correctament.

6.4.1. Descripció del Mòdul d'Estadística

En primer lloc, es fa un recompte de les dades d'entrada. Com s'ha comentat en apartats anteriors, les dades del projecte són la diferència entre la dada anterior i l'actual recollides dels PMTs. Per començar amb la creació de l'arbre, és necessari saber abans la probabilitat d'ocurrència de cadascuna de les dades. No obstant, el fet de calcular les probabilitats consumeix més recursos que si sols es té en compte el nombre total de voltes que es dóna un valor en concret. Com a efectes pràctics és el mateix, el procediment es farà sobre els recomptes.

Seria molt ineficient tindre en consideració tots els possibles valors que es poden donar a l'entrada, en cas de ser possible. Per aquest motiu, sols es comptaran els valors pertanyents a un rang que prèviament s'ha seleccionat utilitzant un programa de Python[3]; la resta seran ignorats pel programa. Si bé és cert que els valors amb menys probabilitat de dins del rang, depenent de l'esdeveniment, podrien quedar-se fora en determinades ocasions deixant pas a altres que no estan sent considerats, s'observarà a les simulacions que a sovint el valor "comprimit" té el mateix nombre de bits que el valor original.

El programa de Python esmentat té una funcionalitat similar al programa d'aquest projecte. Gràcies al càlcul que fa de les probabilitats dels diferents valors, pot escollir-se el rang a tindre en compte. La Figura 12 mostra que els 32 valors més probables es troben dins del rang [-16, 15]. El rang depèn de les dades analitzades. La imatge correspon a un experiment de calibratge concret i tenint en compte un canal (el programa sols en processarà un cada vegada) i un fitxer de dades d'entre tots els disponibles. Aquesta decisió no afecta molt al resultat final. Els valors de menor probabilitat sovint són comprimits amb 12 bits, que és la grandària que ja tenen per defecte i els valors més probables es mantenen més o menys estables a tots els fitxers. A més, la diferència entre probabilitats més menudes és insignificant en comparació amb els nombres més probables com es pot veure a la Figura 12.

```
print(tree)
```

```
[(17, 8.904128052699988e-05), (-17, 8.937476846904857e-05), (16, 9.748964172556666e-05), (-16, 9.937940673050923e-05), (15, 0.00010949520763931946), (-15, 0.00011171846058631071), (-14, 0.00012483565297355913), (14, 0.00012544704753398174), (-13, 0.00013383982740887372), (13, 0.00013467354726399544), (12, 0.00014745725170919 52), (-12, 0.00015195933892685247), (-11, 0.00016763327220314085), (11, 0.00017035675706320513), (10, 0.00018747580475503783), (-10, 0.00018758696740238 74), (9, 0.00021004182216699 91), (-9, 0.00021259856305603904), (-8, 0.00024678107711602 96), (8, 0.00024722572770542 79), (7, 0.00030519704829822 49), (-7, 0.00030647541874274 49), (-6, 0.00040352040987891 33), (6, 0.00040918970489374 1), (5, 0.00059455341934913 72), (-5, 0.00060100085289541 18), (4, 0.00102208496105555 6), (-4, 0.00109478533242217 01), (3, 0.00553884564816299 5), (-3, 0.00561071229967448 8), (-2, 0.05294437893567879), (2, 0.05303308672826374), (-1, 0.23799389216834138), (1, 0.23897923787444791), (0, 0.3959847495964101)]
```

Figura 12. 35 valors més probables amb la seua probabilitat. Càlcul fet amb 200 esdeveniments. Entre parèntesi (nombre, probabilitat/100).

El procediment que se segueix és el mostrat la Figura 13. Es fa primer la petició de lectura. La dada d'entrada és comparada per veure si és a dins del rang de valors seleccionats. En cas favorable, s'afegirà al compte. El recompte es farà durant un temps preestablert suficient perquè les dades recollides siguen més o menys representatives del total.

[3] Veure Annex núm. 3

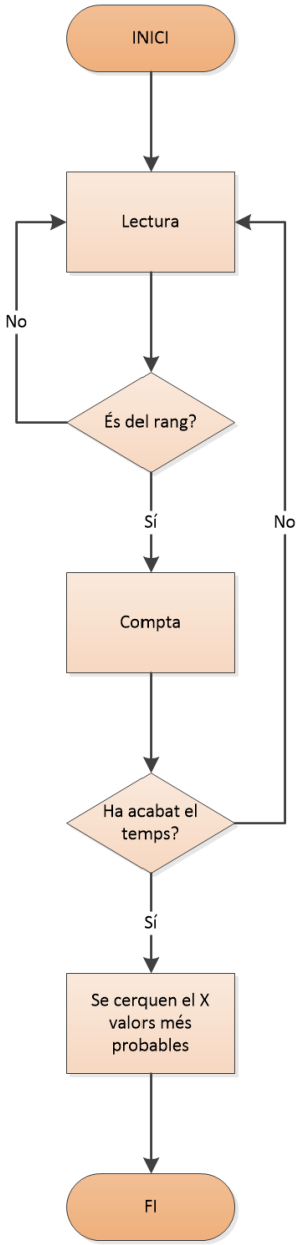

Figura 13. Diagrama de flux del primer bloc

Una volta finalitzat el recompte, les dades recollides són comparades entre elles per ser transferides al següent bloc. No totes les dades que s'han considerat per al recompte són enviades. El sistema té com a entrada aquest nombre de dades que passen. Per açò, les dades són comparades i introduïdes de major a menor probabilitat fins aplegar a la quantitat seleccionada. Els valors corresponents als comptes són enviats a una llista FIFO per ser utilitzats més endavant (a l'apartat 7.4.).

6.4.2. Descripció del Mòdul de l'Arbre

Es procedeix a aquest bloc a aplicar l'algorisme explicat amb anterioritat sobre els recomptes fets, mostrant-se l'esquema de funcionament a la Figura 14. Així, s'extrauen els dos menors i se sumen. Al mateix temps, es registra un 1 per al valor més xicotet dels dos i un 0 per al major (el que serà el bit de "branca") i s'identifica la naturalesa del node ("pare" de més nodes o una "fulla" de l'arbre) i l'altura on és dins de l'arbre. Per altra banda, el resultat de la suma es compara amb les dades romanents i s'ubica al seu lloc corresponent segons l'ordre seguit. El procés acaba quan els dos últims valors són sumats i és enviada la informació relacionada amb aquests i el seu "pare".

La posició dins l'arbre (altura i si és el node esquerre o dret) i la naturalesa del node són registrats tots primer i després enviats a la codificació. És necessari fer-ho d'aquesta manera perquè l'ordre que s'està seguint al fer l'arbre és de menor a major probabilitat i la codificació comença des de "l'arrel" de l'arbre, que és el major de tots. Per tant, fins que no s'obté aquest últim valor no pot començar aquest últim procés.

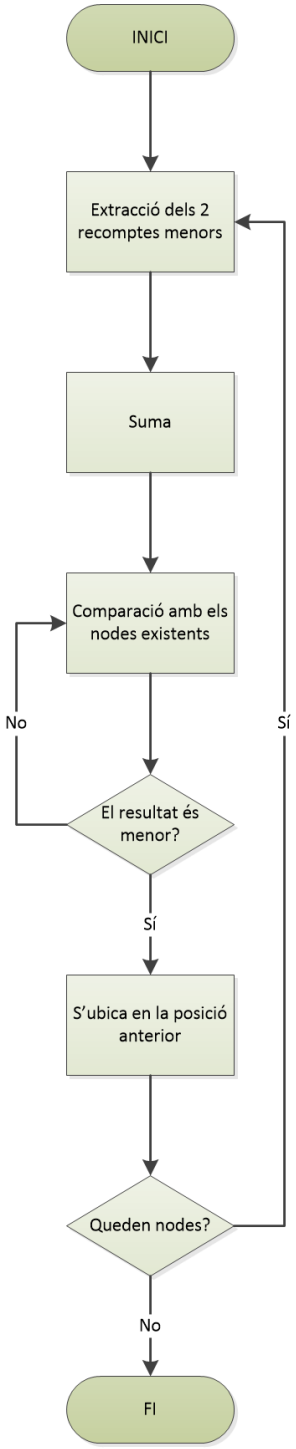

Figura 14. Diagrama de flux del segon bloc

6.4.3. Descripció del Mòdul de Codificació

L'algorisme a seguir és el mostrat la Figura 15. La primera volta s'extrauen dels corresponents registres les dades emmagatzemades al mòdul anterior (posició a l'arbre i tipus de node) i s'emmagatzema l'altura del node "arrel". Açò servirà més endavant per saber el nombre de bits d'un codi en concret.

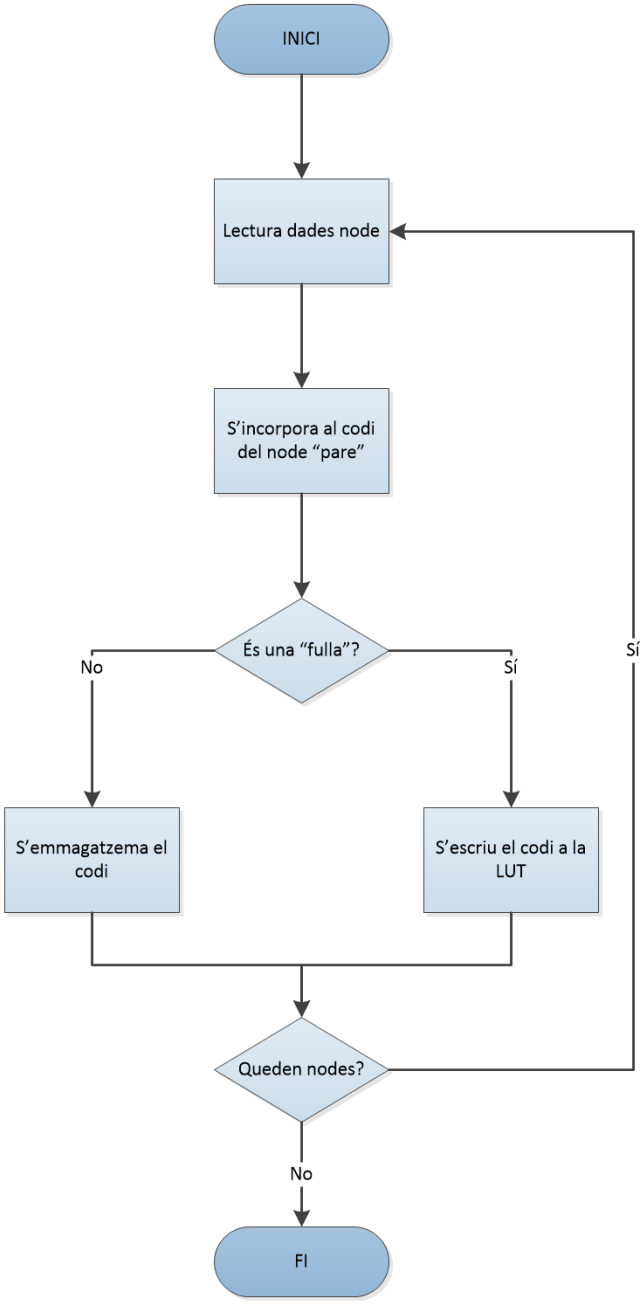

Figura 15. Diagrama de flux del tercer bloc

El que fa el sistema a continuació és llegir la següent dada al node "arrel" i veure si és una "fulla" de l'arbre o no. En cas de ser-ho, envia el seu bit de "branca" (si és l'esquerre o el dret) junt amb el valor que li correspon del rang a un conjunt de registres. Es recorda que aquest valor fa referència a les dades d'entrada del sistema (el rang seleccionat) i que havien sigut prèviament enviades a una llista FIFO en ordre de major a menor probabilitat al primer bloc de tots. També s'enviarà als registres el nombre de bits que té el codi. Al primer cas, si és una "fulla" que ix de l'"arrel", el nombre serà d'1 bit. En cas de no ser una "fulla" de l'arbre, el codi del node i l'altura d'aquest són enviats a unes cues FIFO. L'altura serà la que s'havia emmagatzemat com a màxima restant-li 1. Sabent que cada node "pare" sols té dos "fills", es fa un desplaçament del codi del node "germà" per tornar a tindre el del "pare" i poder així incorporar el seu bit de "branca".

Fetes aquestes primeres passes, el que es fa és llegir el bit de "branca" i el tipus de node als registres de l'anterior bloc (igual que al primer cas) i llegir les cues FIFO per saber les dades del pare d'aquest node. Amb tota aquesta informació es torna a determinar si és una "fulla" o no i s'envien les dades als registres o a la cua respectivament. L'altura del node serà sempre la llegida a la cua restant-li 1, i el nombre de bits que té el codi, la resta entre el valor màxim emmagatzemat a l'inici i aquesta altura calculada. Com bé s'acaba de comentar, al tindre cada "pare" dos "fills", la lectura de la cua sols es farà per al primer. Per al segon es realitza sempre un desplaçament del codi del "germà".

El procés acabarà quan els registres d'intercanvi entre aquest bloc i l'anterior siguen buits. El resultat de tot açò seran unes parelles de registres amb la codificació corresponent a cada "fulla" i el nombre de bits que té el codi.

7. DETALL DEL CIRCUIT DIGITAL IMPLEMENTAT

El programa està estructurat en tres grans blocs. En aquest apartat es procedirà a fer una xicoteta anàlisi de la programació, els circuits implementats i les màquines d'estats de cadascun d'aquests. Es detallarà primerament els mòduls més generals i, després, es concretarà particularment per a cada mòdul.

Els esquemes electrònics que es mostraran són simplificacions dels circuits reals. Per a una explicació més detallada dels components del circuit, pot consultar-se l'Annex núm. 2Annex núm. 1: Detalls de la programació .

7.1. Mòduls superiors

En aquest primer apartat es detallaran les FSMs encarregades de controlar les que seran posteriorment explicades als apartats següents i els mòduls que les contenen. Hi ha dues FSM diferents: una que controla el segon i tercer bloc, és a dir, tot allò relacionat amb l'arbre, i una altra que controla tot el programa, incloent aquesta primera màquina.

La Figura 16 mostra un esquema dels mòduls principal en què es compon el projecte. S'observa en aquesta el mòdul de lectura, el mòdul "estadisticaComplet" (que es correspon al bloc d'Estadística) i el mòdul "ArbreComplet2" (que es correspon al bloc de l'Arbre i de Codificació). També hi apareixen la màquina d'estat principal i els registres d'eixida, que seran explicats al bloc de Codificació, però es decidí ubicar-los al mòdul principal per comoditat.

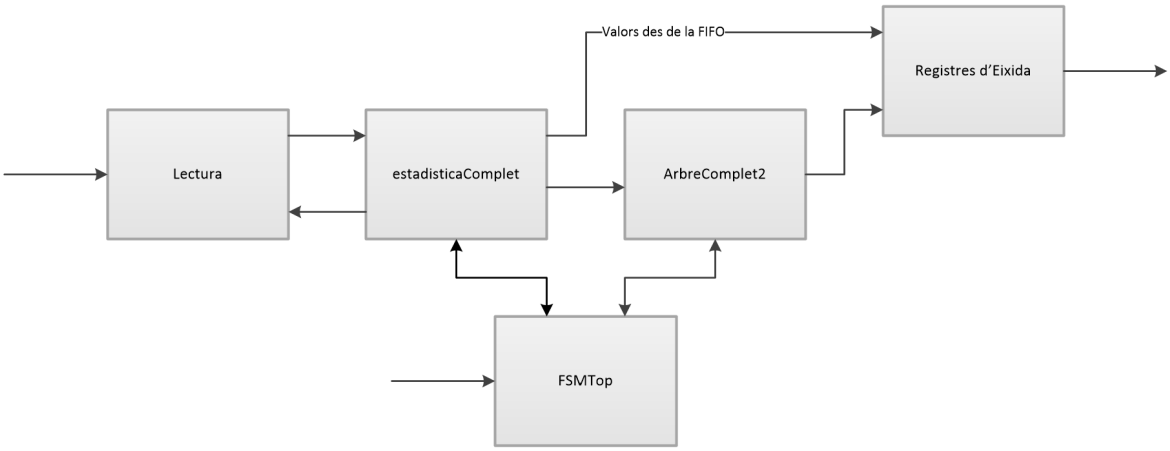

Figura 16. Esquema resum dels mòduls principals del projecte

Per altra banda, a la Figura 17 es presenta un esquema de l'interior del mòdul "ArbreComplet2". Aquest mòdul conté els bloc de l'Arbre i la Codificació, una sèrie de registres de desplaçament per sincronitzar les dades entre aquests mòduls i la màquina d'estats pertinent per coordinar els dos blocs.

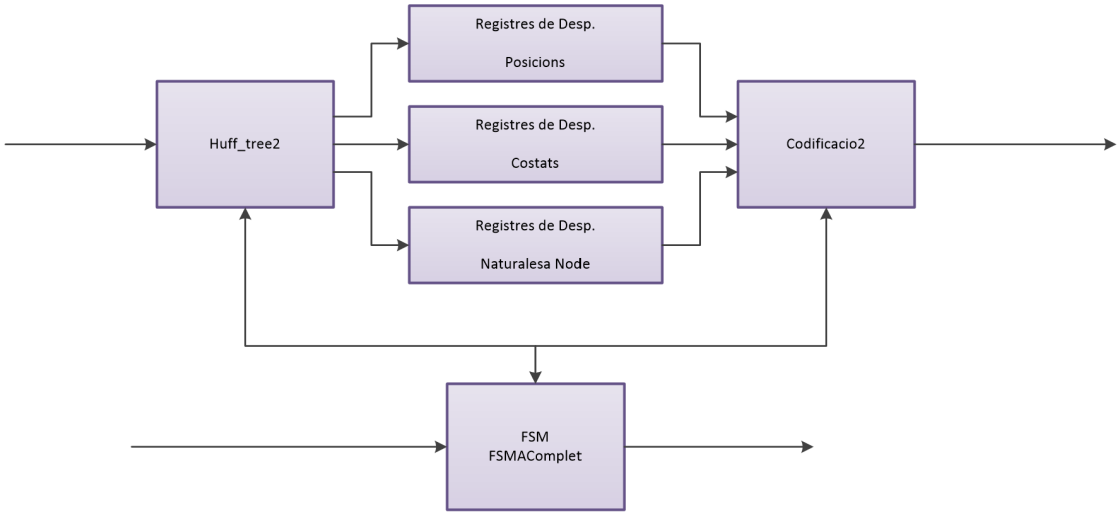

Figura 17. Esquema resum dels mòduls inclosos a "ArbreComplet2"

Començant per "FSMAComplet" a la Figura 18, es passa, com be sent comú, d'un estat de repòs a l'inici de la màquina amb una variable de "Start". Amb aquesta FSM el que es pretendrà es donar els senyals d'inici per a les FSMs del segon i tercer mòdul, és a dir, les que controlen la construcció de l'arbre i la seua codificació.

Així, amb "E1" s'activaria el senyal perquè comence a funcionar la primera i amb "E3", el senyal per a la segona. Entre aquests dos es troba un segon estat de repòs, en el qual entra una volta s'ha activat qualsevol dels dos estats explicats. Amb la finalització de la construcció de l'arbre és quan s'aniria a "E3". Si el que acaba és la codificació, ja s'activaria "E4", on es dóna per finalitzada aquesta FSM.

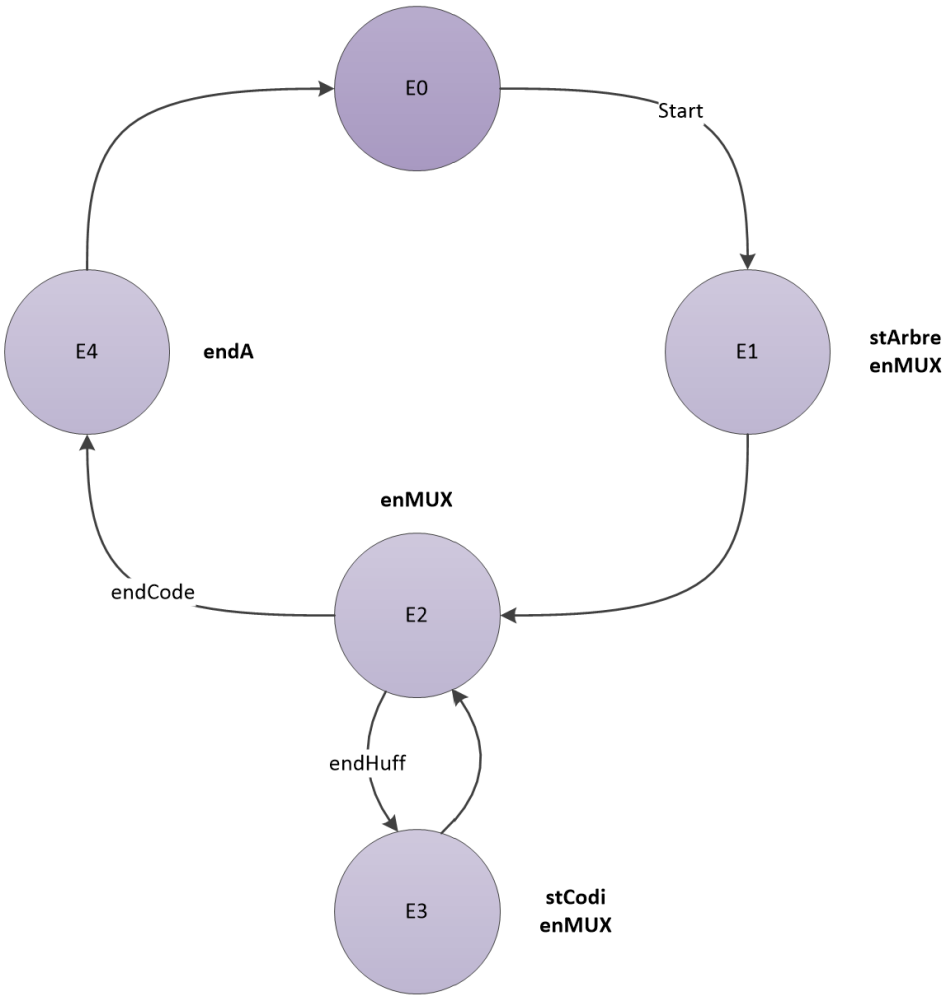

Figura 18. Diagrama d'estats d'"FSMAComplet"

El disseny d'"FSMTop" té un disseny similar a l'FSM anterior, tal i com pot observar-se a la Figura 19. L'estat inicial de repòs canvia no sols quan es dóna el senyal "Start", sinó també quan rep un senyal intern dels programes ja inclosos a l'FPGA, "NewTable_en". És a dir, la primera variable seria la que marcaria l'inici del programa, però la segona indica si es pot crear una taula nova o s'està recopilant dades i el sistema ha d'esperar.

La resta de la màquina d'estats és igual a l'anterior, per tant, queda ja explicada. En aquest cas hi ha una primera màquina d'estats que seria la pertanyent al primer bloc i una segona que seria l'explicada a l'inici d'aquest apartat.

L'única diferència que es troba entre les dues són els senyals "NewTable_send" i "NewTable_send_end". El primer es una variable d'un bit que es manté activa mentre el programa està en funcionament, indicant als altres mòduls de l'FPGA que s'està creant una nova taula de codificacions. El segon senyal prové del mòduls externs al projecte. Aquest indicaria que la taula ha sigut correctament rebuda i incorporada a la "Look-Up Table" corresponent, finalitzant així el programa.

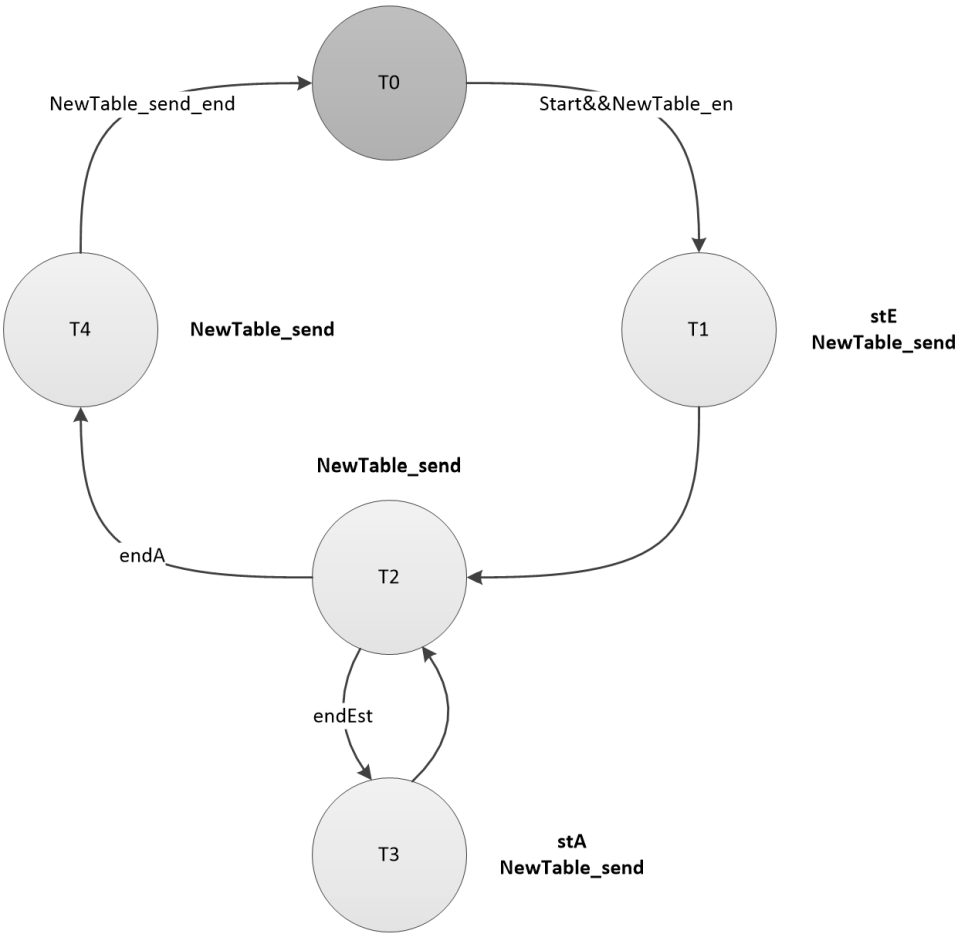

Figura 19. Diagrama d'estats d'"FSMTop"

7.2. Mòdul d'Estadística

Previ al mòdul on es fa la recollida de dades i la selecció dels més probables, hi ha programat un mòdul de lectura de dades. El disseny final d'aquest no és més que un multiplexor per seleccionar el canal d'on es recolliran els valors, ja que aquests vénen directament d'un mòdul anterior ja programat. No obstant això, per poder fer proves de funcionament s'han hagut de programar uns mòduls per adequar les dades d'entrada, obtenint-se la diferència entre l'anterior i l'actual. Per fer les proves pertinents, s'utilitzaran uns fitxers amb les dades llegides pels PMTs; per això es necessiten els mòduls d'adaptació.

La Figura 20 mostra el mòdul d'Estadística. Aquest està compost per un mòdul principal on es troba tota l'electrònica, una màquina d'estats i un cua FIFO on són enviats els valors que formaran part de l'arbre de Huffman. El mòdul de lectura prèviament explicat es troba fora d'aquest realment.

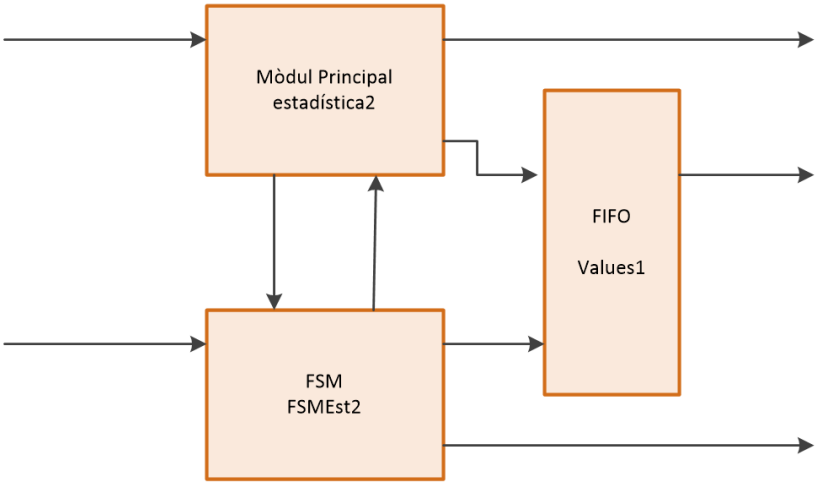

Figura 20. Esquema resum dels mòduls inclosos al mòdul d'Estadística ("estadisticaComplet")

El mòdul d'estadística, que és el que fa el recompte, disposa de 32 comparadors i comptadors per dur a terme aquesta tasca. El seu funcionament és simple: el valor d'entrada arriba als comparadors, es comprova si és un dels valors del rang i, en cas afirmatiu, s'activa el comptador corresponent. Aquest procés de compte de dades es fa durant un temps predeterminat mitjançant un temporitzador.

Abans de continuar amb els mòduls, es presenta la Màquina d'Estats (d'ara endavant FSM, Finite State Machine) d'aquest bloc (Figura 21).

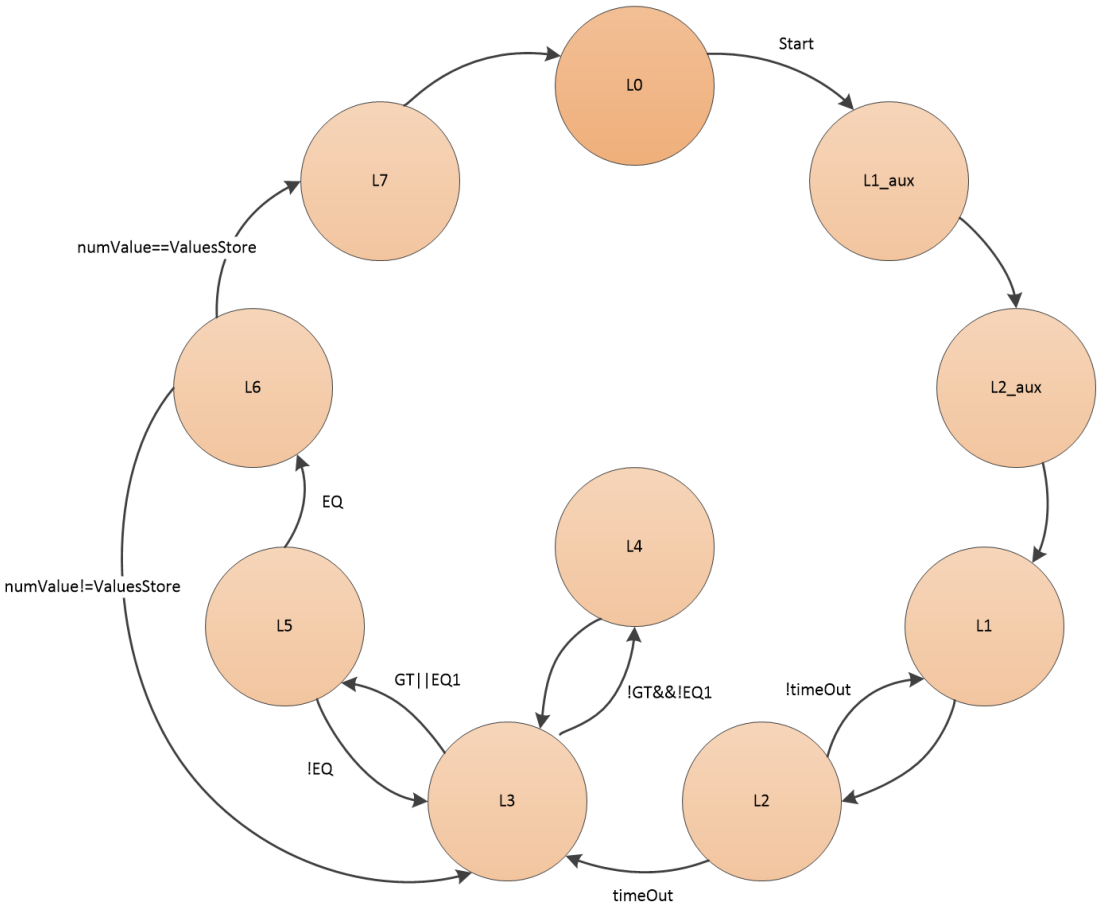

Figura 21. Diagrama d'estats d'"FSMEst2"

A l'FSM, els primers 5 estats són els descrits als paràgrafs anterior i pertanyen a la part de recopilació de dades per al programa. "L0" és l'estat de repòs de l'FSM. Eixint solament d'aquest estat al donar-se una variable "start". "L1_aux" i "L2_aux" tenen el mateix funcionament que "L1" i "L2" respectivament, amb la diferència que "L2" és l'únic estat que habilita el compte. Els estats "L1" i "L1_aux" activen la lectura i mantenen també actiu el temporitzador.

El bucle queda tancat entre "L1" i "L2" fins que el temporitzador aplega al seu màxim. El fet de fer-ho d'aquesta manera, és a dir, fer una primera passada sols llegint i sense tindre en compte el primer valor és per poder verificar el programa. Com les dades d'entrada són la diferència entre l'anterior i l'actual, a la primera lectura no existeix dada anterior. Per tant, per evitar problemes s'ignora aquest valor. Els estats auxiliars no serien necessaris una vegada siga incorporat el programa a l'FPGA.

Una vegada finalitzat el bucle amb l'arribada del temporitzador al seu valor màxim es procedeix a enviar les dades recopilades al següent bloc. Hi ha 32 comptadors i, per tant, 32 valors que es poden enviar. Però, com bé s'ha detallat a l'apartat 6.2. Consideracions prèvies, el programa de compressió està dissenyat per a 17 valors. Les modificacions que puguen donar-se en aquest programa estan sent considerades i la quantitat de dades que s'envien és variable.

La selecció de la quantitat de valors que s'envien es fa mitjançant un parell de multiplexors i un comparador, tal i com es mostra a l'esquema del circuit electrònic de la Figura 22. Un dels multiplexors ("mux2") recorre els valors emmagatzemats als comptadors gràcies a un comparador ("MUXcomp") i a un comptador ("MUXcounter"). L'altre multiplexor es manté fix en un dels 32 comptadors. D'aquesta manera es compara cada valor amb la resta per buscar els majors primer.

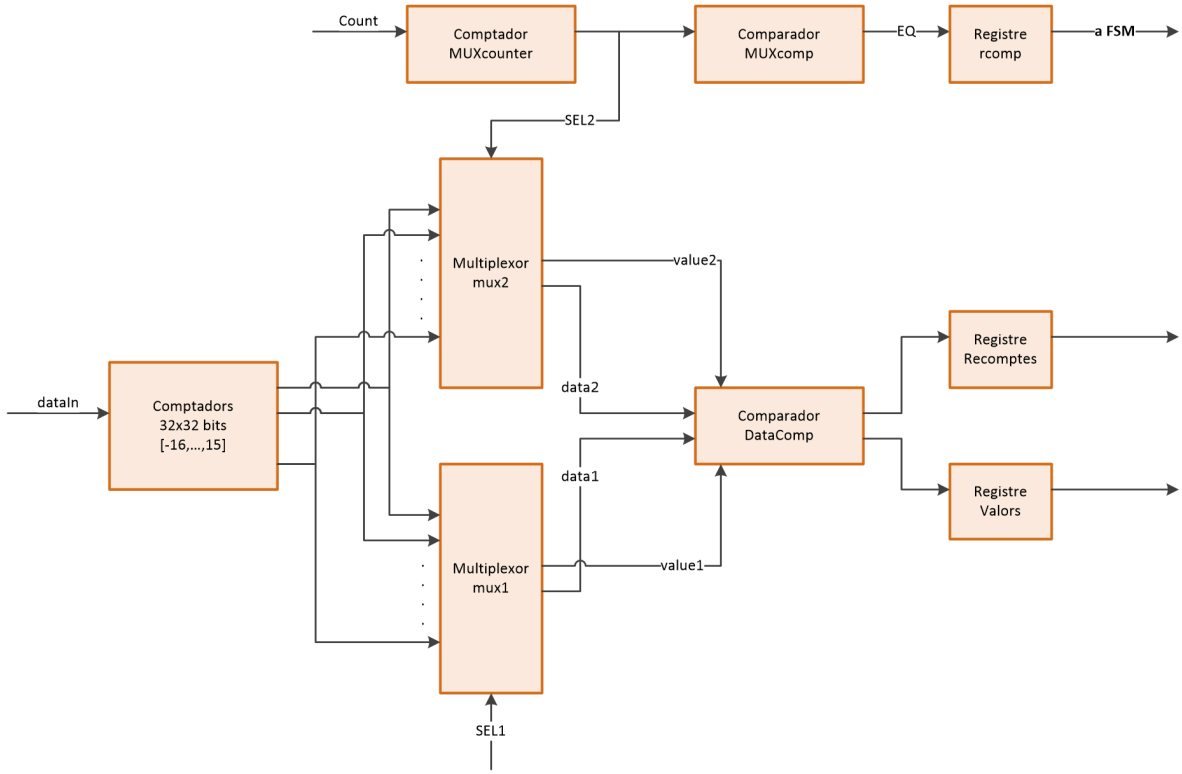

Figura 22. Esquema resum dels components electrònics del mòdul "estadistica2"

Depenent del resultat de les comparacions, el sistema passaria a l'estat "L4" o a l'"L5". L'estat "L4" seria el que fixa el valor amb que s'està fent la comparació. És a dir, s'ha trobat un valor de recompte major que amb el que s'està comparant, es posa aquest com a fixe i es compara amb la resta. Al seu torn, l'estat "L5" és el que va recorrent els valors dels comptadors dels recomptes augmentant el comptador "MUXcounter" (nom a la Figura 22).

Si "mux2" recorre tots els recomptes, significa que la dada que s'ha mantingut fixa és la major de totes. En aquest punt, l'estat passaria a ser "L6", on s'inicialitzen els selectors dels multiplexors, es registra el recompte que s'ha quedat com a major del conjunt, s'envia el valor d'entrada assignat a aquest a una FIFO i s'habilita el bit corresponent a una màscara.

Cal comentar que, en cas que hi haja dos nombres amb la mateixa probabilitat, el multiplexor comença recorrent els comptadors des del -16 fins al 15. Aquest va cercant un valor major (no igual o major); per tant, es queda abans el valor més menut. Per exemple, si -9 i 8 tenen la mateixa probabilitat, s'envia abans el -9 al següent bloc.

Hi ha un comptador més que no apareix a l'esquema. Aquest augmenta el seu valor quan l'FSM és a l'estat "L6", portant el recompte del nombre de valors que han sigut registrats. La seua eixida serà comparada amb la quantitat de valors desitjada per a fer l'arbre.

El bucle "L3-L4-L5-L6" es repeteix fins que el comptador comentat a l'anterior paràgraf assoleix la quantitat de valors que tindrà l'arbre. Aleshores, passarà a l'estat "L7", on es buiden els comptadors i registres i s'envia el senyal de finalització a l'FSM superior per poder començar el següent bloc. La Taula 2 mostra les eixides que s'activen amb cada estat de l'FSM d'aquest bloc.

Estat	enMUX	ENt	ENc	ENv	ENr	read	write	SEL1
L0	0	0	0	0	0	0	0	-16
L1_aux	0	1	0	0	0	1	0	-16
L2_aux	0	1	0	0	0	0	0	-16
L1	0	1	0	0	0	1	0	-16
L2	0	1	0	0	0	0	0	-16
L3	1	0	0	1	0	0	0	*
L4	1	0	0	0	0	0	0	vOut
L5	1	0	1	0	0	0	0	*
L6	1	0	0	0	1	0	1	-16
L7	0	0	0	0	0	0	0	-16

*Dependrà de l'estat anterior

Estat	add	compare	count	countValue	enMask	mode	clrT	clrC	endE
L0	0	0	0	0	0	0	0	0	0
L1_aux	0	0	0	0	0	0	0	0	0
L2_aux	0	0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0	0	0
L2	1	0	0	0	0	0	0	0	0
L3	0	1	0	0	0	0	1	0	0
L4	0	0	0	0	0	0	0	1	0
L5	0	0	1	0	0	0	0	0	0
L6	0	0	0	1	1	1	0	1	0
L7	0	0	0	0	0	0	1	0	1

Taula 2. Eixides de l'"FSMEst2" per estat actiu

Cal comentar, abans de continuar amb el següent apartat, la màscara esmentada en paràgrafs anteriors. Aquest mòdul no s'utilitza al programa del projecte, però donat que pertany a un conjunt major, és necessari per als programes ja existents a l'FPGA. Serveix per identificar quins valors del rang [-16,15] s'estan codificant, sent -16 el LSB i 15 el MSB

7.3. Mòdul de l'Arbre

Un detall dels mòduls que formen part d'aquest bloc es poden veure a la Figura 23. Aquest mòdul està compost pel mòdul principal "CalculPosicio2", on es troben els circuits d'electrònica digital, la màquina d'estats "FSMArbre2" i un multiplexor per a les habilitacions dels registres de la Figura 24, ja que poden ser habilitats des d'aquest bloc o des de l'anterior.

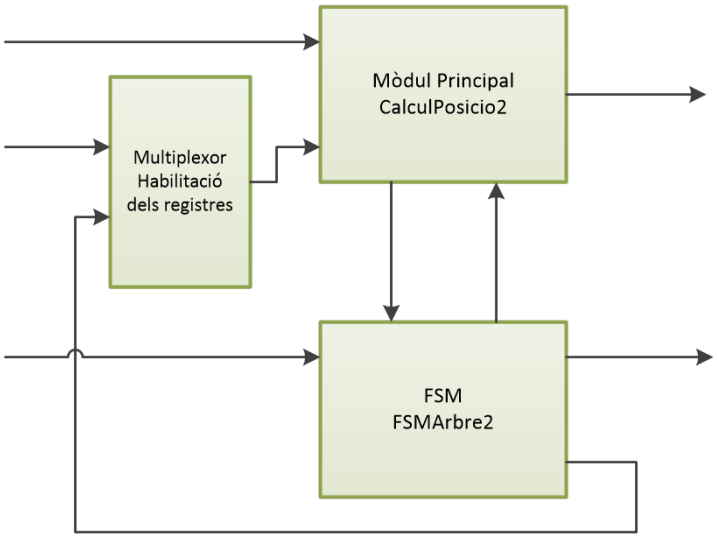

Figura 23. Esquema resum dels mòduls inclosos al mòdul de l'Arbre("Huff_tree2")

L'entrada de dades d'aquest bloc prové de l'eixida dels recomptes fets a l'anterior. Aquestes són emmagatzemades als registres de desplaçament que es mostren a la Figura 24, on es representen les interconnexions entre els components. Al bloc anterior s'utilitza una variable a l'estat "L6" que fa actuar aquests registres com si foren uns de desplaçament habituals. No obstant això, les comunicacions entre aquest són més complexes per a dur a terme la construcció de l'arbre. Permeten registrar dades de l'anterior, del següent i del següent del següent registre, a més de rebre i enviar dades a "l'exterior". Tres estructures de registres com aquest són els que seran utilitzats per a l'algorisme de construcció de l'arbre de Huffman. A la Figura 25 són els anomenats "Estructura dels 32 Registres de Recomptes/Tipus de Node/l'Altura de l'arbre".

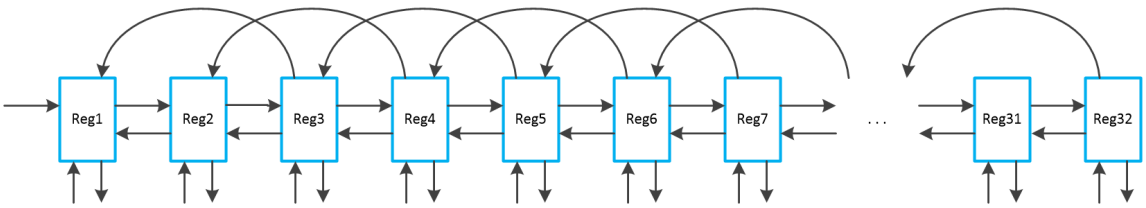

Figura 24. Interconnexions entre els registres per a la construcció de l'arbre

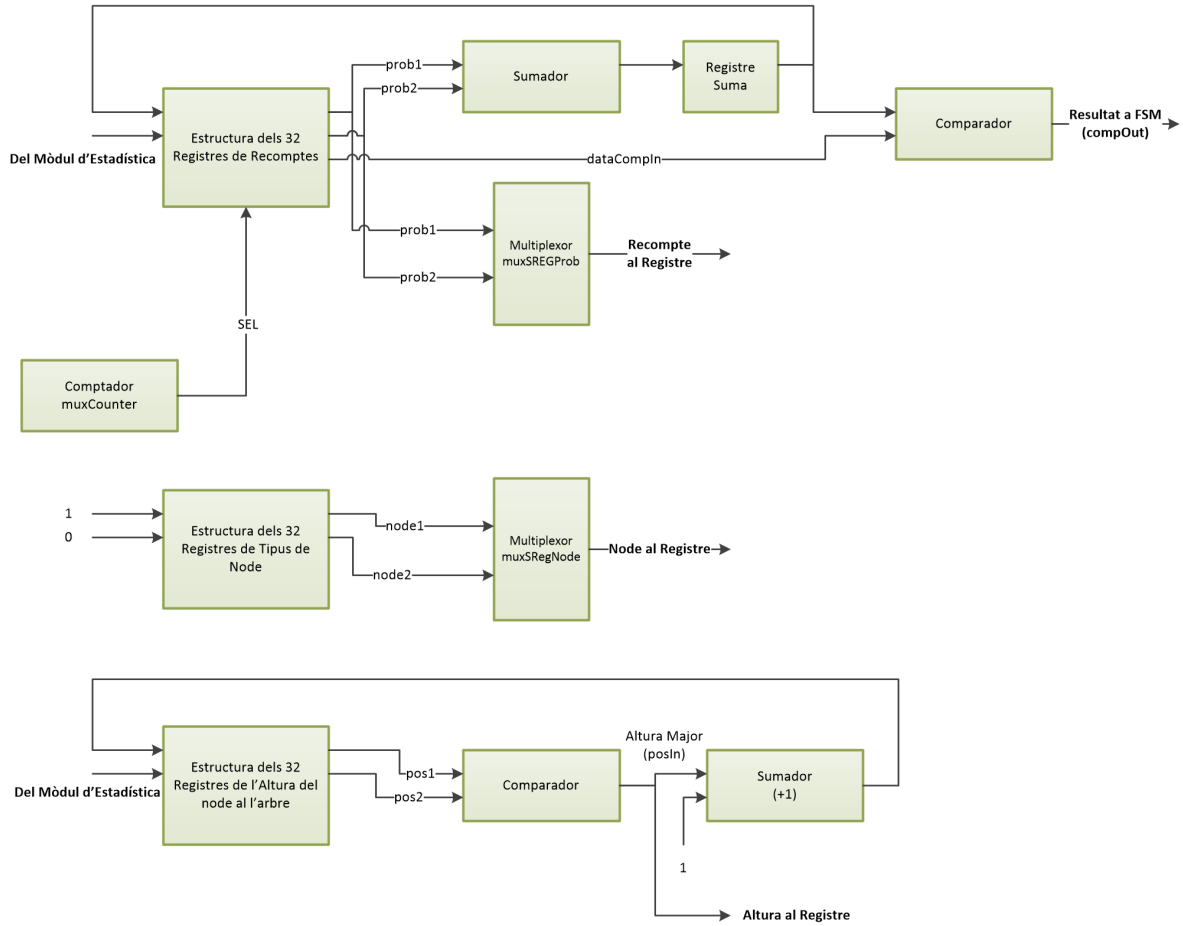

Figura 25. Esquema resum dels components electrònics del mòdul "CalculPosicio2"

Seguint l'explicació amb l'FSM (Figura 26), igual que abans, l'estat "A0" és el de repòs, del qual s'ix quan la corresponen variable de "Start" val 1. Començat el procés, a "A1" s'inicialitzen dos comptadors: un amb la quantitat d'elements que tindrà l'arbre i que serà l'encarregat de portar el compte dels valors romanents als registres vists a la Figura 24, i un altre per fer comparacions.

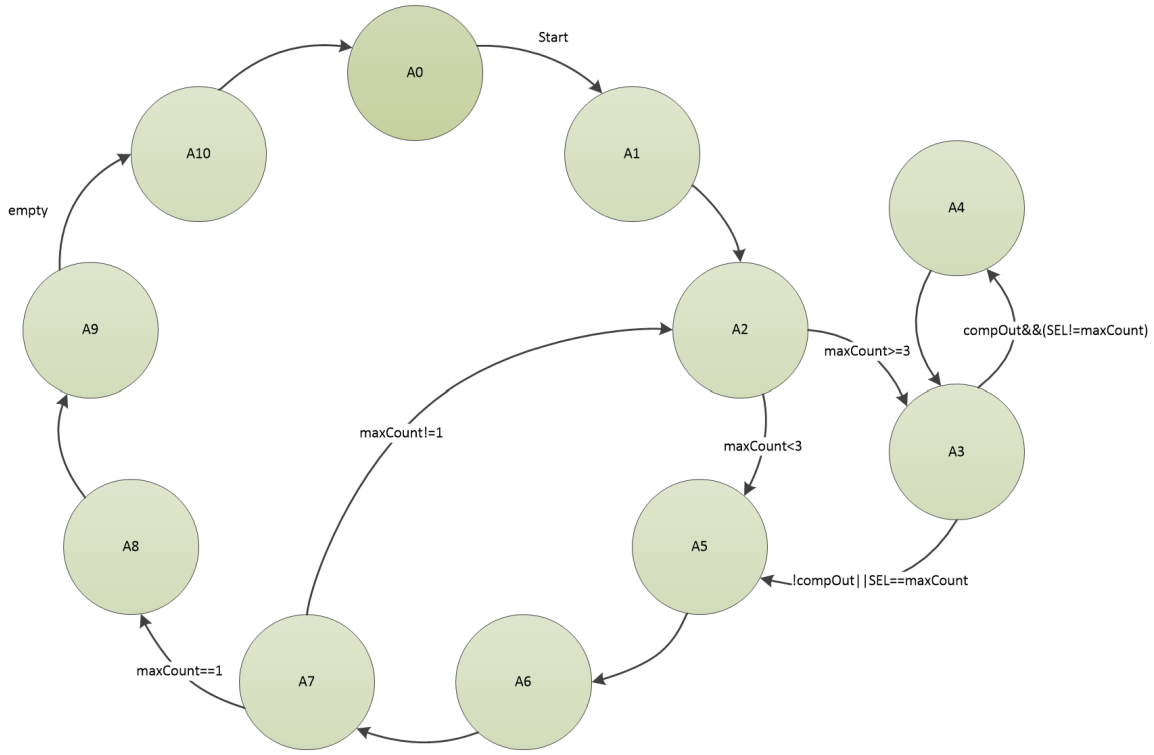

Figura 26. Diagrama d'estats d'"FSMArbre2"

A l'estat "A2" és on realitza la suma dels dos recomptes més menuts (al programa se'ls anomena com a probabilitats, malgrat que realment no ho siguen), registrant-se la suma. Depenent de la quantitat de valors romanents als registres es passarà als estats "A3" o "A5". Com s'aprecia a la Figura 26, si queden menys de 3 valors és quan es passaria a "A5".

En el cas en què es passe a l'estat "A3", es realitza la comparació entre el valor sumat a l'estat anterior i els valors als registres de la Figura 24 mitjançant un sistema similar al explicar al bloc anterior, on un multiplexor recorre la sèrie de registres i es van comparant aquests amb la suma. El multiplexor és el que es mostra a la Figura 27. En aquesta figura s'aprecia el detall de l'entrada i eixida d'aquesta estructura amb el bloc. És té una entrada comuna a tots els registres, que és la suma dels dos menors, i una eixida a través d'un multiplexor tal i com s'acaba d'explicar.

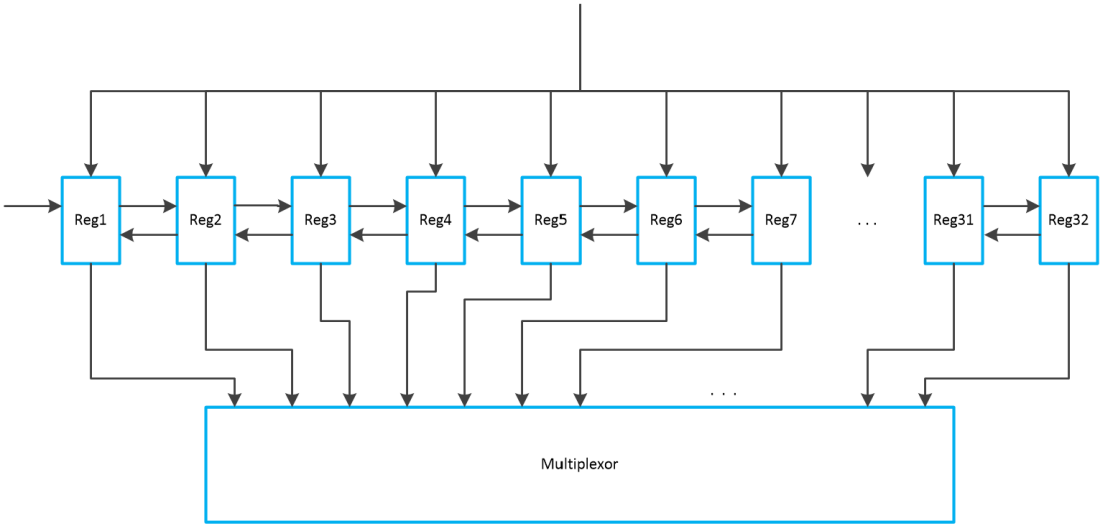

Figura 27. Connexions exteriors de les estructures de 32 registres

Aquesta comparació s'ha de realitzar perquè als registres de la Figura 24 i la Figura 27 els valors estan ordenats de menor a major, tal i com han anat entrant del bloc anterior. Per aquest motiu, a l'introduir un nou valor en aquests, s'ha d'ubicar on li corresponga. Aquesta cerca de la ubicació és el que defineix el bucle format per "A3" i "A4".

En aquest punt és on entra en joc el funcionament d'aquestes "estructures de registres" esmentades. Mitjançant un parell de multiplexors, s'aconsegueix reordenar tots els registres per tenir en compte dues coses: els dos valors més menuts ja no estan i s'incorpora un nou node fruit de la suma d'aquests. És per aquest motiu el fet que tinguen les connexions que s'aprecien a la Figura 24.

Com s'observa a la Figura 25, hi ha tres estructures com aquestes. La de recomptes és la que emmagatzema el valor de la suma. La de tipus de node indica per a cada node si és "pare" o no amb un 1 o un 0. Aquesta estructura és emplenada amb zeros al bloc anterior. Cada cop que entrada en una determinada ubicació un node que és suma de dos, es posa a 1. Per últim, l'estructura de l'altura ubica a la posició que pertoque la major altura dels dos valors sumats afegint-li un 1. Aquesta és emplenada al bloc anterior amb 1.

Si es donara el cas que el node entrant té la mateixa probabilitat que uns dels nodes que ja hi són a les estructures, l'entrant es considerarà sempre com a major. Açò és debut a que el programa cerca un valor major a l'entrant, obviant els que són iguals.

Una vegada acotada la ubicació del nou node, es passa als següents estats i es registren els dos recomptes que s'havien utilitzat per a la suma (els dos menors), així com les dades de les altres estructures de registres. També es registra per al més menut un 1, i per al segon un 0, a un quart registre de desplaçament.

Cal esmentar que el registre dels recomptes té una finalitat de comprovació i verificació del funcionament dels blocs. Aquests valors no tornen a ser utilitzats en el projecte. De fet, l'ISE elimina els registres del circuit final.

A la Figura 26 s'observa que hi ha un altre bucle comprès entre "A2" i "A7". Aquest manté tot el procés explicat fins ara fins que sols queda a les estructures de registres la informació associada a una sola dada, la qual serà l'"arrel" de l'arbre de Huffman. Totes aquestes dades són registrades a l'estat "A8" als registres de desplaçament pertinents.

Per últim, amb l'estat "A9" acabarien parcialment les funcions d'aquest bloc. En aquest estat, s'activa la variable que permet que el següent bloc comence i una variable que modifica el sentit dels registres de desplaçament utilitzats per emmagatzemar la informació de l'arbre de Huffman. Aquesta acció permet que el bloc de codificació puga extraure les dades començant per l'última que ha entrat, la de major probabilitat. És necessari que es faça d'aquesta manera perquè la codificació comença construint-se a partir de la base de l'arbre i no des de les "fulles".

L'FSM romandrà en aquest estat fins que li aplegue un senyal indicant que ja no queden dades als registres de desplaçament esmentats. La màquina d'estats acabaria la seua feina buidant els registres i els comptadors i passant al següent cicle de rellotge al seu estat de repòs.

La Taula 3 mostra els senyals actius a cada estat de l'FSM explicada.

Estat	enMUX	SELe	SELreg	ENps	ENp/ENn	ENrprob/ENrpos	ENrc/ENrn	init	init2
A0	0	0	0	0	0	0	0	0	0
A1	1	1	0	0	0	0	0	1	1
A2	1	1	0	1	0	0	0	1	0
A3	1	1	0	0	0	0	0	0	0
A4	1	1	0	0	0	0	0	0	0
A5	1	1	0	0	0	1	1	0	0
A6	1	1	1	0	0	1	1	0	0
A7	1	1	0	0	1	0	0	0	0
A8	1	1	0	0	0	1	1	0	0
A9	1	1	0	0	0	0	0	0	0
A10	0	1	0	0	0	0	0	0	0

Estat	suma	compara	compta	descompta	costatIn	shiftmode	clr	estat9
A0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0
A2	1	0	0	0	0	0	0	0
A3	0	1	0	0	0	0	0	0
A4	0	0	1	0	0	0	0	0
A5	0	0	0	1	1	0	0	0
A6	0	0	0	0	0	0	0	0
A7	0	0	0	0	0	0	0	0
A8	0	0	0	0	0	0	0	0
A9	0	0	0	0	0	1	0	1
A10	0	0	0	0	0	0	1	0

Taula 3. Eixides d'"FSMArbre2" per a l'estat actiu

7.4. Mòdul de Codificació

El mòdul de Codificació consta dels mòduls presentats a la Figura 28. Igual que als casos anteriors, es troba el mòdul principal "CodiArbre2, l'FSMA "FSMCodi2" i un comptador i un comparador.

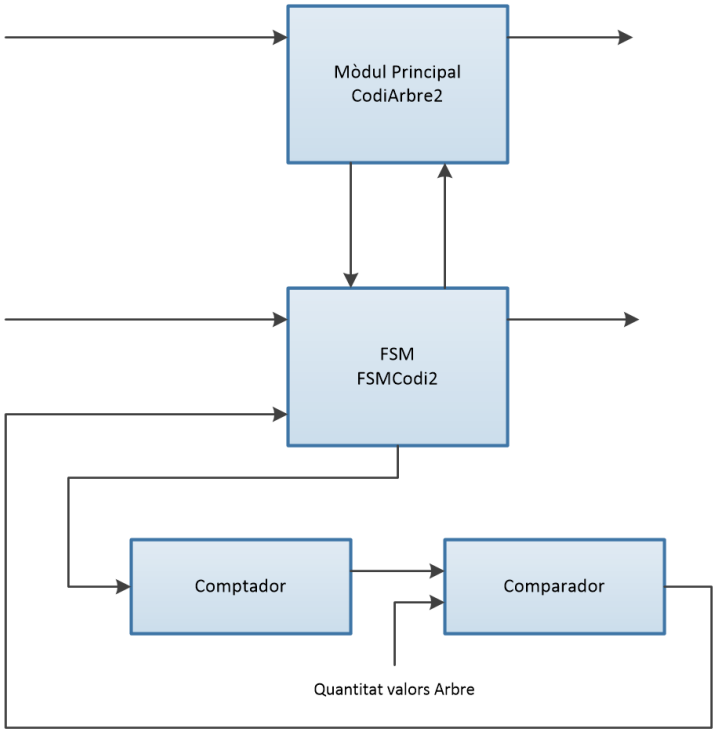

Figura 28. Esquema resum dels mòduls inclosos al mòdul de Codificació ("Codificacio2")

L'FSM és representada a la Figura 29. Comença amb un estat "C0" de repòs del qual ix a l'activar-se la variable "Start" corresponent. Per facilitar l'explicació, es proporciona la Figura 30 amb els circuits digitals més importants utilitzats que, com s'observa, s'estructuren en tres parts. Un circuit per a les dades d'altura, que determinaran el nombre de bits; un per al tipus de node, que dirigirà el camí de l'FSM, i un per al costat del node, que permetrà construir la codificació.

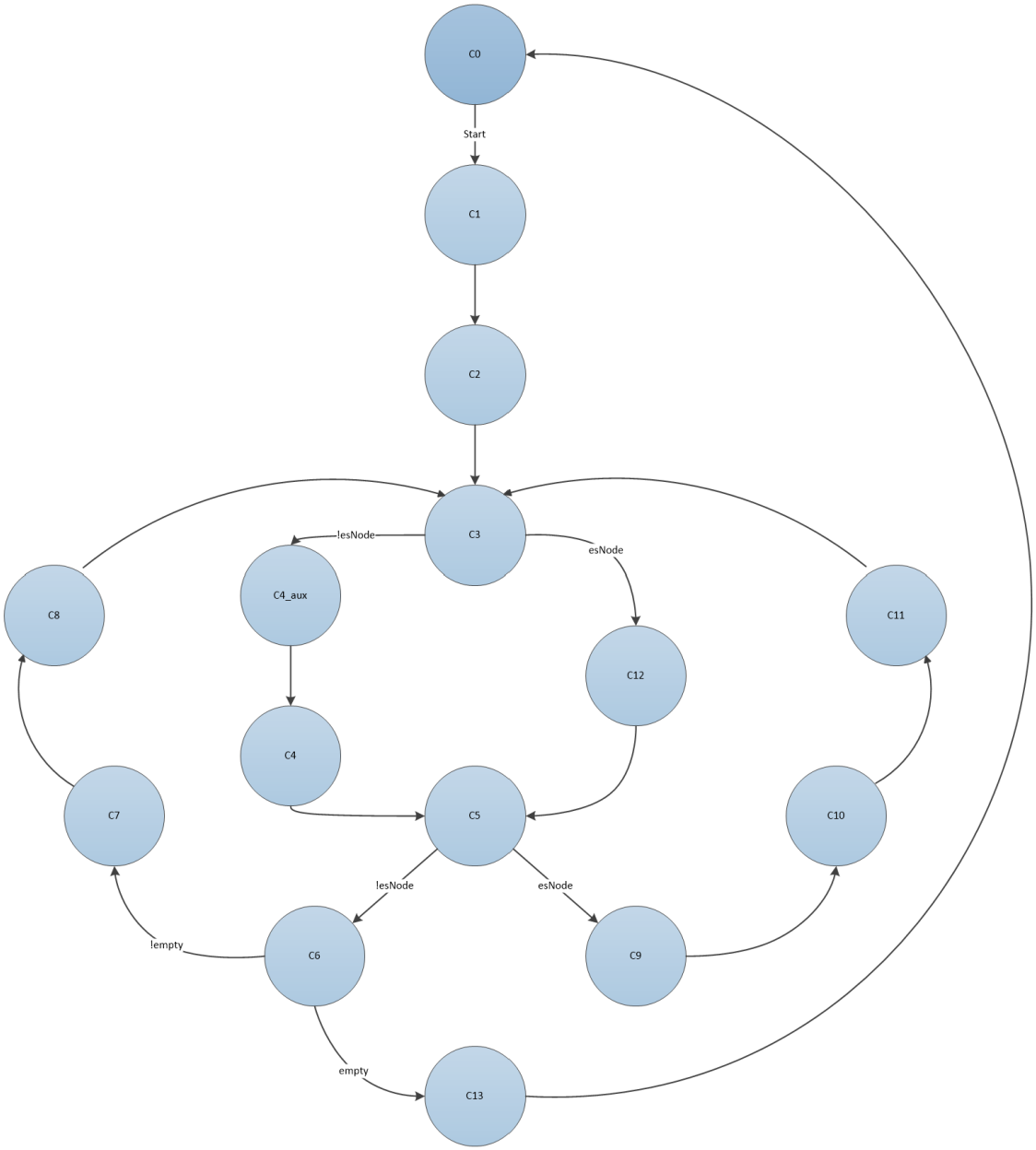

Figura 29. Diagrama d'estats d'"FSMCodi2"

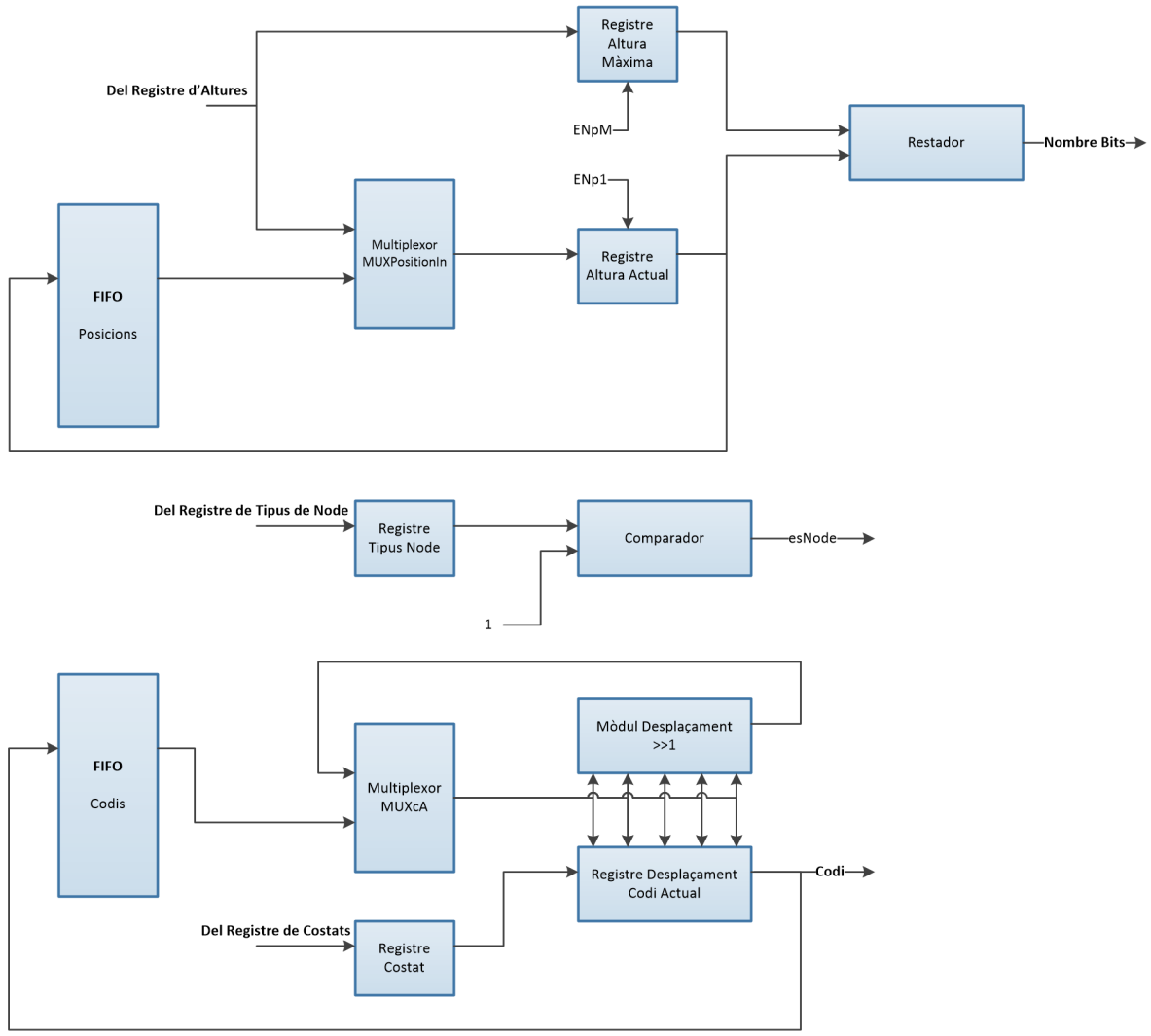

Figura 30. Esquema resum dels components electrònics de "CodiArbre2"

L'estat "C1" emmagatzema a "Registre Altura Actual" i "Registre Altura Màxima" (Figura 30) l'altura corresponent al node "arrel". També es registra el bit de "branca" i la naturalesa del node. Al següent estat, es repeteixen les accions anteriors, emmagatzemant-se la informació pertanyent al primer node "fill" de "l'arrel", el segon node amb el recompte major. A més, s'extrau de la FIFO utilitzada al primer bloc de tots el primer valor emmagatzemat, que serà el més probable de la cua.

L'estat "C3" junt amb el "C5" serveixen d'eix central de l'FSM. Una vegada queda tot registrat als dos estats anteriors, es comprova amb un comparador la naturalesa del primer node "fill" i s'incorpora el bit de "branca" al registre de desplaçament "Codi Actual" que apareix a la Figura 30.

En aquest punt s'entra a una divergència. Si el node resultara ser una "fulla", seria enviat als registres d'eixida a l'estat "C4", passant per l'estat "C4_aux", que fa la petició de lectura a la FIFO on es troben els valors. A "C4" a més, es registren també les dades del segon node "fill" tal i com s'ha fet als estats "C1" i "C2", i es realitza un desplaçament sobre "Codi Actual" cap a la dreta d'un bit per tornar a ser el codi del node "pare".

Si a la comparació feta a l'estat "C3" s'obtinguera que el node no és una "fulla", s'escriuria a unes FIFO les dades pertinents del acumulat fins al moment i de l'altura del node que s'està tractant. En aquest estat ("C12") no es llig un valor de la FIFO, ja que no s'ha enviat cap dada als registres d'eixida i els valors sols pertanyen a les "fulles". Es manté el valor llegit anterior fins que aparega el pròxim node "fulla". Igual com a l'estat "C4", es registren les dades del següent node.

Passada la divergència, es torna a un estat on es torna comprova la naturalesa del node, en aquest cas del segon "fill" que acaba de ser registrat als estats anteriors. L'estat "C5" té exactament les mateixes eixides que "C3", sols que no condueix als mateixos estats. Així, al sistema torna a aparèixer una altra divergència segons el tipus de node. Si fóra node "fulla, s'aniria a l'estat "C6", que és com "C4" però sense registrar cap nou valor, solament l'escriptura als registres d'eixida. Per al cas contrari, es tindria el mateix entre "C12" i "C9", quedant-se l'escriptura també.

Continuant primer pel cas on és "fulla", la ruta oberta per "C6", l'FSM entraria a l'estat "C7". Es llegeixen totes les FIFO utilitzades fins ara i es passa al següent estat. A continuació, es registren des del bloc anterior el tipus de node i el bit de "branca", i des de les FIFO, el valor següent, l'altura i el codi del pare del node que s'està estudiant.

El registre de desplaçament "Codi Actual" funciona també com un registre qualsevol. L'entrada per al bit de "branca" podria considerar-se una entrada en sèrie i l'entrada des del multiplexor "MUXca" (Figura 30), en paral·lel. D'aquesta última forma és com li arriben les dades del mòdul que desplaça el codi i de la FIFO corresponent. Completat tot açò, es tornaria a l'estat "C3" per fer-li la comparativa al nou node.

Continuant ara a partir de l'estat "C9", l'estat "C10" fa les mateixes funcions que el "C7". Però com l'anterior valor no era una "fulla", s'ha de mantenir el valor extret de la FIFO del primer bloc. És a dir, no es fa la petició de lectura d'aquesta. Així, el següent estat fa el corresponent al "C8", però sense registrar cap valor provinent d'aquesta FIFO.

El bucle que formen els estats del "C3" al "C8"/"C11" s'acaba quan no queden dades als registres de desplaçament. En aquest punt, es passaria a un estat on es buiden els registres i comptadors utilitzats i s'activa un senyal indicant que la màquina d'estats ha acabat la seua feina. La variable que indica que no queden dades als registres de desplaçament és la que s'envia també al bloc anterior i que ha sigut explicat a l'apartat corresponent.

La Taula 4 mostra un resum dels senyals que es donen a cada estat.

Estat	enC	ENr	ENp1	ENpM	ENn1	ENc1	ENca	ENv	readV	readFIFO	writeFIFO
C0	0	0	0	0	0	0	0	0	0	0	0
C1	1	1	1	1	1	1	0	0	1	0	0
C2	1	1	1	0	1	1	1	1	0	0	0
C3	1	0	0	0	0	0	1	0	0	0	0
C4_aux	0	0	0	0	0	0	0	0	1	0	0
C4	1	1	0	0	1	1	1	1	0	0	0
C5	1	0	0	0	0	0	1	0	0	0	0
C6	1	0	0	0	0	0	0	0	0	0	0
C7	1	0	0	0	0	0	0	0	1	1	0
C8	1	1	1	0	1	1	1	1	0	0	0
C9	1	0	0	0	0	0	0	0	0	0	1
C10	1	0	0	0	0	0	0	0	0	1	0
C11	1	1	1	0	1	1	1	0	0	0	0
C12	1	1	0	0	1	1	1	0	0	0	1
C13	0	0	0	0	0	0	0	0	0	0	0

Estat	SELf	SELfP	SELfC	SELd	comparaN	res	wrLUT	clr	clrpM	estat13
C0	0	0	0	0	0	0	0	0	0	0
C1	1	0	0	0	0	0	0	0	0	0
C2	1	0	0	0	0	0	0	0	0	0
C3	1	1	0	0	1	0	0	0	0	0
C4_aux	1	0	0	0	0	0	0	0	0	0
C4	1	1	0	1	0	1	1	0	0	0
C5	1	1	0	0	1	0	0	0	0	0
C6	1	1	0	0	0	1	1	0	0	0
C7	1	1	0	0	0	0	0	0	0	0
C8	1	1	1	1	0	0	0	0	0	0
C9	1	1	0	0	0	0	0	0	0	0
C10	1	1	0	0	0	0	0	0	0	0
C11	1	1	1	1	0	0	0	0	0	0
C12	1	1	0	1	0	0	0	0	0	0
C13	0	0	0	0	0	0	0	1	1	1

Taula 4. Eixides d'"FSMCodi2" per a l'estat actiu

Queda solament explicar el funcionament del mòdul dels registres d'eixida del programa. A aquests apleguen els codis, el valor corresponent del rang seleccionat a l'inici i el nombre de bits del codi. Aquest últim nombre ve donat per la resta entre l'altura màxima que es dóna a l'arbre i l'altura del node que s'està enviant.

Aquest mòdul de registres compta amb 64 registres per a les dades comentades i un mòdul per a l'habilitació d'aquests. Els 64 registres estan repartits en dos mòduls diferents: 32 per als codis i 32 per al nombre de bits. L'esquema del funcionament pot veure's a la Figura 31. El valor al qual s'associen les dades, és comparat a l'inici. Depenent del resultat, el codi i el nombre de bits que s'estan enviant s'emmagatzemaran en els seus corresponents registres. Aquestes dades són les que posteriorment aniran a la "Look-Up Table" ja implementada al mòdul de compressió de l'FPGA. Per tal d'accedir-hi és pel que s'ha programat al primer bloc una màscara.

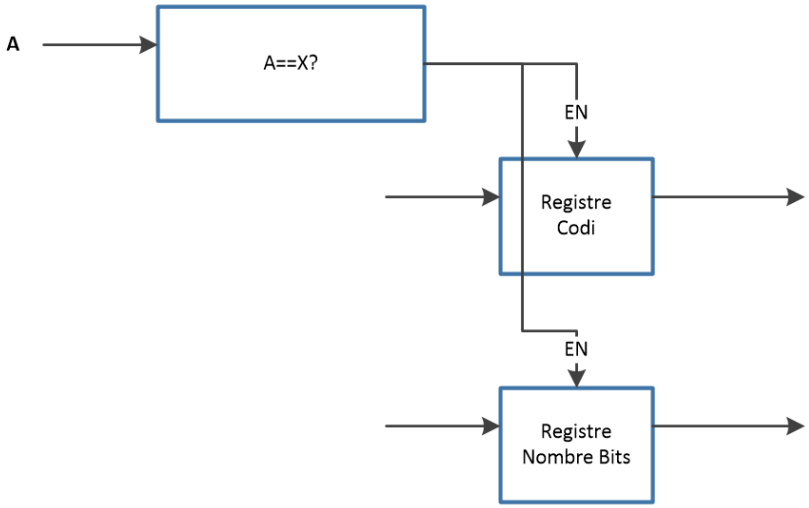

Figura 31. Esquema de funcionament dels registres d'eixida

8. RECURSOS UTILITZATS DE L'FPGA

Un dels objectius del projecte és l'estudi de la viabilitat del programa per ser implementat a l'actual FPGA. Aquesta es troba ja saturada, és a dir, la major part dels seus recursos estan sent utilitzats. És per aquest motiu que en el disseny del present projecte s'ha intentat reduir l'ús dels components de la targeta. Un clar exemple d'açò ha sigut la decisió de codificar l'arbre sols amb les dades d'un canal en comptes d'utilitzar els 12. La Figura 32 i la Figura 33 mostren una taula dels recursos de lògica utilitzats de l'FPGA i la RAM respectivament.

Tal i com s'ha descrit a l'apartat 5.1. Targeta FPGA, el bloc més bàsic d'una Virtex-6 és una "Slice". En el cas de la Virtex-6 emprada està formada per una LUT (generació de funcions lògiques) més dos registres tipus D.

```
Slice Logic Utilization:
  Number of Slice Registers:                 5,100
    Number used as Flip Flops:               3,693
    Number used as Latches:                  1,407
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                      4,329
    Number used as logic:                    4,296
      Number using O6 output only:           2,694
      Number using O5 output only:             990
      Number using O5 and O6:                  612
      Number used as ROM:                        0
    Number used as Memory:                       0
    Number used exclusively as route-thrus:     33
      Number with same-slice register load:      0
      Number with same-slice carry load:        33
      Number with other load:                    0
```

Figura 32. Recursos de lògica utilitzats al projecte

```
Specific Feature Utilization:
  Number of RAMB36E1/FIFO36E1s:                  0
  Number of RAMB18E1/FIFO18E1s:                  3
    Number using RAMB18E1 only:                  3
    Number using FIFO18E1 only:                  0
```

Figura 33. Recursos de memòria utilitzats al projecte

Per poder estudiar la viabilitat de la implantació del projecte a l'FPGA actual en funcionament amb el programa de compressió, es mostren a continuació la Figura 34 i la Figura 35. Amb aquestes dades és possible calcular el percentatge d'ocupació que suposaria el circuit del present projecte en l'FPGA. Quedarien de la següent manera els recursos més determinants:

- Slice Registers: 1,7% (5100 de 301440)
- Slice LUTs: 2,9% (4329 de 150720)
- RAM18E1: 0,3% (3 de 832)

Sabent açò, es pot concloure que el nou mòdul no suposaria un problema pel que fa a la inserció en l'FPGA junt als programes de compressió actuals. Haguera sigut un cas preocupant si l'ocupació haguera aplegat a un 60% aproximadament, on es veurien afectats els camins crítics del circuit (de totes les connexions entre eixida del biestable a entrada de biestable) per l'alta ocupació dels recursos d'interconnexió.

```
Slice Logic Utilization:
  Number of Slice Registers:                36,627 out of 301,440   12%
    Number used as Flip Flops:              36,415
    Number used as Latches:                      6
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:              206
  Number of Slice LUTs:                     47,843 out of 150,720   31%
    Number used as logic:                   45,647 out of 150,720   30%
      Number using O6 output only:          24,903
      Number using O5 output only:           2,100
      Number using O5 and O6:               18,644
      Number used as ROM:                        0
    Number used as Memory:                     744 out of  58,400    1%
      Number used as Dual Port RAM:            556
        Number using O6 output only:             4
        Number using O5 output only:             8
        Number using O5 and O6:                544
      Number used as Single Port RAM:            0
      Number used as Shift Register:           188
        Number using O6 output only:           172
        Number using O5 output only:             0
        Number using O5 and O6:                 16
    Number used exclusively as route-thrus:  1,452
      Number with same-slice register load:    967
      Number with same-slice carry load:       485
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                18,251 out of  37,680   48%
```

Figura 34. Recursos de lògica utilitzats a l'FPGA actual

```
Specific Feature Utilization:
  Number of RAMB36E1/FIFO36E1s:                103 out of     416   24%
    Number using RAMB36E1 only:                103
    Number using FIFO36E1 only:                  0
  Number of RAMB18E1/FIFO18E1s:                 45 out of     832    5%
    Number using RAMB18E1 only:                 45
    Number using FIFO18E1 only:                  0
```

Figura 35. Recursos de memòria utilitzats a l'FPGA actual

9. VERIFICACIÓ DEL FUNCIONAMENT DEL PROJECTE

Diverses simulacions han sigut realitzades per comprovar el correcte funcionament del programa. Entre elles, cada mòdul ha sigut comprovat per separat abans d'ajuntar-los tots en un mateix programa. No obstant, aquestes simulacions no seran mostrades al treball, ja que són simples comprovacions i no aporten més del que ja s'ha explicat en apartats anteriors. En el present apartat es mostraran els resultats obtinguts al processar arbres de diferents grandàries. Açò permetrà comprovar que efectivament l'arbre està realitzant-se tant amb pocs valors com amb molts. A més, ja que el sistema de compressió ja implementat està dissenyat per a 17 valors, s'ha realitzat també aquesta simulació.

Per agilitzar la comprovació, s'emprarà el mateix programa de Python utilitzat per obtindre el rang d'elements que es tenen en compte. Aquest programa construeix també un arbre de Huffman i realitza la pertinent codificació. L'altra manera que hi havia de comprovar que els resultats eren els correctes era fent els arbres a mà. Òbviament, quan els arbres contenen molts nodes, acaba sent una tasca prou laboriosa.

8.1. Estimació del Temps de Recompte en Python

El temps de processat pot ser elevat si s'ha de tenir en compte la totalitat d'esdeveniments que es produeixen. És per tant necessari analitzar l'estabilitat de l'arbre de Huffman per veure si és possible reduir el nombre d'esdeveniments que s'ha de tenir en consideració. Per poder fer açò, s'ha fet ús del programa de Python, on s'han obtingut les codificacions per a cert nombre d'esdeveniments, tal i com es mostra a la Taula 5 i a la Taula 6. En aquestes dues es mostra a l'esquerra els codis que s'obtindrien si es consideraren tots els esdeveniments. En verd queden marcats els codis que sí coincideixen.

S'ha decidit realitzar dues taules diferents. La Taula 5 és per a un arbre de 17 valors. Es recorda que el mòdul de compressió ja implementat està definit per a aquest nombre de codis. Per tant, una de les opcions ha sigut fer aquesta taula.

Codis	Valors	Nombre d'events processats al càlcul de probabilitats									
		100	125	150	175	200	225	250	275	300	325
00101011111	-8	00101011111	00101011111	00101011111	00101011111	00101011111	00101011111	00101011111	00101011011	00101011011	00101011011
00101010100	-7	00101011101	00101011101	00101010111	00101010111	00101011101	00101011101	00101010101	00101010101	00101010101	00101010100
00101010011	-6	00101010010	00101010010	00101010010	00101010011	00101010011	00101010011	00101010011	00101010011	00101010011	00101010011
0010101011	-5	0010101010	0010101011	0010101110	0010101010	0010101010	0010101011	0010101110	0010101011	0010101011	0010101011
001010110	-4	001010110	001010110	001010110	001010110	001010110	001010110	001010110	001010111	001010111	001010111
001011	-3	001011	001011	001011	001011	001011	001011	001011	001011	001011	001011
00100	-2	0011	0011	0011	0011	00100	00100	00100	00100	00100	00100
000	-1	000	000	000	000	000	000	000	000	000	000
1	0	1	1	1	1	1	1	1	1	1	1
01	1	01	01	01	01	01	01	01	01	01	01
0011	2	00100	00100	00100	00100	0011	0011	0011	0011	0011	0011
0010100	3	0010100	0010100	0010100	0010100	0010100	0010100	0010100	0010100	0010100	0010100
0010101000	4	0010101000	0010101000	0010101000	0010101000	0010101000	0010101000	0010101000	0010101000	0010101000	0010101000
0010101110	5	0010101011	0010101010	0010101010	0010101110	0010101011	0010101010	0010101011	0010101100	0010101100	0010101100
00101010010	6	00101010011	00101010011	00101010011	00101010010	00101010010	00101010010	00101010010	00101010010	00101010010	00101010010
00101010101	7	00101011100	00101011100	00101010110	00101010110	00101011100	00101011100	00101010100	00101010100	00101010100	00101010101
00101011110	8	00101011110	00101011110	00101011110	00101011110	00101011110	00101011110	00101011110	00101011010	00101011010	00101011010

Taula 5. Codis segons el nombre d'esdeveniments per a un arbre de 17 valors

La Taula 6 està feta per a un arbre de 32 valors. Els motius d'aquesta taula són en part tenir-ne una segona amb la qual anar comparant els resultats amb l'anterior. A més, al ser l'arbre més gran que pot donar-se és possible que tinga més inestabilitat al considerar més esdeveniments.

Codis	Valors	Nombre d'events processats al càlcul de probabilitats									
		100	125	150	175	200	225	250	275	300	325
001011110001	-16	001011110001	001011110001	001011110001	001011110001	001011100101	001011100101		001011110001	001011110001	001011110001
001011100011	-15	001011100011	001011100011	001011110000	001011100011	001011100100	001011100100	001011100011	001011100011	001011100011	001011100011
001011100010	-14	001011010110	00101101	001011100010	001011100010	001011100010	001011010101	001011010101	001011100010	001011100010	001011100010
001011010100	-13	001011010111	001011010100	001011010100	00101010100	001011010100	001011010011	001011010011	001011001011	001011001100	001011001100
00101111101	-12	00101111111	001011000101	00101111101	00101111101	00101111101	00101111101	00101111101	00101111101	00101111101	00101111101
00101111100	-11	00101111011	001011000100	00101111100	00101111011	00101111011	00101111100	00101111011	00101111100	00101111100	00101111100
00101111001	-10	00101111001	00101111001	00101111001	00101111001	00101111001	00101111001	00101111001	00101111001	00101111001	00101111001
00101110010	-9	00101110011	00101110011	00101110010	00101110011	00101111000	00101111000	00101110011	00101110011	00101110010	00101110010
00101110000	-8	00101110000	00101110000	00101110000	00101110000	00101110000	00101110000	00101110000	00101110000	00101110000	00101110000
0010111111	-7	00101101000	00101101000	00101101000	00101101000	00101101000	00101101000	00101101000	00101100100	00101100100	0010111111
0010111011	-6	0010111010	0010111010	0010111010	0010111011	0010111011	0010111011	0010111011	0010111011	0010111011	0010111011
0010110000	-5	0010110000	0010110000	0010110001	0010110000	0010110000	0010110001	0010110001	0010110000	0010110000	0010110000
001011001	-4	001011001	001011001	001011001	001011001	001011001	001011001	001011001	001011010	001011010	001011010
0010100	-3	0010100	0010100	0010100	0010100	0010100	0010100	0010100	0010100	0010100	0010100
00100	-2	0011	0011	0011	0011	00100	00100	00100	00100	00100	00100
000	-1	000	000	000	000	000	000	000	000	000	000
1	0	1	1	1	1	1	1	1	1	1	1
01	1	01	01	01	01	01	01	01	01	01	01
0011	2	00100	00100	00100	00100	0011	0011	0011	0011	0011	0011
0010101	3	0010101	0010101	0010101	0010101	0010101	0010101	0010101	0010101	0010101	0010101
001011011	4	001011011	001011011	001011011	001011011	001011011	001011011	001011011	001011011	001011011	001011011
0010110001	5	0010110001	001011111	0010110000	0010110001	0010110001	0010110000	0010110000	0010110001	0010110001	0010110001
0010111010	6	0010111011	0010111011	0010111011	0010111010	0010111010	0010111010	0010111010	0010111010	0010111010	0010111010
00101101000	7	0010111110	00101100011	0010111111	001011111	0010111111	0010111111	0010111111	0010111111	0010111111	00101100100
00101101011	8	00101101010	00101101011	0001101011	00101101011	00101101011	00101101011	00101101011	00101100111	00101100111	00101100111
00101110011	9	00101110010	00101110010	00101110011	00101110010	0010111001	00101110011	00101110010	00101110010	00101110011	00101110011
00101111010	10	00101111010	00101111010	00101111010	00101111010	00101111010	00101111010	00101111010	00101111010	00101111010	00101111010
00101111011	11	00101111110	00101111011	00101111011	00101111100	00101111100	00101111011	00101111100	00101111011	00101111011	00101111011
001011010010	12	001011010010	001011010010	001011010010	001011010010	001011010010	001011010010	001011010010	001011001010	001011001010	001011001010
001011010011	13	001010010101	001011010011	001011010011	001011010011	001011010011	001011010100	001011010100	001011001100	001011001011	001011001011
001011010101	14	001011100010	001011100010	001011010101	001011010101	001011010101	001011100010	001011100010	001011001101	001011001101	001011001101
001011110000	15	001011110000	001011110000	001011100011	001011110000	001011100011	001011100011	001011110000	001011110000	001011110000	001011110000
	16							001011110001			

Taula 6. Codis segons el nombre d'esdeveniments per a un arbre de 32 valors

Abans d'analitzar les taules, caldria fer una xicoteta consideració. Els resultats del projecte estan donant-se en base a uns fitxers de dades d'una operació de calibratge que s'ha realitzat. No obstant, el rang és teòricament vàlid per a qualsevol altra font de radiació seleccionada. A més, les dades poden canviar de probabilitat, però la majoria d'aquestes, per la forma que té l'ona d'energia de l'experiment, es concentraran al voltant del 0.

En ambdues taules pot observar-se que a partir de 200 esdeveniments, a totes les simulacions que s'ha fet, els codis des de -3 fins a 4 són iguals que considerant totes les dades. Aquesta informació és molt important, ja que com pot veure's a la Figura 36, pràcticament el 99% de les dades es troba dins d'aquest rang. La figura en qüestió és per a un arbre de 32 valors considerant 200 esdeveniments.

Aquestes dades són igualment vàlides per a un arbre de 17 valors, ja que les dades són les mateixes, sols que a aquest segon se n'agafen menys.

```
print(tree)
```

```
[(-16, 9.817495546309012e-05), (-15, 0.00010557894328939563), (15, 0.00010586741034756338), (-14, 0.000118656116593
0002), (14, 0.00012182925423284542), (-13, 0.00012836784088464477), (13, 0.00012942555343126277), (12, 0.00014221425
96766996), (-12, 0.00014250272673486732), (11, 0.00016086846277154726), (-11, 0.00016173386394605O5), (10, 0.000176
92646234288522), (-10, 0.00017798417488950O3), (-9, 0.00020173462934531l5), (9, 0.00020260003051981475), (-8, 0.000
23788916730233588), (8, 0.00024058152651190153), (-7, 0.00028971708208647454), (7, 0.00029433255501715O5), (-6, 0.0
0038712279206111l3), (6, 0.00039087286381729l), (5, 0.00058731893042953l2), (-5, 0.00058808817591798l5), (4, 0.0009
99153733807022), (-4, 0.00107415516893063l2), (3, 0.00551414397256255l), (-3, 0.00558828000651166l), (-2, 0.0529926
4860933397), (2, 0.053022168404953135), (-1, 0.23812830649355696), (1, 0.23919380765074255), (0, 0.3960395972961406
)]
```

Figura 36. Probabilitats dels primers 32 valors, calculat amb 200 esdeveniments. Entre parèntesi (valor, probabilitat sobre 1)

Aquesta ja pot ser raó més que suficient per considerar que a partir del temps que es tarde en processar 200 esdeveniments l'arbre és estable. La resta de codis varien. No obstant, per al cas de l'arbre de 17 valors, els codis diferents mantenen el mateix nombre de bits, per la qual cosa la compressió és la mateixa. Per a l'altre arbre, al ser més gran, la variabilitat entre els codis és major, però el percentatge que representen respecte el total i amb la quantitat de bits amb què estan sent codificats, és pot assegurar que la influència amb el resultat final no pot diferir molt.

8.2. Banc de proves

Per a la realització de simulacions de programes en llenguatges HDL, és necessari la utilització dels anomenats bancs de proves. Aquests són una mena de script on s'instancien els mòduls que s'han de simular. Al contrari que aquests mòduls, el tipus de llenguatge utilitzat (és Verilog igualment) és més abstracte i menys centrat en hardware. En aquest script s'introdueixen estímuls corresponents a les entrades dels mòduls per veure el comportament d'aquests.

Per al cas d'aquest projecte en concret, s'ha instanciat el mòdul principal del programa i un mòdul de lectura necessari per poder llegir els fitxer de dades. Aquest últim no serà necessari una vegada implementat el circuit a l'FPGA. Un programa de Python és l'encarregat de donar format de fitxer de text a un fitxer d'eixida de l'experiment, on les dades dels PMTs vénen recollides en columnes que representen cadascun dels 12 que hi ha. La columna que es llig és el que selecciona el multiplexor a l'entrada del mòdul que s'ha explicat a 6.4.1. .

A la instanciació dels mòduls se li sumen dos "procedural blocks". Per una banda, la creació del rellotge del que s'utilitzarà a la simulació i que, per al sistema ha de ser de 40 MHz. I per una altra banda, el bloc on es detallen els passos que segueix la simulació. La simulació ve marcada per estímuls que es corresponen amb les entrades dels mòduls que s'està simulant. Barrejant un canvi d'estímuls i moments d'espera del sistema, es conforma la simulació.

Aquesta part pot veure's en més detall pel que fa a la programació a l'Annex núm. 2Annex núm. 2Annex núm. 2Annex núm. 2.

8.3. Simulació del circuit implementat

Es presenten a continuació una sèrie de captures de les simulacions realitzades i el resultat que dóna el programa de Python. A les simulacions es representarà per a cada cas una visió general de la simulació, els codis que s'obtenen i el nombre de bits d'aquest.

Com s'ha esmentat a l'apartat corresponent, totes les simulacions ha sigut realitzades amb ModelSim. La quantitat d'esdeveniments considerats augmenta el temps de simulació. Per aquest motiu, s'ha decidit fer les proves per a 20 esdeveniments, tant a ModelSim com a Python. El fet de realitzar-se amb aquests dos programes és per comparar els resultats i veure que els codis coincideixen pràcticament. S'avança que l'aparició de codis intercanviats enter dos valors ve donat perquè aquests tenen la mateixa probabilitat.

Arbre de 8 Valors

La primera simulació del circuit realitzada ha sigut per a un arbre de 8 valors. D'aquesta manera es comprova que el programa funciona per a arbres menuts. La Figura 37 mostra una vista general de la pantalla de simulació de la qual es detallaran les parts més interessants per a l'apartat.

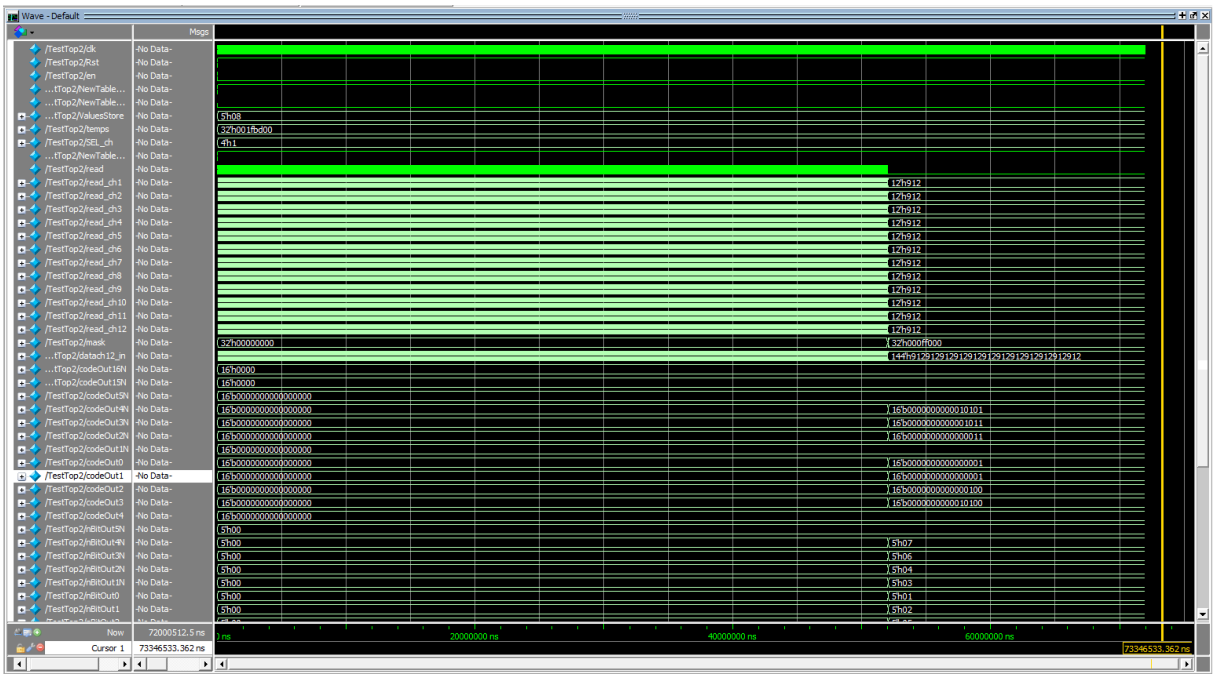

Figura 37. Vista general de la simulació

La Figura 38 representa els codis obtinguts a la part de dalt i el nombre de bits d'aquests codis a la part de baix. Pot observar-se que els nombres seleccionats són el rang [-4, 3].

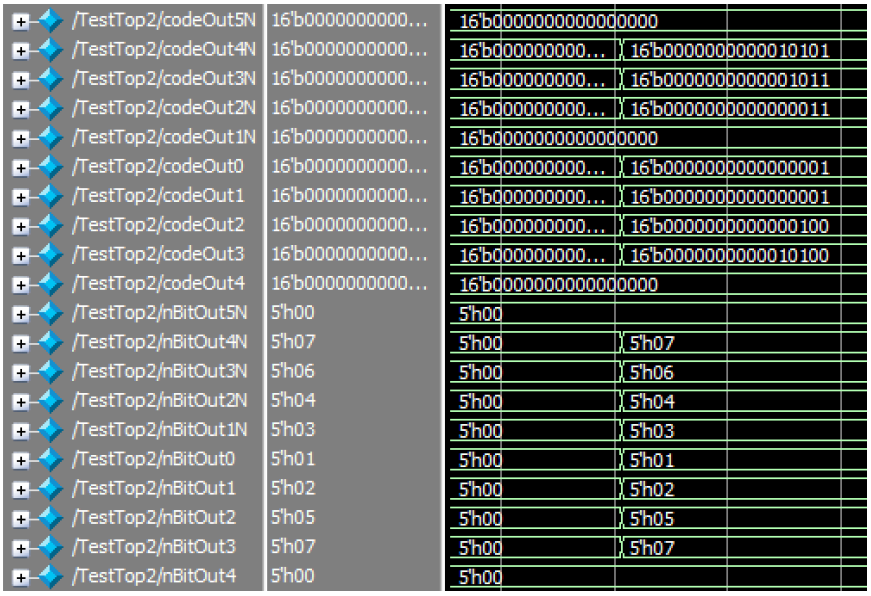

Figura 38. Codis i nombre de bits

La mateixa simulació (mateix nombre d'esdeveniments i un arbre amb la mateixa quantitat de valor) ha sigut realitzat al programa de Python, obtenint-se el que es veu a la Figura 39. S'aprecia que el rang seleccionat pel programa de Python és el mateix que el del programa del projecte, [-4, 3]. A més, tots els codis coincideixen, verificant-se d'aquesta manera el seu correcte funcionament amb arbres menuts.

```
dic
dict_items([(0, '1'), (1, '01'), (-2, '0011'), (-3, '001011'), (-4, '0010101'), (3, '0010100'), (2, '00100'), (-1,
'000')])
```

Figura 39. Codificació resultant del Python

Arbre de 17 Valors

La següent simulació feta ha sigut per a un arbre de 17 valors, ja que, com s'ha esmentat amb anterioritat, la taula de codificació implementada a l'actual FPGA està dissenyada per a 17 valors. Per tant, és menester que s'haja de verificar aquest cas. Igual que amb l'anterior prova, a la Figura 40 pot observar-se una vista general de la simulació realitzada al ModelSim, que serà detallada a continuació.

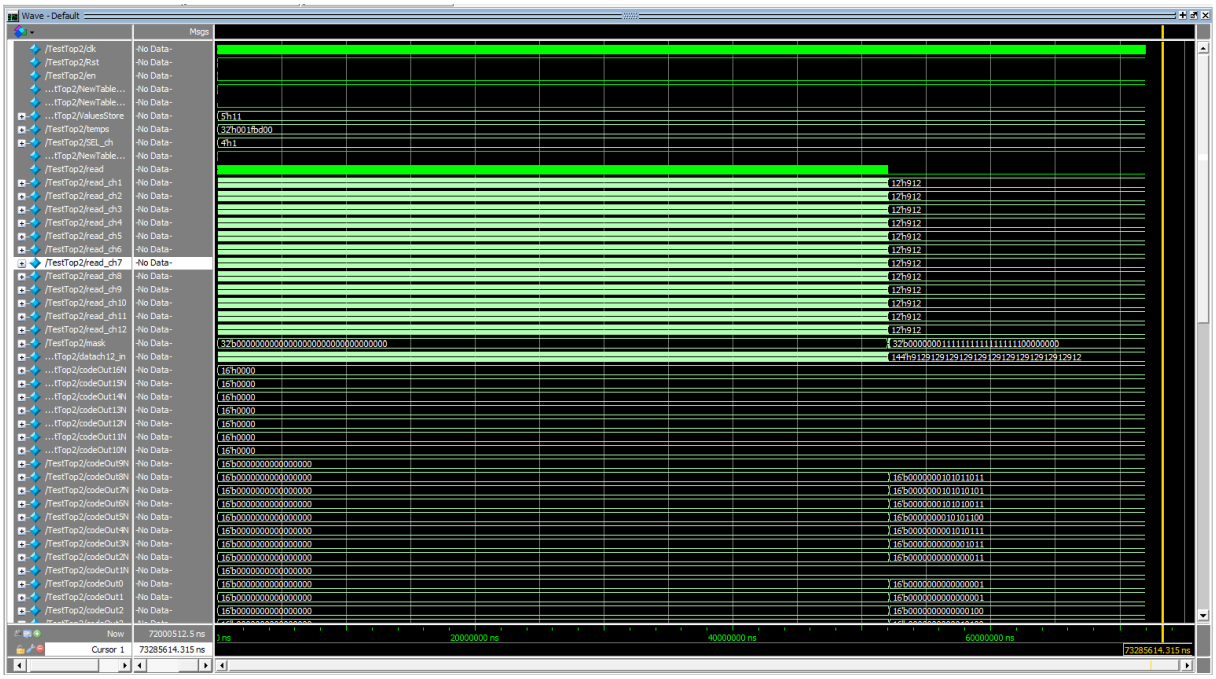

Figura 40. Vista general de la simulació

La Figura 41 i la Figura 42 mostren els codis i el nombre de bits respectivament d'aquesta simulació. Com és d'esperar, el rang de nombres seleccionat inclou el de la verificació anterior, sent en aquest cas [-8, 8].

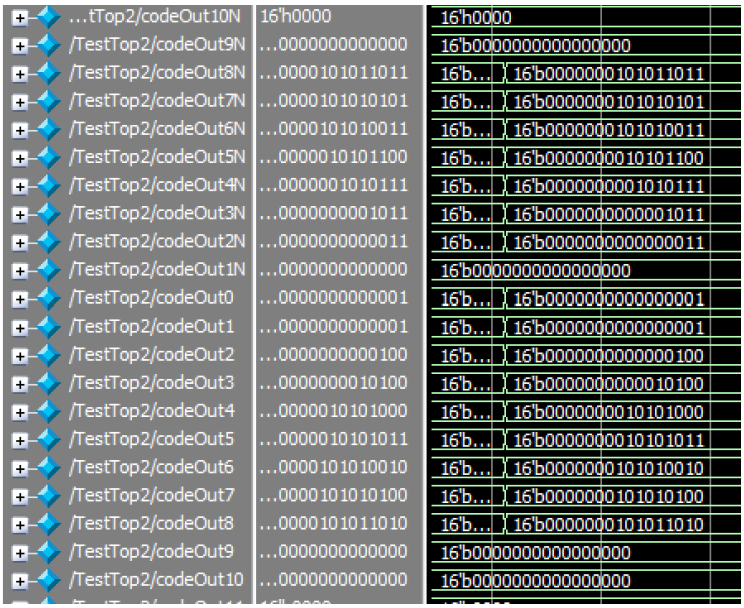

Figura 41. Codis

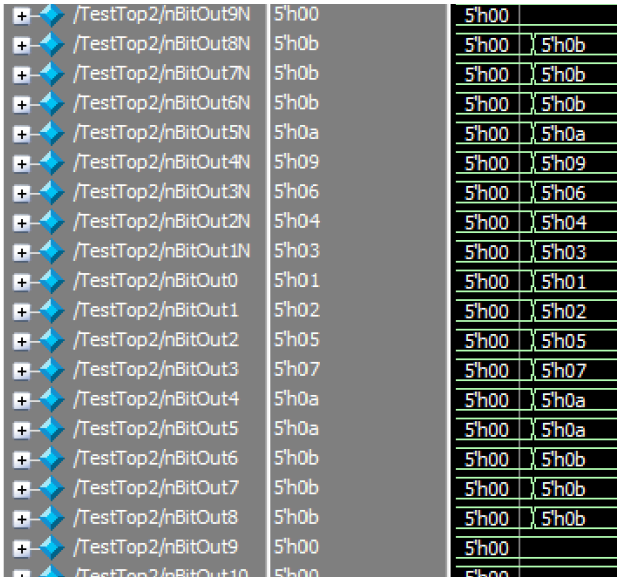

Figura 42. Nombre de bits

Realitzant la simulació al programa de Python, s'obté la Figura 43. Igual que amb l'arbre anterior, els codis i els valors seleccionats coincideixen. Per tant, el programa és vàlid per a l'actual taula implementada a l'FPGA. Sols restaria veure el seu comportament per a arbres amb més valors.

```
dic
dict_items([(0, '1'), (1, '01'), (-2, '0011'), (-3, '001011'), (-4, '001010111'), (-8, '00101011011'), (8, '0010101
1010'), (-5, '0010101100'), (5, '0010101011'), (-7, '00101010101'), (7, '00101010100'), (-6, '00101010011'), (6, '0
0101010010'), (4, '0010101000'), (3, '0010100'), (2, '00100'), (-1, '000')])
```

Figura 43. Codificació resultant del Python

Arbre de 26 Valors

Per últim, s'ha realitzat una simulació per a un arbre de 26 valors. D'aquesta manera s'acaba de verificar el funcionament del programa comprovant que no hi ha problemes al construir un arbre amb una grandària considerable. A la Figura 44 es mostra la una vista general d'aquesta última simulació, la qual es detallarà a continuació.

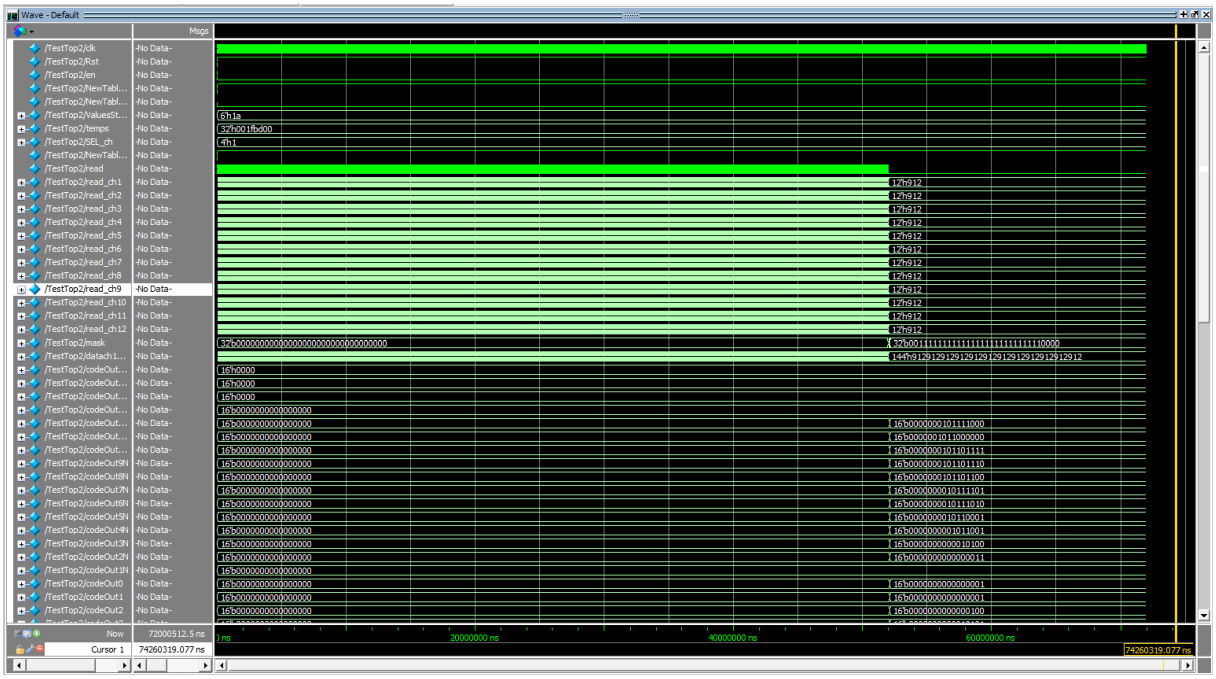

Figura 44. Vista general de la simulació

La Figura 45 i la Figura 46 mostren els codis i els nombres de bits de l'arbre, respectivament. En aquest cas, el rang seleccionat és [-12, 13].

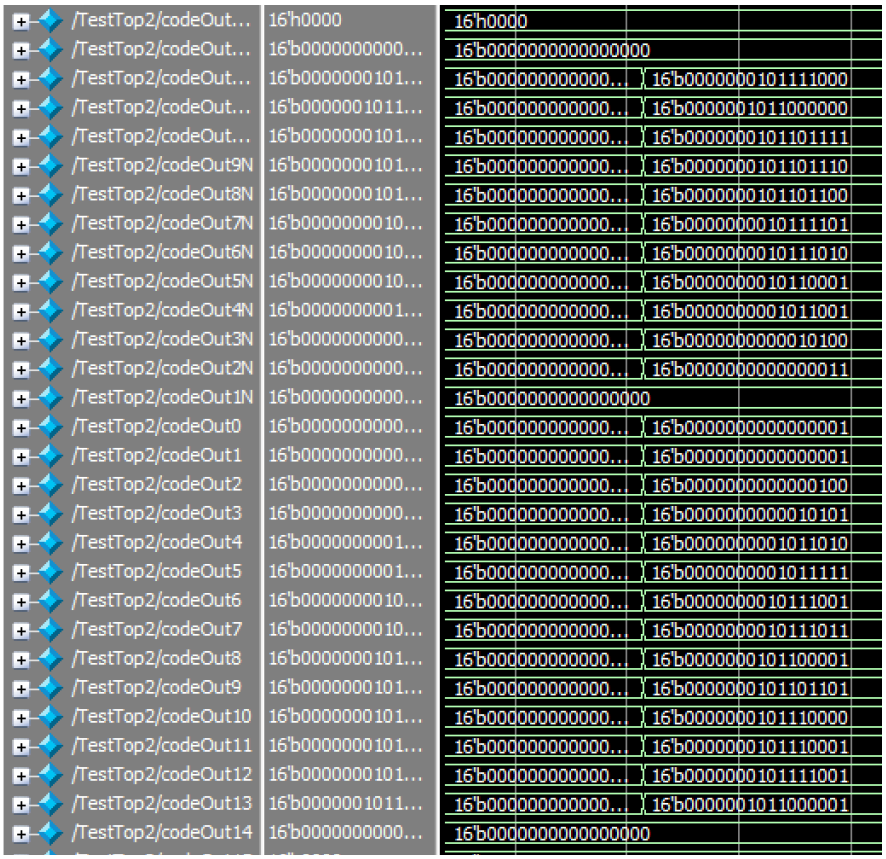

Figura 45. Codis

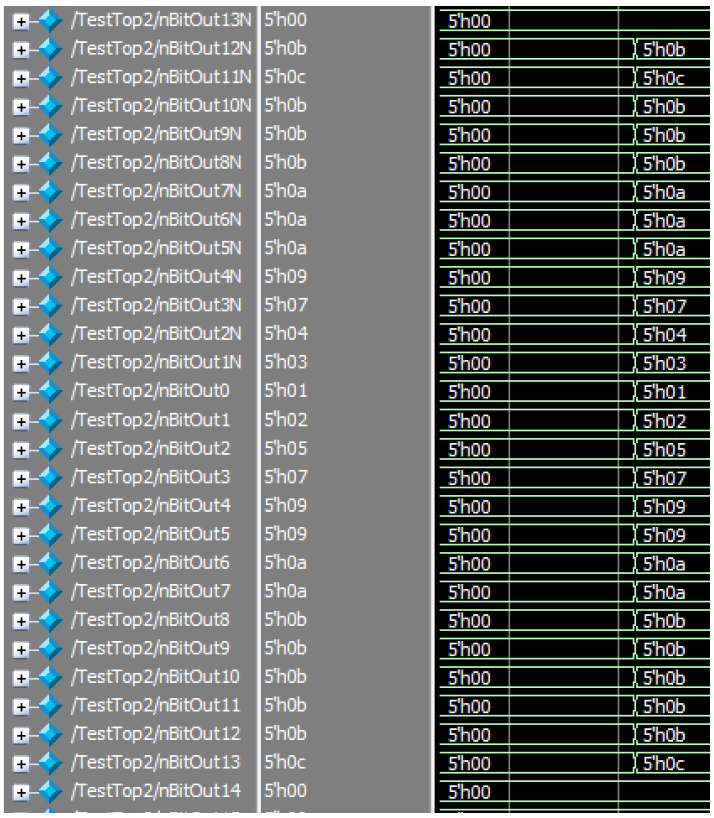

Figura 46. Nombre de bits

La Figura 47 mostra els resultats obtinguts amb el programa de Python. Per a aquesta quantitat d'esdeveniments, ambdós arbres continuen coincidint, tant en els valors escollits com en els codis assignats. Es verifica d'aquesta manera que el sistema funciona també per a arbres grans i, per tant, que el sistema és plenament funcional.

```
dic
dict_items([(0, '1'), (1, '01'), (-2, '0011'), (5, '001011111'), (-7, '0010111101'), (12, '00101111001'), (-12, '00
101111000'), (7, '0010111011'), (-6, '0010111010'), (6, '0010111001'), (11, '00101110001'), (10, '00101110000'), (-
10, '00101101111'), (-9, '00101101110'), (9, '00101101101'), (-8, '00101101100'), (4, '001011010'), (-4, '001011001
'), (-5, '0010110001'), (8, '00101100001'), (13, '0010110000001'), (-11, '0010110000000'), (3, '0010101'), (-3, '0010
100'), (2, '00100'), (-1, '000')])
```

Figura 47. Codificació resultant del Python

10. CONCLUSIONS

A mode de recordatori, es repeteixen els objectius detallats al seu apartat corresponent:

- L'estudi dels codi Huffman i l'obtenció de l'arbre de codis que s'utilitzaran posteriorment per a fer la compressió mitjançant l'ús del llenguatge de programació Python.
- Definició del funcionament del circuit.
- Implementació del circuit en Verilog per a una FPGA de Xilinx.
- Estimació dels recursos utilitzats en FPGA per a l'estudi de la viabilitat de la seua integració en els mòduls ja existents del sistema d'adquisició de dades.
- Verificació del disseny.

Tenint els objectius presents, es poden traure les següents conclusions. S'ha realitzat un estudi dels codi Huffman per obtindre els valors més probables de les possibles dades de l'experiment i s'ha obtingut els codis corresponents per ser comparats posteriorment amb els del programa utilitzant Python.

S'ha definit i detallat el funcionament del circuit que ha sigut programat en llenguatge Verilog. El programa s'ha estructurat en tres blocs diferents que han sigut verificats individualment. No sols cadascun d'aquests, sinó també els superiors que els contenen, fins a aplegar al programa complet.

Aquest programa podrà ser posteriorment implementat en l'FPGA que ja conté els mòduls d'adquisició i compressió de les dades de l'experiment. I s'han estimat els recursos emprats pel programa pel que respecta a l'FPGA, concloent-se que sí podria ser viable la incorporació d'aquest. Els recursos no assoleixen un percentatge suficientment elevat com per afectar en gran mesura les connexions amb més retard del circuit (camins crítics).

Finalment, es pot concloure segons els resultats obtinguts a l'apartat VERIFICACIÓ DEL FUNCIONAMENT DEL PROJECTE que el projecte compleix els objectius requerits pel que fa a funcionalitat del disseny. La codificació s'està realitzant de manera correcta. Els valors que més vegades apareixen al fitxer de dades són comprimits amb un nombre menor de bits, que és el que es busca en aquesta compressió, i permet una adaptació a qualsevol tipus de senyal que genere el experiment.

A nivell personal, ha resultat molt interessant fer un projecte en aquesta disciplina. Normalment, al realitzar una programació se sol desenvolupar la solució de manera instintiva mitjançant funcions, tal i com es fa amb llenguatges més abstractes. En aquest cas, es té un tipus de llenguatge en el qual s'ha de descompondre la solució en xicotets components electrònics i després, fer una màquina d'estats que duga a terme les accions determinades. Aquest canvi en el pensament ha sigut el més complicat de tot el projecte. És també satisfactori pensar que no es tracta d'un projecte aïllat, sinó que forma part d'un projecte molt més gran en el qual han participat altres alumnes amb el seus respectius treballs.

Quedaria com a treball futur la integració del circuit implementat a dintre del projecte general que està funcionant a l'experiment. La major funció del present treball és l'adaptabilitat dels codis a les dades d'entrada. Resultaria interessant també com a futur projecte l'anàlisi de la variabilitat dels codis depenent de la font radioactiva utilitzada. Malgrat que aquesta no pot ser canviada a voluntat, es podria aprofitar els processos de calibratge que es realitzen cada cert temps. La durada d'aquests estudis, degut a que els períodes de calibratge poden durar fins a algun mes, fan impossible la seua inclusió al present treball.

11. BIBLIOGRAFIA

[1] «Economía Europa Press,» Europa Press, 14 Novembre 2019. [En línia]. Available: https://www.europapress.es/economia/noticia-uso-dispositivos-iot-aumento-66-dos-ultimos-anos-telefonica-20191114114851.html. [Últim accés: 11 Juliol 2020].

[2] «itReseller Tech&Consulting,» IT Digital Media Group, 24 Gener 2020. [En línia]. Available: https://www.itreseller.es/en-cifras/2020/01/en-tres-anos-se-venderan-mas-de-3000-millones-de-dispositivos-inteligentes. [Últim accés: 11 Juliol 2020].

[3] M. Morales Sandoval, *Notas sobre Compresión de Datos,* INAOE, 2003.

[4] «next,» [En línia]. Available: https://next.ific.uv.es/next/experiment/physics.html. [Últim accés: 4 Maig 2020].

[5] «IFIC,» [En línia]. Available: https://webific.uv.es/web/content/buscando-la-verdadera-naturaleza-del-neutrino. [Últim accés: 4 Maig 2020].

[6] F. R. Villatoros, «La Ciencia de la Mula Francis,» Naukas, 12 Juliol 2012. [En línia]. Available: https://francis.naukas.com/2012/07/12/la-desintegracion-doble-beta-sin-neutrinos/. [Últim accés: 4 Maig 2020].

[7] «NeoFronteras,» 28 Abril 2019. [En línia]. Available: https://neofronteras.com/?p=6718. [Últim accés: 4 Maig 2020].

[8] «next,» [En línia]. Available: https://next.ific.uv.es/next/experiment/detector.html. [Últim accés: 4 Maig 2020].

[9] V. Álvarez, V. Herrero-Bosch, R. Esteve, A. Laing, J. Rodríguez, M. Querol, F. Monrabal, J. Toledo i J. Gómez-Cadenas, «The electronics of the energy plane of the NEXT_White detector,» *Nuclear Inst. and Methods in Physics Research,* vol. 917, pp. 68-76, 2019.

[10] R. Esteve, J. Toledo, F. Monrabal, D. Lorca, L. Serra, A. Marí, J. Gómez-Cadenas, I. Liubarsky i F. Mora, «The trigger system in the NEXT-DEMO detector,» *JINST,* vol. 7, núm. 12, pp. 17-21, 2012.

[11] R. Esteve, *Data Format and GUI (version INDIA),* 2019.

[12] J. Rodríguez, *Módulo Hardware de Compresión en Tiempo Real,* València: Treball Fi de Grau, Universitat Politècnica de València, 2019.

[13] R. Esteve, «Dispositivos de Lógica Programable,» de *apunts de l'assignatura SEDA (MEI, esp. Electrònica)*, Universitat Politècnica de València, València.

[14] Xilinx, Inc., «What is an FPGA,» [En línia]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. [Últim accés: 6 Maig 2020].

[15] Xilinx, Inc.;, «Virtex-6 Family Overview,» Agost 2015. [En línia]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds150.pdf. [Últim accés: 11 Juny 2020].

[16] Xilinx, Inc., «ISE Design Suite,» [En línia]. Available: https://www.xilinx.com/products/design-tools/ise-design-suite.html. [Últim accés: 6 Maig 2020].

[17] Xilinx, Inc., «ISE WebPACK Design Software,» [En línia]. Available: https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html. [Últim accés: 6 Maig 2020].

[18] Mentor, «ModelSim,» [En línia]. Available: https://www.mentor.com/products/fv/modelsim/. [Últim accés: 6 Maig 2020].

[19] Project Jupyter, «Jupyter,» [En línia]. Available: https://jupyter.org. [Últim accés: 9 Juny 2020].

[20] L. Llamas, «Luis Llamas,» 7 Novembre 2017. [En línia]. Available: https://www.luisllamas.es/que-es-una-fpga/. [Últim accés: 10 Juliol 2020].

DISSENY I IMPLEMENTACIÓ D'UN ARBRE DE CODIFICACIÓ HUFFMAN EN TEMPS REAL EN FPGA PER A L'OPTIMITZACIÓ DE LA COMPRESSIÓ DE DADES DEL PLA D'ENERGIA DE L'EXPERIMENT NEXT

Document núm. 3: Pressupost

1. INTRODUCCIÓ

Es detalla en aquest document el pressupost del present projecte, on es mostraran les diverses unitats en què es compon. Es recorda que aquest treball forma part d'un projecte major, per la qual cosa es considera que la targeta FPGA on hi anirà el programa ja és present al projecte i no s'ha d'adquirir. Com a material utilitzat, sols es tindran en compte els programes que han sigut necessaris per a dur a terme el treball i l'ordinador utilitzat.

2. QUADRES DE PREU

2.1. Mà d'Obra

Serà necessària l'activitat d'un Enginyer Industrial amb coneixements d'electrònica digital i de programació en HDL per a la realització d'aquest projecte. A la Taula 7 s'ha estimat un preu per a l'hora de treball de l'enginyer.

Concepte	Total (€)
Total per Hora de Treball	22,36

Taula 7. Preu de la mà d'obra de l'Enginyer Industrial

2.2. Material

El material se separa en dos quadres de preus diferents. Per un banda es té el hardware utilitzat, que consta d'un ordinador iMac d'Apple , i per altra banda el software.

Per a l'ordinador s'ha considerat el preu estàndard al web de la botiga d'Apple i un període d'amortització d'uns 4 anys. S'ha considerat també per al preu a amortitzar que es podrien fer uns 6 projectes com aquest a l'any, més o menys. Tenint en compte que la càrrega d'hores està estipulada amb 300 hores, les hores a amortitzar a l'any seran 1800. El preu del hardware queda a la Taula 8.

Concepte	Total (€)	Preu a amortitzar (€/h)
Ordinador Apple iMac	1699	0,24

Taula 8. Preu del Hardware

El software consta dels dos programes utilitzats al projecte i de l'Office de Microsoft per a la redacció del projecte i altres afers com les taules fetes amb l'Excel. Tant les versions de l'ISE com del ModelSim seleccionades són gratuïtes. Per tant, sols es tindrà en compte per al pressupost el preu del Microsoft Office Professional, la versió de compra única sense serveis online. Aquesta versió serveix sols per a un ordinador, just per al projecte. Les noves versions d'Office es llancen aproximadament cada 3 anys, que és el que es té en compte per al període d'amortització. El preus considerats per al software es mostren a la Taula 9.

Concepte	Total (€)	Preu a amortitzar (€/h)
ISE Design Suite	0	0
Modelsim PE Student Edition	0	0
Office Professional 2019	579	0,11

Taula 9. Preu del Software

2.3. Quadres de preus descomposts

L'activitat del projecte s'ha subdividit en 7 tasques corresponents a les distintes parts d'aquest. A cadascuna de les tasques se li associa una càrrega d'hores dels elements utilitzats per dur-les a terme (mà d'obra i material) segons el rendiment amb què s'han realitzat. Les taules següents mostren aquest desglossament.

U1	Anàlisi d'alternatives i estudi previ		
Costs directes			
Rendiment	Descripció	Preu	Import (€/h)
0,08	(h) Enginyer Industrial	22,36	1,863333333
0,08	(h) Ordinador Apple iMac	0,24	0,019664352
		TOTAL	1,882997685
Costs indirectes			
	3%	1,882997685	0,056489931
	TOTAL UNITAT (€/h)	**1,939487616**	

Taula 10. Preus descomposts de la Unitat 1

U2	Programació i simulació del bloc de lectura i processat inicial de dades		
Costs directes			
Rendiment	Descripció	Preu	Import (€/h)
0,10	(h) Enginyer Industrial	22,36	2,236
0,10	(h) Ordinador Apple iMac	0,24	0,023597222
0,12	(h) ISE Design Suite	0	0
0,2	(h) ModelSim	0	0
		TOTAL	2,259597222
Costs indirectes			
	3%	2,259597222	0,067787917
	TOTAL UNITAT (€/h)	**2,327385139**	

Taula 11. Preus descomposts de la Unitat 2

U3	Programació i simulació del bloc de construcció de l'arbre de Huffman		
Costs directes			
Rendiment	Descripció	Preu	Import (€/h)
0,10	(h) Enginyer Industrial	22,36	2,236
0,10	(h) Ordinador Apple iMac	0,24	0,023597222
0,12	(h) ISE Design Suite	0	0
0,2	(h) ModelSim	0	0
		TOTAL	2,259597222
Costs indirectes			
	3%	2,259597222	0,067787917
	TOTAL UNITAT (€/h)	**2,327385139**	

Taula 12. Preus descomposts de la Unitat 3

U4	Programació i simulació del bloc de codificació de l'arbre		
Costs directes			
Rendiment	Descripció	Preu	Import (€/h)
0,10	(h) Enginyer Industrial	22,36	2,236
0,10	(h) Ordinador Apple iMac	0,24	0,023597222
0,12	(h) ISE Design Suite	0	0
0,2	(h) ModelSim	0	0
		TOTAL	2,259597222
Costs indirectes			
	3%	2,259597222	0,067787917
	TOTAL UNITAT (€/h)	**2,327385139**	

Taula 13. Preus descomposts de la Unitat 4

U5	Programació del conjunt de blocs		
Costs directes			
Rendiment	Descripció	Preu	Import (€/h)
0,08	(h) Enginyer Industrial	22,36	1,863333333
0,08	(h) Ordinador Apple iMac	0,24	0,019664352
0,09	(h) ISE Design Suite	0	0
		TOTAL	1,882997685
Costs indirectes			
	3%	1,882997685	0,056489931
	TOTAL UNITAT (€/h)	**1,939487616**	

Taula 14. Preus descomposts de la Unitat 5

U6	Verificació del funcionament del programa i depuració		
Costs directes			
Rendiment	Descripció	Preu	Import (€/h)
0,47	(h) Enginyer Industrial	22,36	10,43466667
0,47	(h) Ordinador Apple iMac	0,24	0,11012037
0,55	(h) ISE Design Suite	0	0
0,4	(h) ModelSim	0	0
		TOTAL	10,54478704
Costs indirectes			
	3%	10,54478704	0,316343611
	TOTAL UNITAT (€/h)	**10,86113065**	

Taula 15. Preus descomposts de la Unitat 6

U7	Redacció dels documents del projecte		
Costs directes			
Rendiment	Descripció	Preu	Import (€/h)
0,07	(h) Enginyer Industrial	22,36	1,490666667
0,07	(h) Ordinador Apple iMac	0,24	0,015731481
1	Office Professional 2019	0,11	0,107222222
		TOTAL	1,61362037
Costs indirectes			
	3%	1,61362037	0,048408611
	TOTAL UNITAT (€/h)	**1,662028981**	

Taula 16. Preus descomposts de la Unitat 7

2.4. Preus unitaris

La Taula 17 mostra un resum dels totals de les taules anteriors tenint en compte una aproximació de les hores invertides en cadascuna de les tasques en què s'ha dividit el treball.

Núm.	Descripció	Mesura (h)	Preu (€/h)	Import (€)
1	Anàlisi d'alternatives i estudi previ	25	1,939487616	48,48719039
2	Programació i simulació del bloc de lectura i processat inicial de dades	30	2,327385139	69,82155417
3	Programació i simulació del bloc de construcció de l'arbre de Huffman	30	2,327385139	69,82155417
4	Programació i simulació del bloc de codificació de l'arbre	30	2,327385139	69,82155417
5	Programació del conjunt de blocs	25	1,939487616	48,48719039
6	Verificació del funcionament del programa i depuració	140	10,86113065	1520,558291
7	Redacció dels documents del projecte	20	1,662028981	33,24057963

Taula 17. Preus unitaris

3. PRESSUPOST FINAL

La Taula 18 presenta el pressupost final d'aquest projecte tenint en compte tot allò detallat en els apartats anteriors.

Pressupost	Import (€)
Total de les unitats de mesura	1860,24
13% Despeses Generals	241,83
Subtotal	2102,07
21% IVA	441,43
TOTAL	**2543,50**

Taula 18. Pressupost final

DISSENY I IMPLEMENTACIÓ D'UN ARBRE DE CODIFICACIÓ HUFFMAN EN TEMPS REAL EN FPGA PER A L'OPTIMITZACIÓ DE LA COMPRESSIÓ DE DADES DEL PLA D'ENERGIA DE L'EXPERIMENT NEXT

Annex núm. 1: Detalls de la programació

1. INTRODUCCIÓ

En aquest annex es tractarà d'aprofundir d'una manera més tècnica en algunes parts dels circuits explicats a l'apartat DETALL DEL CIRCUIT DIGITAL IMPLEMENTAT. Es mostrarà també els dissenys RTL realitzats per l'ISE Design Suite. No obstant, per qüestions de claredat i per facilitar l'explicació i la comprensió d'aquests circuits, s'evitarà la utilització d'aquests, quedant-se simplement com a exemples informatius.

2. MÒDULS DE LECTURA

Es presenten a continuació els mòduls programats per a la lectura amb el format RTL del Xilinx ISE. Els afegits són un registre on s'emmagatzema el valor anterior i un restador per obtenir la diferència i enviar-la al recompte de dades. La Figura 48 mostra el bloc de lectura que s'utilitzarà per al projecte complet, la Figura 49 són els mòduls que s'han emprat per poder fer les proves i la Figura 50Figura 48, la programació d'aquests mòduls.

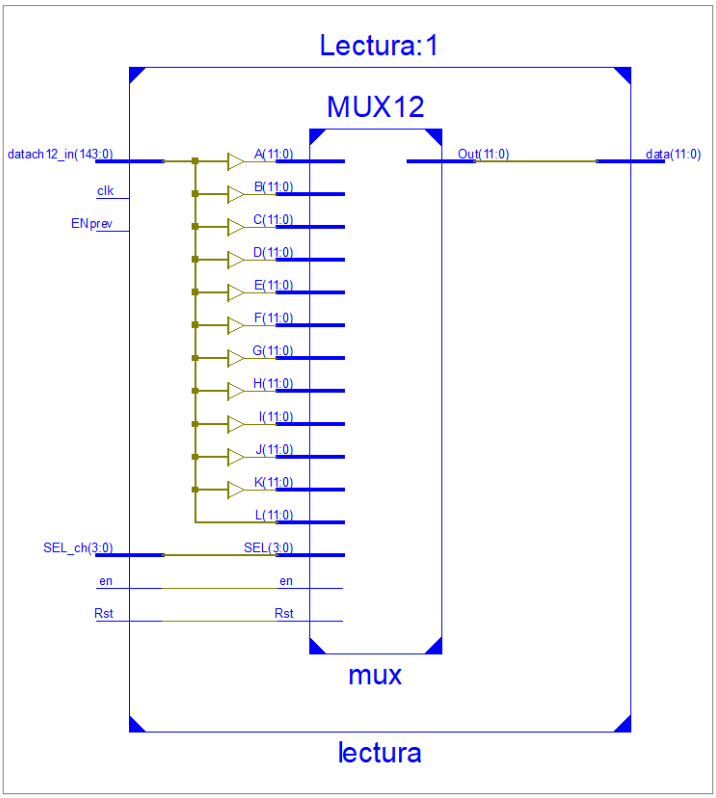

Figura 48. RTL del mòdul "Lectura" del projecte

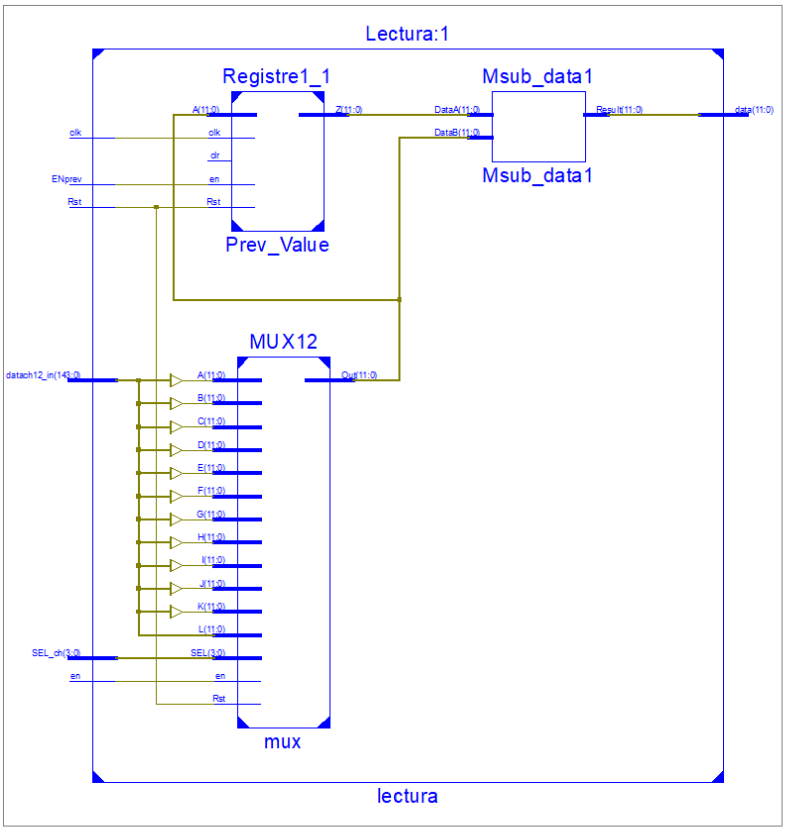

Figura 49. RTL del mòdul "Lectura" utilitzat per a les simulacions

```
// Only one channel is selected for the data counting
   MUX12 #(.Bits(Bits)) mux(
      .en(en), .Rst(Rst), .SEL(SEL_ch),
      .A(read_ch1), .B(read_ch2), .C(read_ch3), .D(read_ch4), .E(read_ch5), .F(read_ch6),
      .G(read_ch7), .H(read_ch8), .I(read_ch9), .J(read_ch10), .K(read_ch11), .L(read_ch12),
      .Out(data1)
      );

//    Previous value register
      Registre1 #(.Bits(Bits)) Prev_Value(.en(ENprev), .Rst(Rst), .clk(clk), .A(data1), .Z(prev_val));

      //assign data = data1-32'h0912;  //2322 -- Input data, baseline surpass compression
      assign data = prev_val-data1;    //input data, differential compression
```

Figura 50. Elements programats al mòdul "Lectura"

3. RECOMPTES DELS VALORS DEL RANG SELECCIONAT

La Figura 51 mostra l'agrupació dels mòduls utilitzats per a obtindre els recomptes i un temporitzador que, com s'ha esmentat, serveix per aturar l'adquisició. La variable "temps" connectada al temporitzador és una variable d'entrada al programa i és la que delimita el temps de recompte.

```
//Counters----------------------
   Comp_counter c16_neg(.Rst(Rst), .clr(mask[0]), .clk(clk), .en(add), .value(dataIn), .A(12'h010), .C(prob16n));
   Comp_counter c15_neg(.Rst(Rst), .clr(mask[1]), .clk(clk), .en(add), .value(dataIn), .A(12'h00f), .C(prob15n));
   Comp_counter c14_neg(.Rst(Rst), .clr(mask[2]), .clk(clk), .en(add), .value(dataIn), .A(12'h00e), .C(prob14n));
   Comp_counter c13_neg(.Rst(Rst), .clr(mask[3]), .clk(clk), .en(add), .value(dataIn), .A(12'h00d), .C(prob13n));
   Comp_counter c12_neg(.Rst(Rst), .clr(mask[4]), .clk(clk), .en(add), .value(dataIn), .A(12'h00c), .C(prob12n));
   Comp_counter c11_neg(.Rst(Rst), .clr(mask[5]), .clk(clk), .en(add), .value(dataIn), .A(12'h00b), .C(prob11n));
   Comp_counter c10_neg(.Rst(Rst), .clr(mask[6]), .clk(clk), .en(add), .value(dataIn), .A(12'h00a), .C(prob10n));
   Comp_counter c9_neg(.Rst(Rst), .clr(mask[7]), .clk(clk), .en(add), .value(dataIn), .A(12'h009), .C(prob9n));
   Comp_counter c8_neg(.Rst(Rst), .clr(mask[8]), .clk(clk), .en(add), .value(dataIn), .A(12'h008), .C(prob8n));
   Comp_counter c7_neg(.Rst(Rst), .clr(mask[9]), .clk(clk), .en(add), .value(dataIn), .A(12'h007), .C(prob7n));
   Comp_counter c6_neg(.Rst(Rst), .clr(mask[10]), .clk(clk), .en(add), .value(dataIn), .A(12'h006), .C(prob6n));
   Comp_counter c5_neg(.Rst(Rst), .clr(mask[11]), .clk(clk), .en(add), .value(dataIn), .A(12'h005), .C(prob5n));
   Comp_counter c4_neg(.Rst(Rst), .clr(mask[12]), .clk(clk), .en(add), .value(dataIn), .A(12'h004), .C(prob4n));
   Comp_counter c3_neg(.Rst(Rst), .clr(mask[13]), .clk(clk), .en(add), .value(dataIn), .A(12'h003), .C(prob3n));
   Comp_counter c2_neg(.Rst(Rst), .clr(mask[14]), .clk(clk), .en(add), .value(dataIn), .A(12'h002), .C(prob2n));
   Comp_counter c1_neg(.Rst(Rst), .clr(mask[15]), .clk(clk), .en(add), .value(dataIn), .A(12'h001), .C(prob1n));
   Comp_counter c0(.Rst(Rst), .clr(mask[16]), .clk(clk), .en(add), .value(dataIn), .A(12'h000), .C(prob0));
   Comp_counter c1(.Rst(Rst), .clr(mask[17]), .clk(clk), .en(add), .value(dataIn), .A(12'h001), .C(prob1));
   Comp_counter c2(.Rst(Rst), .clr(mask[18]), .clk(clk), .en(add), .value(dataIn), .A(12'h002), .C(prob2));
   Comp_counter c3(.Rst(Rst), .clr(mask[19]), .clk(clk), .en(add), .value(dataIn), .A(12'h003), .C(prob3));
   Comp_counter c4(.Rst(Rst), .clr(mask[20]), .clk(clk), .en(add), .value(dataIn), .A(12'h004), .C(prob4));
   Comp_counter c5(.Rst(Rst), .clr(mask[21]), .clk(clk), .en(add), .value(dataIn), .A(12'h005), .C(prob5));
   Comp_counter c6(.Rst(Rst), .clr(mask[22]), .clk(clk), .en(add), .value(dataIn), .A(12'h006), .C(prob6));
   Comp_counter c7(.Rst(Rst), .clr(mask[23]), .clk(clk), .en(add), .value(dataIn), .A(12'h007), .C(prob7));
   Comp_counter c8(.Rst(Rst), .clr(mask[24]), .clk(clk), .en(add), .value(dataIn), .A(12'h008), .C(prob8));
   Comp_counter c9(.Rst(Rst), .clr(mask[25]), .clk(clk), .en(add), .value(dataIn), .A(12'h009), .C(prob9));
   Comp_counter c10(.Rst(Rst), .clr(mask[26]), .clk(clk), .en(add), .value(dataIn), .A(12'h00a), .C(prob10));
   Comp_counter c11(.Rst(Rst), .clr(mask[27]), .clk(clk), .en(add), .value(dataIn), .A(12'h00b), .C(prob11));
   Comp_counter c12(.Rst(Rst), .clr(mask[28]), .clk(clk), .en(add), .value(dataIn), .A(12'h00c), .C(prob12));
   Comp_counter c13(.Rst(Rst), .clr(mask[29]), .clk(clk), .en(add), .value(dataIn), .A(12'h00d), .C(prob13));
   Comp_counter c14(.Rst(Rst), .clr(mask[30]), .clk(clk), .en(add), .value(dataIn), .A(12'h00e), .C(prob14));
   Comp_counter c15(.Rst(Rst), .clr(mask[31]), .clk(clk), .en(add), .value(dataIn), .A(12'h00f), .C(prob15));
//-------------------------------

//Timer
   Timer timer1(.Rst(Rst), .clk(clk), .en(ENt), .clr(clrT), .temps(temps), .timeOut(timeOut));

```

Figura 51. Comptadors per al rang de valors més probables i temporitzador

Aquests comptadors són una barreja de comparadors i comptadors, tal i com pot veure's a la Figura 52. La variable "A" del mòdul en qüestió és el valor amb el que es compara el d'entrada al sistema per veure si el comptador augmenta o no.

```
module Comp_counter #(parameter Bits=32)(
   input Rst, clr, en, clk,
   input [11:0] value, A,
   output reg [Bits-1:0] C
   );

   always@(negedge Rst or posedge clr or posedge en)
   begin
      if(!Rst || clr) C<=0;
      else
         begin
            if(en) if(value==A) C<=C+1'b1;   //The counter o
         end
   end

endmodule
```

Figura 52. Programació dels comptadors dels valors més probables

4. MÒDULS PER A L'OBTENCIÓ DELS VALORS DE L'ARBRE

Com bé s'ha comentat a l'apartat corresponent, la feina de cercar el major recompte entre els comptadors la fan un parell de multiplexors. El primer d'aquests, "mux1", és el mostrat a la Figura 53. Poden apreciar-se dues coses d'aquesta figura: cadascuna de les eixides és seleccionada a partir d'un selector que pren els mateixos valors considerats per al recompte i cada eixida ve acompanyada per aquest valor. El motiu d'aquestes peculiaritats es veurà més endavant.

```
always @(SEL or A or B or b or C or c or D or d or E or e or F or f or
G or g or H or h or I or i or J or j or K or k or L or l or M or m or
N or n or O or o or P or p or q or en or Rst)
begin
   if(!Rst) Out<=32'bz;
   else
      begin
         if(en)
         begin
            case(SEL)
               -5'h10: begin Out<=q; value<=-5'h10; end//-16
               -5'h0f: begin Out<=p; value<=-5'h0f; end //-15
               -5'h0e: begin Out<=o; value<=-5'h0e; end //-14
               -5'h0d: begin Out<=n; value<=-5'h0d; end  //-13
               -5'h0c: begin Out<=m; value<=-5'h0c; end  //-12
               -5'h0b: begin Out<=l; value<=-5'h0b; end  //-11
               -5'h0a: begin Out<=k; value<=-5'h0a; end  //-10
               -5'h09: begin Out<=j; value<=-5'h09; end  //-9
               -5'h08: begin Out<=i; value<=-5'h08; end  //-8
               -5'h07: begin Out<=h; value<=-5'h07; end  //-7
               -5'h06: begin Out<=g; value<=-5'h06; end  //-6
               -5'h05: begin Out<=f; value<=-5'h05; end  //-5
               -5'h04: begin Out<=e; value<=-5'h04; end  //-4
               -5'h03: begin Out<=d; value<=-5'h03; end  //-3
               -5'h02: begin Out<=c; value<=-5'h02; end  //-2
               -5'h01: begin Out<=b; value<=-5'h01; end  //-1
               5'h00: begin Out<=A; value<=5'h00; end //0
               5'h01: begin Out<=B; value<=5'h01; end //1
               5'h02: begin Out<=C; value<=5'h02; end //2
               5'h03: begin Out<=D; value<=5'h03; end //3
               5'h04: begin Out<=E; value<=5'h04; end //4
               5'h05: begin Out<=F; value<=5'h05; end //5
               5'h06: begin Out<=G; value<=5'h06; end //6
               5'h07: begin Out<=H; value<=5'h07; end //7
               5'h08: begin Out<=I; value<=5'h08; end //8
               5'h09: begin Out<=J; value<=5'h09; end //9
               5'h0a: begin Out<=K; value<=5'h0a; end //10
               5'h0b: begin Out<=L; value<=5'h0b; end //11
               5'h0c: begin Out<=M; value<=5'h0c; end //12
               5'h0d: begin Out<=N; value<=5'h0d; end //13
               5'h0e: begin Out<=O; value<=5'h0e; end //14
               5'h0f: begin Out<=P; value<=5'h0f; end //15
               default: Out<=32'bz;
            endcase
         end
      end
end
```

Figura 53. Bloc de funcionament del multiplexor "mux1"

El segon multiplexor, "mux2", difereix en el selector d'aquest, va de 0 a 31. És acompanyat per un comptador l'eixida del qual actua com al selector. D'aquesta manera és com aquest multiplexor recorre les eixides dels comptadors detallats a l'apartat anterior. Les eixides dels dos multiplexors són conduïdes a un comparador perquè s'obtinga el major nombre.

Si es té el cas en què s'ha trobat un valor major a l'eixida de "mux2" que a la de"mux1, se li dóna a "SEL1" (el selector de "mux1") un valor igual a "vOut" (Figura 56) i es posa el comptador del multiplexor a 0. La variable "vOut" és l'eixida d'un registre al que li ve la dada del comparador utilitzat per a les eixides dels multiplexors i que es detallarà un poc més avall.

Al segon cas, on és l'eixida de "mux1" la que és igual o major a la de "mux2", augmenta el comptador del multiplexor per passar al següent recompte.

La Figura 54 mostra el codi del comparador utilitzat per al multiplexor. Les entrades "A" i "B" són les que es comparen, els comptes, i "C" i "D" són valors associats a les entrades "A" i "B" respectivament, el que a la Figura 53 s'anomenen "value". És per aquest motiu el que el selector de "mux1" siga així. Gràcies a aquest comparador no sol es troba el major recompte, sinó també el valor associat a aquest. I aquesta informació és la que s'envia a "mux1", que manté un valor fix, i selecciona directament el major recompte a partir del seu valor associat.

```
1  module Comparador #(parameter Bits=5)(
2      input en, Rst,
3      input [Bits-1:0] A, B,     //Comparison inputs
4      input [4:0] C, D,          //Input values associated with
5      output GT, LT, EQ,
6      output [Bits-1:0] Out,
7      output [4:0] Out2
8      );
9
10 // Comparison result between A and B
11    assign GT = Rst ? (en ? (A>B ? 1'b1 : 1'b0): 0): 0;
12    assign LT = Rst ? (en ? (A<B ? 1'b1 : 1'b0): 0): 0;
13    assign EQ = Rst ? (en ? (A==B ? 1'b1 : 1'b0): 0): 0;
14
15 // Provides the higher input between A and B
16    assign Out = Rst ? (en ? (GT ? A : B) : 0) : 0;
17
18 // Provides the value associated with the result above
19    assign Out2 = Rst ? (en ? (GT ? C : D) : 0) : 0;
20
21 endmodule
22
```

Figura 54. Mòdul "Comparador"

A continuació, es mostren la Figura 55 i la Figura 56. En aquestes es troben els mòduls anteriorment descrits programats. A la primera poden veure's els dos multiplexors junt amb el comparador del segon i un comparador amb un registre. Aquests dos últims estan per evitar que el comptador se'n passe del màxim permès (31).

```
//MUXs----------------
// This multiplexor changes its input depending on the results of the comparator below, it remains the higher one
   MUX33_2 mux1(
      .en(enMUX), .Rst(Rst), .SEL(SEL1),
      .A(prob0), .B(prob1), .b(prob1n), .C(prob2), .c(prob2n), .D(prob3), .d(prob3n), .E(prob4), .e(prob4n),
      .F(prob5), .f(prob5n), .G(prob6), .g(prob6n), .H(prob7), .h(prob7n), .I(prob8), .i(prob8n),
      .J(prob9), .j(prob9n), .K(prob10), .k(prob10n), .L(prob11), .l(prob11n), .M(prob12), .m(prob12n),
      .N(prob13), .n(prob13n), .O(prob14), .o(prob14n), .P(prob15), .p(prob15n), .q(prob16n),
      .value(value1), .Out(data1)
      );

// This multiplexor changes its input depending on the following counter in order to search the next higher value
   MUX33 mux2(
      .en(enMUX), .Rst(Rst), .SEL(SEL2),
      .A(prob0), .B(prob1), .b(prob1n), .C(prob2), .c(prob2n), .D(prob3), .d(prob3n), .E(prob4), .e(prob4n),
      .F(prob5), .f(prob5n), .G(prob6), .g(prob6n), .H(prob7), .h(prob7n), .I(prob8), .i(prob8n),
      .J(prob9), .j(prob9n), .K(prob10), .k(prob10n), .L(prob11), .l(prob11n), .M(prob12), .m(prob12n),
      .N(prob13), .n(prob13n), .O(prob14), .o(prob14n), .P(prob15), .p(prob15n), .q(prob16n),
      .value(value2), .Out(data2)
      );

   comptParam MUXcounter(.A(count), .Rst(Rst), .clr(clrC), .en(enMUX), .C(SEL2));   //Counter for 'mux2' selector
   Comparador MUXcomp(.en(count), .Rst(Rst), .A(SEL2), .B(5'h1f), .EQ(EQ));         //Comparator to not exceed the
   Registrel #(.Bits(1)) rcomp(.en(ENc), .Rst(Rst), .clk(clk), .A(EQ), .Z(EQ2));
//---------------------
```

Figura 55. Part dels multiplexors al mòdul "estadistica2"[4]

A la Figura 56 es poden observar el comparador per a les eixides dels multiplexors amb els seus registres per a les eixides, un altre comptador per saber quants nombre s'han enviat ja al següent bloc (es recorda que aquesta dada és elegible) i el mòdul que crea la màscara a mesura que es van seleccionant els majors recomptes.

```
//Comparator that searches the higher value amongst the counted ones with the previous counters
   Comparador #(.Bits(32)) DataComp(.en(compare), .Rst(Rst), .A(data1), .B(data2), .GT(GT), .EQ(EQ1), .Out(data3), .C(value1), .D(value2), .Out2(value3));
   Registrel r1(.en(ENv), .Rst(Rst), .clk(clk), .A(value3), .Z(vOut));                  //Value register
   Registrel #(.Bits(32)) r2(.en(compare), .Rst(Rst), .clk(clk), .A(data3), .Z(prob));  //Probability register
//--------------------

//Stored data counter
   comptParam #(.Bits(6)) DataCount(.A(countValue), .Rst(Rst), .en(enMUX), .resta(resta), .C(numValue));

//Mask
   Huff_mask mask_module(.en(enMask), .Rst(Rst), .clr(clrMask), .dataIn(vOut), .dataOut(mask));
```

Figura 56. Resta de mòduls a "estadistica2"[5]

[4] Aquesta figura es pot veure ampliada a l'Annex 2

[5] Aquesta figura es pot veure ampliada a l'Annex 2

La Figura 57 mostra la programació de la màscara acabada d'esmentar. Es tracta simplement d'una variable de 32 bits, on cadascun d'aquests bits pertany a un valor en concret. Cada cop que s'envia un recompte al següent bloc, el seu valor associat passa per aquest mòdul indicant-hi el bit que ha de passar a ser 1. Els valors són posteriorment enviats a una FIFO per ser recuperats al tercer bloc, la codificació.

```
always @(Rst or en or dataIn or clr)
begin
   if(!Rst||clr) dataOut<=32'h0;
   else
      begin
         if(en)
         begin
            case(dataIn)
               -5'h10: dataOut[0]<=1'b1;  //-16
               -5'h0f: dataOut[1]<=1'b1;  //-15
               -5'h0e: dataOut[2]<=1'b1;  //-14
               -5'h0d: dataOut[3]<=1'b1;  //-13
               -5'h0c: dataOut[4]<=1'b1;  //-12
               -5'h0b: dataOut[5]<=1'b1;  //-11
               -5'h0a: dataOut[6]<=1'b1;  //-10
               -5'h09: dataOut[7]<=1'b1;  //-9
               -5'h08: dataOut[8]<=1'b1;  //-8
               -5'h07: dataOut[9]<=1'b1;  //-7
               -5'h06: dataOut[10]<=1'b1; //-6
               -5'h05: dataOut[11]<=1'b1; //-5
               -5'h04: dataOut[12]<=1'b1; //-4
               -5'h03: dataOut[13]<=1'b1; //-3
               -5'h02: dataOut[14]<=1'b1; //-2
               -5'h01: dataOut[15]<=1'b1; //-1
               5'h00: dataOut[16]<=1'b1;  //0
               5'h01: dataOut[17]<=1'b1;  //1
               5'h02: dataOut[18]<=1'b1;  //2
               5'h03: dataOut[19]<=1'b1;  //3
               5'h04: dataOut[20]<=1'b1;  //4
               5'h05: dataOut[21]<=1'b1;  //5
               5'h06: dataOut[22]<=1'b1;  //6
               5'h07: dataOut[23]<=1'b1;  //7
               5'h08: dataOut[24]<=1'b1;  //8
               5'h09: dataOut[25]<=1'b1;  //9
               5'h0a: dataOut[26]<=1'b1;  //10
               5'h0b: dataOut[27]<=1'b1;  //11
               5'h0c: dataOut[28]<=1'b1;  //12
               5'h0d: dataOut[29]<=1'b1;  //13
               5'h0e: dataOut[30]<=1'b1;  //14
               5'h0f: dataOut[31]<=1'b1;  //15
               default: dataOut<=32'h0;
            endcase
         end
      end
end
```

Figura 57. Funcionament del mòdul "Huff_mask"

5. ENTRADA AL MÒDUL PRINCIPAL DE L'ARBRE

A la Figura 58 es mostra el multiplexor "MUXSEL" i el mòdul "cp2", que construirà l'arbre amb la corresponent FSM. Les variables d'entrada del multiplexor, "ENrEst" i "ENp" són les habilitacions al registre de desplaçament on s'emmagatzemen els recomptes que s'envien del bloc anterior. La primera correspon al mòdul d'estadística i la segona a l'FSM del segon bloc. S'ha de fer d'aquesta manera perquè no interferisquen els dos senyals i es puguen registrar les dades primer des del primer bloc i, després, extraure-les en aquest segon bloc.

```
MUX2 #(.Bits(1)) MUXSEL(.en(1'b1), .SEL(SELe), .Rst(Rst), .A(ENrEst), .B(ENp), .Out(ENpOut));

CalculPosicio2 #(.Bits(32)) cp2(
    .Rst(Rst), .clk(clk), .clr(clr), .en(enMUX),
    .mode(mode), .SELreg(SELreg),
    .ENp(ENpOut), .ENn(ENn), .ENps(ENps),
    .suma(suma), .compta(compta), .descompta(descompta), .init(init), .init2(init2), .compara(compara),
    .definedValue(definedValue), .dataEst(dataIn),
    .compOut(compOut), .maxCount(maxCount), .SEL(SEL),
    .probInReg(probInReg), .posIn(posInReg), .nodeInReg(nodeInReg)
    );
```

Figura 58. Instanciació del mòdul "CalculPosicio2" i el multiplexor d'entrada d'habilitacions a aquest

6. ESTRUCTURES DE 32 REGISTRES

La construcció de l'arbre es realitza mitjançant tres estructures de 32 registres cadascuna. Una emmagatzemant els recomptes, una altra obtenint les altures de cada node i una última on es troben els tipus de node. La instanciació d'aquestes tres és a la Figura 59. S'observa, per una banda, que les tres s'habiliten amb la mateixa variable i tenen el mateix selector.

```
// Registers-----------------
    Structure_32Register #(.Bits(32)) Probabilities(
        .Rst(Rst), .clk(clk), .en(ENp), .enMUX(en), .clr(clr), .mode(mode), .SEL(SEL),
        .dataIn1(probsum), .dataIn2(dataEst), .dataOut1(prob1), .dataOut2(prob2), .dataComp(dataCompIn)
        );
    Structure_32Register #(.Bits(1)) Nodes(
        .Rst(Rst), .clk(clk), .en(ENn), .enMUX(1'b0), .clr(clr), .mode(mode), .SEL(SEL),
        .dataIn1(1'b1), .dataIn2(1'b0), .dataOut1(node1), .dataOut2(node2)
        );
    Structure_32Register #(.Bits(5)) Positions(
        .Rst(Rst), .clk(clk), .en(ENp), .enMUX(1'b0), .clr(clr), .mode(mode), .SEL(SEL),
        .dataIn1(posIn+5'h01), .dataIn2(5'h01), .dataOut1(pos1), .dataOut2(pos2)
        );
//---------------------------
```

Figura 59. Tres estructures de 32 registres instanciades

Cadascuna d'aquestes, com bé s'acaba de comentar, conté al seu interior 32 registres. A la Figura 61 pot veure's la programació d'un dels mòduls, en la qual sols es mostra un dels registres per qüestions d'espai. S'aprecia que, a part dels registres, també té un multiplexor. Aquest s'utilitza per seleccionar l'eixida del mòdul segons un comptador extern a aquest. Pot resultar familiar açò ja que és semblant al funcionament dels dos multiplexors del primer bloc que cerquen els majors recomptes. Per tant, aquest multiplexor (el de la Figura 61), és l'encarregat de recórrer els 32 registres per esbrinar on hi anirà ubicada la suma dels dos recomptes menors, tal i com es detalla a la Memòria del projecte.

```
module Structure_32Register #(parameter Bits=32)(
   input Rst, clk, en, clr, mode, enMUX,
   input [4:0] SEL,
   input [Bits-1:0] dataIn1,
   input [Bits-1:0] dataIn2,     //Comes from the 'estadistica' module
   output [Bits-1:0] dataOut1, dataOut2,  //Related to the two less probable values
   output [Bits-1:0] dataComp
   );

   wire [Bits-1:0] data1, data2, data3, data4, data5, data6, data7, data8;
   wire [Bits-1:0] data9, data10, data11, data12, data13, data14, data15, data16;
   wire [Bits-1:0] data17, data18, data19, data20, data21, data22, data23, data24;
   wire [Bits-1:0] data25, data26, data27, data28, data29, data30, data31, data32;

   assign dataOut1 = data1;
   assign dataOut2= data2;

   MUX32 #(.Bits(Bits)) muxComp(
      .en(enMUX), .Rst(Rst), .SEL(SEL),
      .A(data1), .B(data2), .C(data3), .D(data4), .E(data5), .F(data6), .G(data7), .H(data8), .I(data9),
      .J(data10), .K(data11), .L(data12), .M(data13), .N(data14), .O(data15), .P(data16), .Q(data17),
      .R(data18), .S(data19), .T(data20), .U(data21), .V(data22), .W(data23), .X(data24), .Y(data25),
      .Z(data26), .A2(data27), .B2(data28), .C2(data29), .D2(data30), .E2(data31), .F2(data32), .Out(dataComp)
      );

   Register32bit P1(
      .Rst(Rst), .clk(clk), .en(en), .clr(clr), .mode(mode),
      .StoragePosition(5'h00), .StoragePositionIn(SEL-5'h02),
      .dataIn(dataIn1), .data_prevReg(dataIn2), .data_nextReg(data2), .dataReg2(data3), .dataOut(data1)
      );
```

Figura 60. Mòduls interns de "Structure_32Register"

Tota l'estona està sent anomenat que les estructures contenen 32 registres. Realment, el que es veu a la Figura 61 no són exactament registres. L'interior d'aquests mòduls pot veure's a la Figura 61 i consta d'un registre, un multiplexor i lògica combinacional per al selector d'aquest. Amb tot açò es fa possible ubicar el nou node i reorganitzar els que ja hi són a dintre. Depenent del resultat de la comparació, el multiplexor de la Figura 61 s'haurà quedat seleccionant una eixida determinada. Si a aquesta li restem 2, perquè els dos recomptes menors no s'han de tenir en compte, s'obté la posició exacta on deu ubicar-se el nou node que ha d'entrar a l'estructura.

Si la posició és igual al lloc que ocupa el registre ("StoragePosition"), l'entrada de dades al registre és la provinent de "l'exterior"; si és menor a la ubicació, li entra la del registre següent, és a dir, el valor del registre es desplaça cap a l'esquerra. Sent major a la ubicació, el desplaçament serà a dos registres cap a l'esquerra. Al registre sols hi ha desplaçament cap a la dreta amb la variable "mode" activa, que ve donada quan s'envien les dades des del bloc anterior.

```
module Register32bit #(parameter Bits=32)(
   input Rst, clk, en, clr,
   input mode,    //Variable from the module 'estadistica', enables the entry of the 'prevReg'
   input [4:0] StoragePosition, StoragePositionIn,
   input [Bits-1:0] dataIn, data_prevReg, data_nextReg, dataReg2,    //dataReg2 is the one from the following of the following register
   output [Bits-1:0] dataOut
   );

   wire [Bits-1:0] dataToReg;
   wire [1:0] SEL;

   assign SEL =  mode ? 2'b01 : (StoragePositionIn==StoragePosition ? 2'b00 : (StoragePositionIn<StoragePosition ? 2'b10 : (StoragePositionIn>StoragePosition ? 2'b11 : 2'bzz)));

   MUX4 #(.Bits(Bits)) mux(.en(1'b1), .SEL(SEL), .Rst(Rst), .A(dataIn), .B(data_prevReg), .C(data_nextReg), .D(dataReg2), .Out(dataToReg));

   Registrel #(.Bits(Bits)) ProbR(.en(en), .Rst(Rst), .clk(clk), .clr(clr), .A(dataToReg), .Z(dataOut));

endmodule
```

Figura 61. Programació de "Register32bit"[6]

[6] Aquesta figura es pot veure ampliada a l'Annex 2

7. COMPTADOR UTILITZAT

Com s'ha comentat a l'apartat 6.4.2. del document Memòria, El mòdul de comptador és el mateix utilitzat fins ara i és únic a tot el programa. S'observa que en aquest bloc, l'entrada "en" està posada a 0, ja que sols s'utilitzarà per descomptar. El funcionament del codi és a la imatge Figura 62.

```
1  module comptParam #(parameter Bits=5)(
2     input A, Rst, clr,
3     input en,       //Normal mode enable, increases the counter
4     input resta,    //Enable for decreasing the counter, 'en' prevails
5     input init,     //initializes the counter with the value 'definedVal'
6     input [Bits-1:0] definedVal,
7     output reg [Bits-1:0] C
8     );
9
10    always @(posedge A or posedge init or negedge Rst or posedge clr)
11    begin
12       if(!Rst||clr) C<=5'b0;
13       else
14          begin
15             if(init) C<=definedVal;
16             else
17                begin
18                   if(en) C<=C+1'b1;
19                   else if(resta) C<=C-1'b1;
20                end
21          end
22    end
23
24 endmodule
```

Figura 62. Programació del comptador "comptParam"

8. ESQUEMES RTL DEL BLOC DE CONSTRUCCIÓ DE L'ARBRE

L'aspecte final del circuit RTL del mòdul "CalculPosicio2" creat per l'ISE és el mostrat a la Figura 63. S'aprecia en aquesta tots els mòduls esmentats del segon bloc, així com les seues connexions. Per sobre d'aquest mòdul es troba "Huff_tree2", els continguts del qual són el mòdul "cp2" i "controlArbre2"(l'FSM), mostrat a la Figura 64.

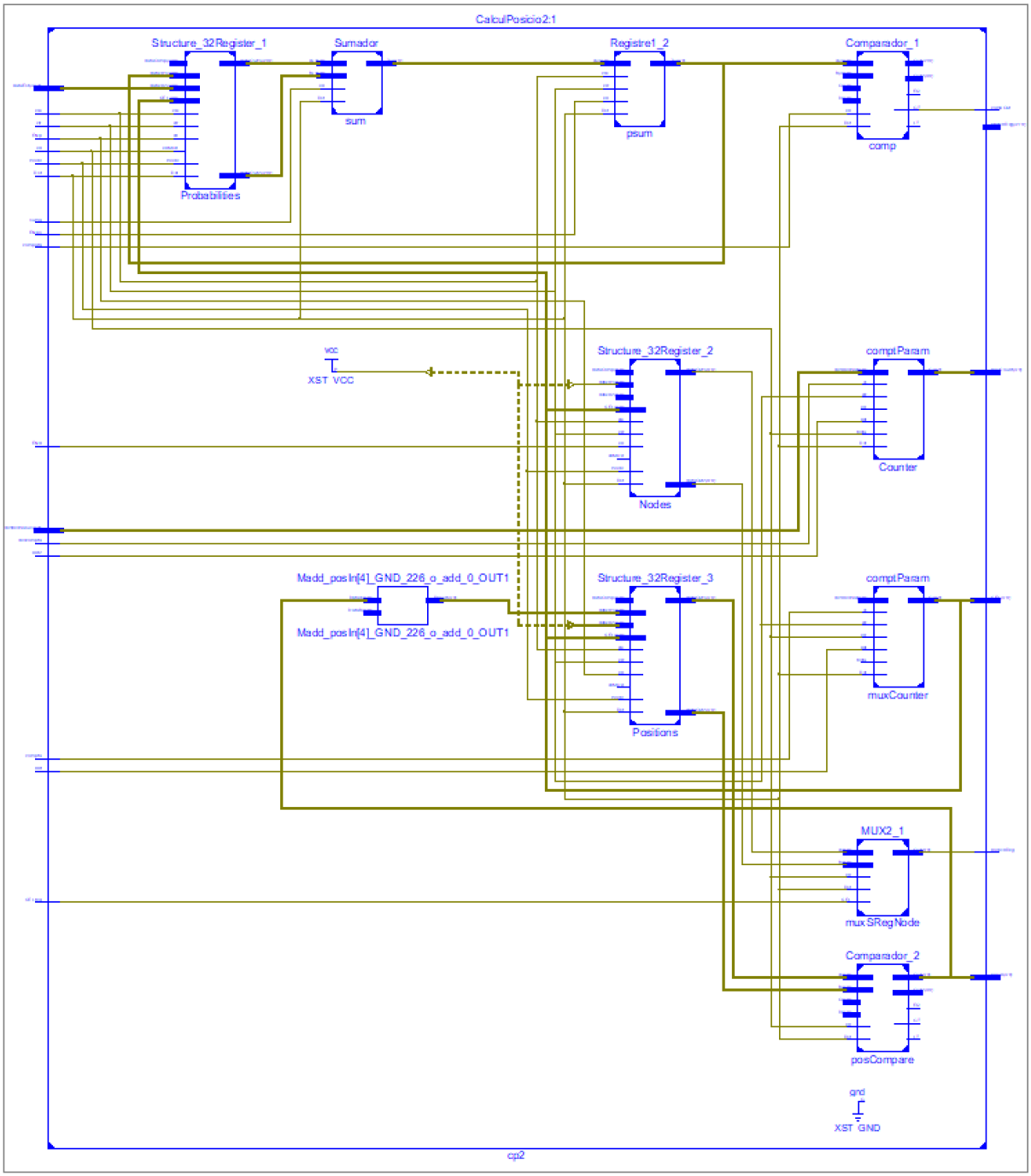

Figura 63. RTL de "CalculPosicio2"

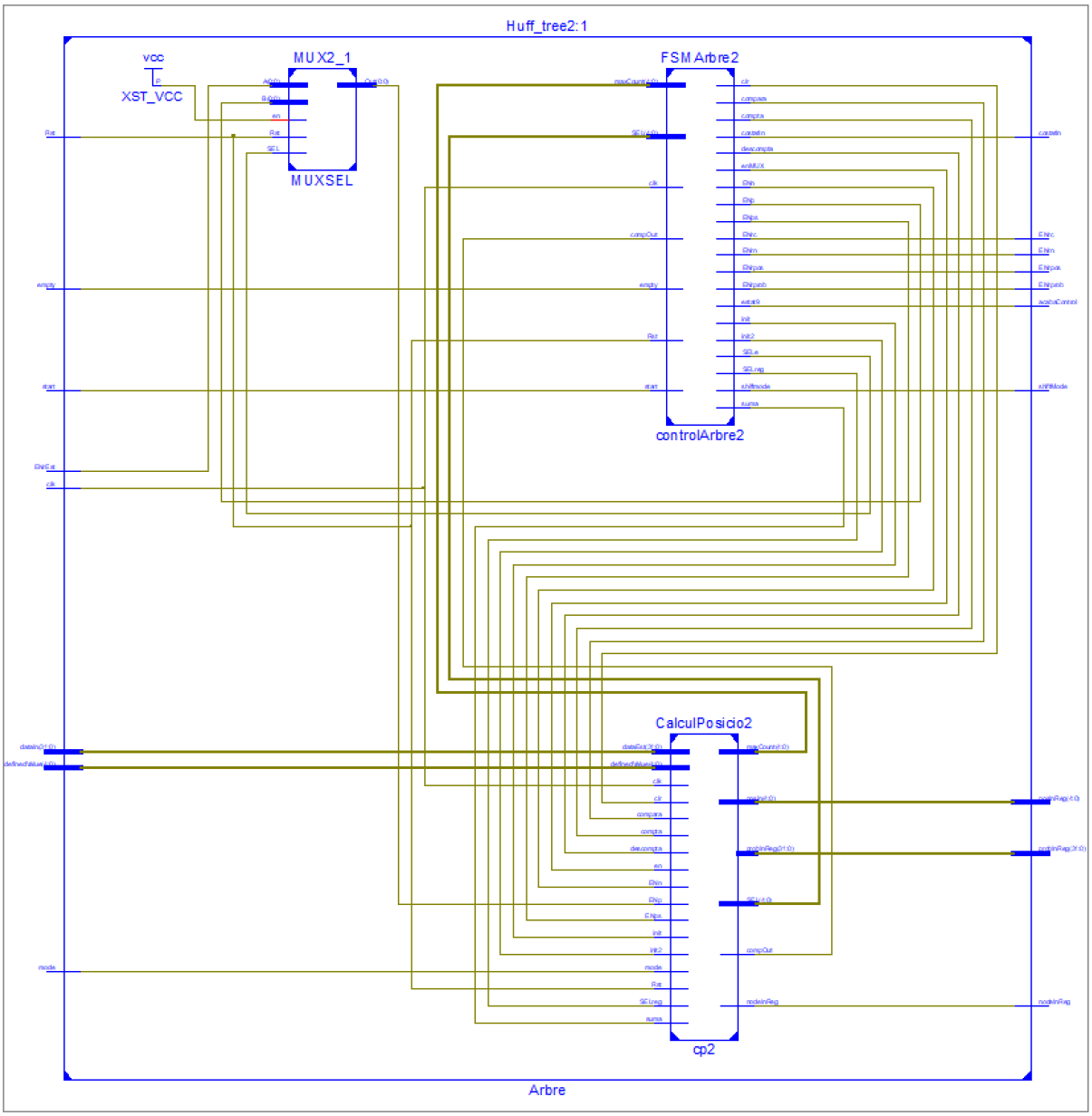

Figura 64. RTL de "Huff_tree2"

9. ESQUEMA RTL DEL BLOC DE CODIFICACIÓ DE L'ARBRE

El bloc de Codificació es compon del mòdul principal (on es troba tota l'electrònica necessària per al funcionament), l'FSM, les cues FIFO on s'emmagatzemen el codi i l'altura dels nodes que no són fulles i un comptador i comparador per dur el recompte de quants valors queden als registres des d'on s'agafen les dades. L'esquema es pot veure a la Figura 65.

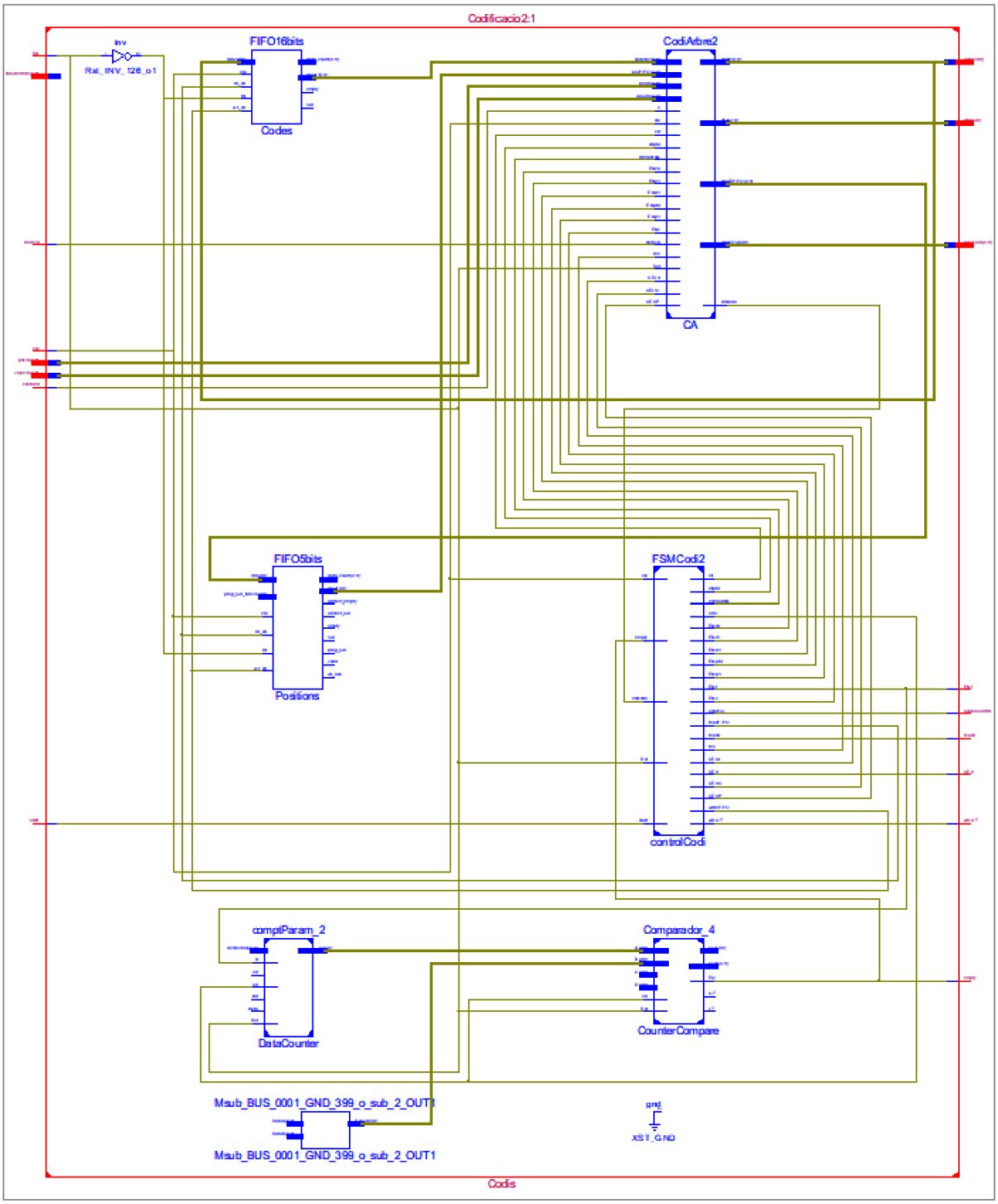

Figura 65. RTL de "Codificacio"

10. REGISTRES D'EIXIDA DEL PROGRAMA

Realitzada la codificació de l'arbre, les dades són enviades a uns registres d'eixida des d'on seran transferits a la LUT implementada a l'FPGA. Aquests registres formen part del mòdul que es presenta a la Figura 66. A dintre d'aquest es troba un mòdul de 32 registres on van els codis, un altre igual on van el nombre de bits i un tercer mòdul que dirigeix les dades d'entrada als registres corresponents.

```
module LUTcomplet #(parameter Bits=16)(
   input Rst, clk, en,
   input write,
   input [4:0] nBitIn, value,
   input [Bits-1:0] codeIn,
   output [Bits-1:0] codeOut16N, codeOut15N, codeOut14N, codeOut13N,
   output [Bits-1:0] codeOut12N, codeOut11N, codeOut10N, codeOut9N,
   output [Bits-1:0] codeOut8N, codeOut7N, codeOut6N, codeOut5N,
   output [Bits-1:0] codeOut4N, codeOut3N, codeOut2N, codeOut1N,
   output [Bits-1:0] codeOut0, codeOut1, codeOut2, codeOut3,
   output [Bits-1:0] codeOut4, codeOut5, codeOut6, codeOut7,
   output [Bits-1:0] codeOut8, codeOut9, codeOut10, codeOut11,
   output [Bits-1:0] codeOut12, codeOut13, codeOut14, codeOut15,
   output [4:0] nBitOut16N, nBitOut15N, nBitOut14N, nBitOut13N,
   output [4:0] nBitOut12N, nBitOut11N, nBitOut10N, nBitOut9N,
   output [4:0] nBitOut8N, nBitOut7N, nBitOut6N, nBitOut5N,
   output [4:0] nBitOut4N, nBitOut3N, nBitOut2N, nBitOut1N,
   output [4:0] nBitOut0, nBitOut1, nBitOut2, nBitOut3,
   output [4:0] nBitOut4, nBitOut5, nBitOut6, nBitOut7,
   output [4:0] nBitOut8, nBitOut9, nBitOut10, nBitOut11,
   output [4:0] nBitOut12, nBitOut13, nBitOut14, nBitOut15
   );

// Register enable
   wire en16N, en15N, en14N, en13N, en12N, en11N, en10N, en9N;
   wire en8N, en7N, en6N, en5N, en4N, en3N, en2N, en1N;
   wire en0, en1, en2, en3, en4, en5, en6, en7;
   wire en8, en9, en10, en11, en12, en13, en14, en15;

```

Figura 66. Entrades, eixides i variables internes del mòdul "LUTcomplet"

A la Figura 67 es poden veure el mòdul que adreça als registres pertinents i el mòdul on es troben els registres dels codis. Com pot observar-se, les habilitacions dels registres vénen del primer mòdul. Cadascun dels 32 registres de que es compon es corresponen a cadascun dels 32 valors escollits a l'inici com a més probables a l'experiment.

```
29
30  // Memory addresses depending on the input value
31     MemoryAddress writing(
32        .Rst(Rst), .en(write), .SEL(value),
33        .Out16n(en16N), .Out15n(en15N), .Out14n(en14N), .Out13n(en13N),
34        .Out12n(en12N), .Out11n(en11N), .Out10n(en10N), .Out9n(en9N),
35        .Out8n(en8N), .Out7n(en7N), .Out6n(en6N), .Out5n(en5N),
36        .Out4n(en4N), .Out3n(en3N), .Out2n(en2N), .Out1n(en1N),
37        .Out0(en0), .Out1(en1), .Out2(en2), .Out3(en3),
38        .Out4(en4), .Out5(en5), .Out6(en6), .Out7(en7),
39        .Out8(en8), .Out9(en9), .Out10(en10), .Out11(en11),
40        .Out12(en12), .Out13(en13), .Out14(en14), .Out15(en15)
41        );
42
43  // LUTs --------------
44     codeLUT cLUT(
45        .Rst(Rst), .clk(clk), .codeIn(codeIn),
46        .en16N(en16N), .en15N(en15N), .en14N(en14N), .en13N(en13N),
47        .en12N(en12N), .en11N(en11N), .en10N(en10N), .en9N(en9N),
48        .en8N(en8N), .en7N(en7N), .en6N(en6N), .en5N(en5N),
49        .en4N(en4N), .en3N(en3N), .en2N(en2N), .en1N(en1N),
50        .en0(en0), .en1(en1), .en2(en2), .en3(en3),
51        .en4(en4), .en5(en5), .en6(en6), .en7(en7),
52        .en8(en8), .en9(en9), .en10(en10), .en11(en11),
53        .en12(en12), .en13(en13), .en14(en14), .en15(en15),
54        .codeOut16N(codeOut16N), .codeOut15N(codeOut15N), .codeOut14N(codeOut14N), .codeOut13N(codeOut13N),
55        .codeOut12N(codeOut12N), .codeOut11N(codeOut11N), .codeOut10N(codeOut10N), .codeOut9N(codeOut9N),
56        .codeOut8N(codeOut8N), .codeOut7N(codeOut7N), .codeOut6N(codeOut6N), .codeOut5N(codeOut5N),
57        .codeOut4N(codeOut4N), .codeOut3N(codeOut3N), .codeOut2N(codeOut2N), .codeOut1N(codeOut1N),
58        .codeOut0(codeOut0), .codeOut1(codeOut1), .codeOut2(codeOut2), .codeOut3(codeOut3),
59        .codeOut4(codeOut4), .codeOut5(codeOut5), .codeOut6(codeOut6), .codeOut7(codeOut7),
60        .codeOut8(codeOut8), .codeOut9(codeOut9), .codeOut10(codeOut10), .codeOut11(codeOut11),
61        .codeOut12(codeOut12), .codeOut13(codeOut13), .codeOut14(codeOut14), .codeOut15(codeOut15)
62        );
63
```

Figura 67. Mòduls interns de "LUTcomplet"

El mòdul de registres, bé el dels codis, bé el dels nombres de bits, es veu programat a la Figura 68. El mòdul "Memory Address" indica a quin registre ha d'anar cada dada segons el valor que porta associat i que s'envia des de la codificació cada cop que es troba un node "fulla". La programació d'aquest mòdul es pot veure a la Figura 69.

```
// Registers----------
   Registrel #(.Bits(Bits)) rl6_neg(.en(enl6N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut16N));
   Registrel #(.Bits(Bits)) rl5_neg(.en(enl5N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut15N));
   Registrel #(.Bits(Bits)) rl4_neg(.en(enl4N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut14N));
   Registrel #(.Bits(Bits)) rl3_neg(.en(enl3N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut13N));
   Registrel #(.Bits(Bits)) rl2_neg(.en(enl2N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut12N));
   Registrel #(.Bits(Bits)) rl1_neg(.en(enl1N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut11N));
   Registrel #(.Bits(Bits)) rl0_neg(.en(enl0N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut10N));
   Registrel #(.Bits(Bits)) r9_neg(.en(en9N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut9N));
   Registrel #(.Bits(Bits)) r8_neg(.en(en8N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut8N));
   Registrel #(.Bits(Bits)) r7_neg(.en(en7N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut7N));
   Registrel #(.Bits(Bits)) r6_neg(.en(en6N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut6N));
   Registrel #(.Bits(Bits)) r5_neg(.en(en5N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut5N));
   Registrel #(.Bits(Bits)) r4_neg(.en(en4N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut4N));
   Registrel #(.Bits(Bits)) r3_neg(.en(en3N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut3N));
   Registrel #(.Bits(Bits)) r2_neg(.en(en2N), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut2N));
   Registrel #(.Bits(Bits)) rl_neg(.en(enlN), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOutlN));

   Registrel #(.Bits(Bits)) r0(.en(en0), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut0));
   Registrel #(.Bits(Bits)) rl(.en(enl), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOutl));
   Registrel #(.Bits(Bits)) r2(.en(en2), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut2));
   Registrel #(.Bits(Bits)) r3(.en(en3), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut3));
   Registrel #(.Bits(Bits)) r4(.en(en4), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut4));
   Registrel #(.Bits(Bits)) r5(.en(en5), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut5));
   Registrel #(.Bits(Bits)) r6(.en(en6), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut6));
   Registrel #(.Bits(Bits)) r7(.en(en7), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut7));
   Registrel #(.Bits(Bits)) r8(.en(en8), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut8));
   Registrel #(.Bits(Bits)) r9(.en(en9), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut9));
   Registrel #(.Bits(Bits)) rl0(.en(enl0), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut10));
   Registrel #(.Bits(Bits)) rl1(.en(enl1), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut11));
   Registrel #(.Bits(Bits)) rl2(.en(enl2), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut12));
   Registrel #(.Bits(Bits)) rl3(.en(enl3), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut13));
   Registrel #(.Bits(Bits)) rl4(.en(enl4), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut14));
   Registrel #(.Bits(Bits)) rl5(.en(enl5), .Rst(Rst), .clk(clk), .A(codeIn), .Z(codeOut15));
//--------------------
```

Figura 68. Registres de "codeLUT"

```
assign Out16n = Rst ? (en ? (SEL==-5'h10 ? 1'b1 : 1'b0) : 0) : 0;
assign Out15n = Rst ? (en ? (SEL==-5'h0f ? 1'b1 : 1'b0) : 0) : 0;
assign Out14n = Rst ? (en ? (SEL==-5'h0e ? 1'b1 : 1'b0) : 0) : 0;
assign Out13n = Rst ? (en ? (SEL==-5'h0d ? 1'b1 : 1'b0) : 0) : 0;
assign Out12n = Rst ? (en ? (SEL==-5'h0c ? 1'b1 : 1'b0) : 0) : 0;
assign Out11n = Rst ? (en ? (SEL==-5'h0b ? 1'b1 : 1'b0) : 0) : 0;
assign Out10n = Rst ? (en ? (SEL==-5'h0a ? 1'b1 : 1'b0) : 0) : 0;
assign Out9n = Rst ? (en ? (SEL==-5'h09 ? 1'b1 : 1'b0) : 0) : 0;
assign Out8n = Rst ? (en ? (SEL==-5'h08 ? 1'b1 : 1'b0) : 0) : 0;
assign Out7n = Rst ? (en ? (SEL==-5'h07 ? 1'b1 : 1'b0) : 0) : 0;
assign Out6n = Rst ? (en ? (SEL==-5'h06 ? 1'b1 : 1'b0) : 0) : 0;
assign Out5n = Rst ? (en ? (SEL==-5'h05 ? 1'b1 : 1'b0) : 0) : 0;
assign Out4n = Rst ? (en ? (SEL==-5'h04 ? 1'b1 : 1'b0) : 0) : 0;
assign Out3n = Rst ? (en ? (SEL==-5'h03 ? 1'b1 : 1'b0) : 0) : 0;
assign Out2n = Rst ? (en ? (SEL==-5'h02 ? 1'b1 : 1'b0) : 0) : 0;
assign Out1n = Rst ? (en ? (SEL==-5'h01 ? 1'b1 : 1'b0) : 0) : 0;

assign Out0 = Rst ? (en ? (SEL==5'h00 ? 1'b1 : 1'b0) : 0) : 0;
assign Out1 = Rst ? (en ? (SEL==5'h01 ? 1'b1 : 1'b0) : 0) : 0;
assign Out2 = Rst ? (en ? (SEL==5'h02 ? 1'b1 : 1'b0) : 0) : 0;
assign Out3 = Rst ? (en ? (SEL==5'h03 ? 1'b1 : 1'b0) : 0) : 0;
assign Out4 = Rst ? (en ? (SEL==5'h04 ? 1'b1 : 1'b0) : 0) : 0;
assign Out5 = Rst ? (en ? (SEL==5'h05 ? 1'b1 : 1'b0) : 0) : 0;
assign Out6 = Rst ? (en ? (SEL==5'h06 ? 1'b1 : 1'b0) : 0) : 0;
assign Out7 = Rst ? (en ? (SEL==5'h07 ? 1'b1 : 1'b0) : 0) : 0;
assign Out8 = Rst ? (en ? (SEL==5'h08 ? 1'b1 : 1'b0) : 0) : 0;
assign Out9 = Rst ? (en ? (SEL==5'h09 ? 1'b1 : 1'b0) : 0) : 0;
assign Out10 = Rst ? (en ? (SEL==5'h0a ? 1'b1 : 1'b0) : 0) : 0;
assign Out11 = Rst ? (en ? (SEL==5'h0b ? 1'b1 : 1'b0) : 0) : 0;
assign Out12 = Rst ? (en ? (SEL==5'h0c ? 1'b1 : 1'b0) : 0) : 0;
assign Out13 = Rst ? (en ? (SEL==5'h0d ? 1'b1 : 1'b0) : 0) : 0;
assign Out14 = Rst ? (en ? (SEL==5'h0e ? 1'b1 : 1'b0) : 0) : 0;
assign Out15 = Rst ? (en ? (SEL==5'h0f ? 1'b1 : 1'b0) : 0) : 0;
```

Figura 69. Programació de "Memory Address"

11. ESQUEMES RTL DELS MÒDULS SUPERIORS

Els mòduls "Huff_tree" i "Codificacio" mostrats a la Figura 64 i a la Figura 65 pertanyen a un mòdul superior anomenat "ArbreComplet2". Conté també els registres de desplaçament "SRpos", "SRcos" i "SRnode" (programat està també "SRprob", però com no s'utilitza ISE l'elimina del circuit final) i el multiplexor per a les dues habilitacions diferents que hi ha per a aquests, tal i com es pot observar a la Figura 70. En aquest mòdul superior és on es pot observar la primera de les FSM, "FSMAComplet".

Els registres de desplaçament "SRprob", "SRpos", "SRcos" i "SRnode" és on s'acumulen les dades a l'eixir de la construcció de l'arbre fins que entren a la codificació. Es recorda que eren necessaris perquè la construcció de l'arbre es fa començant pels menors recomptes i la codificació ho fa des del major.

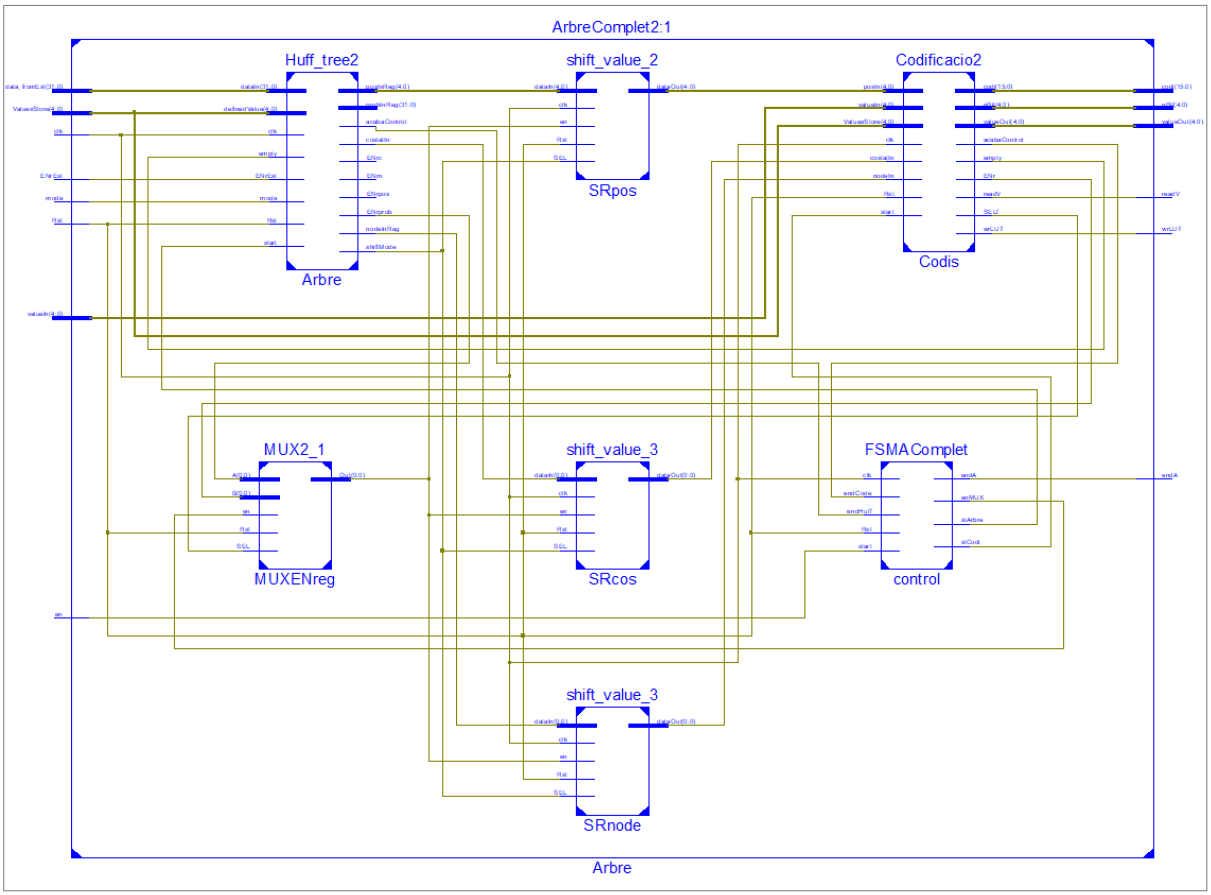

Figura 70. RTL del mòdul "ArbreComplet2"

Al seu torn, els mòduls "estadística2" i "FSMEst2" estan incorporats a "estadisticaComplet" (Figura 71). I aquest últim junt amb "ArbreComplet2", al "Top module" del programa, "TopCal", representat a la Figura 72. També inclou el mòdul de lectura, que finalment és sols un multiplexor per als canals d'entrada; la FIFO "Values1", que conté els valors ordenats de major a menor probabilitat a "estadística2"; el mòdul anomenat com a "LUT", que per qüestions de comoditat en la programació s'ha deixat en aquest mòdul i no dins de "ArbreComplet2", i l'FSM del "Top module", "FSMTop" (anomenada internament com a "ControlTop").

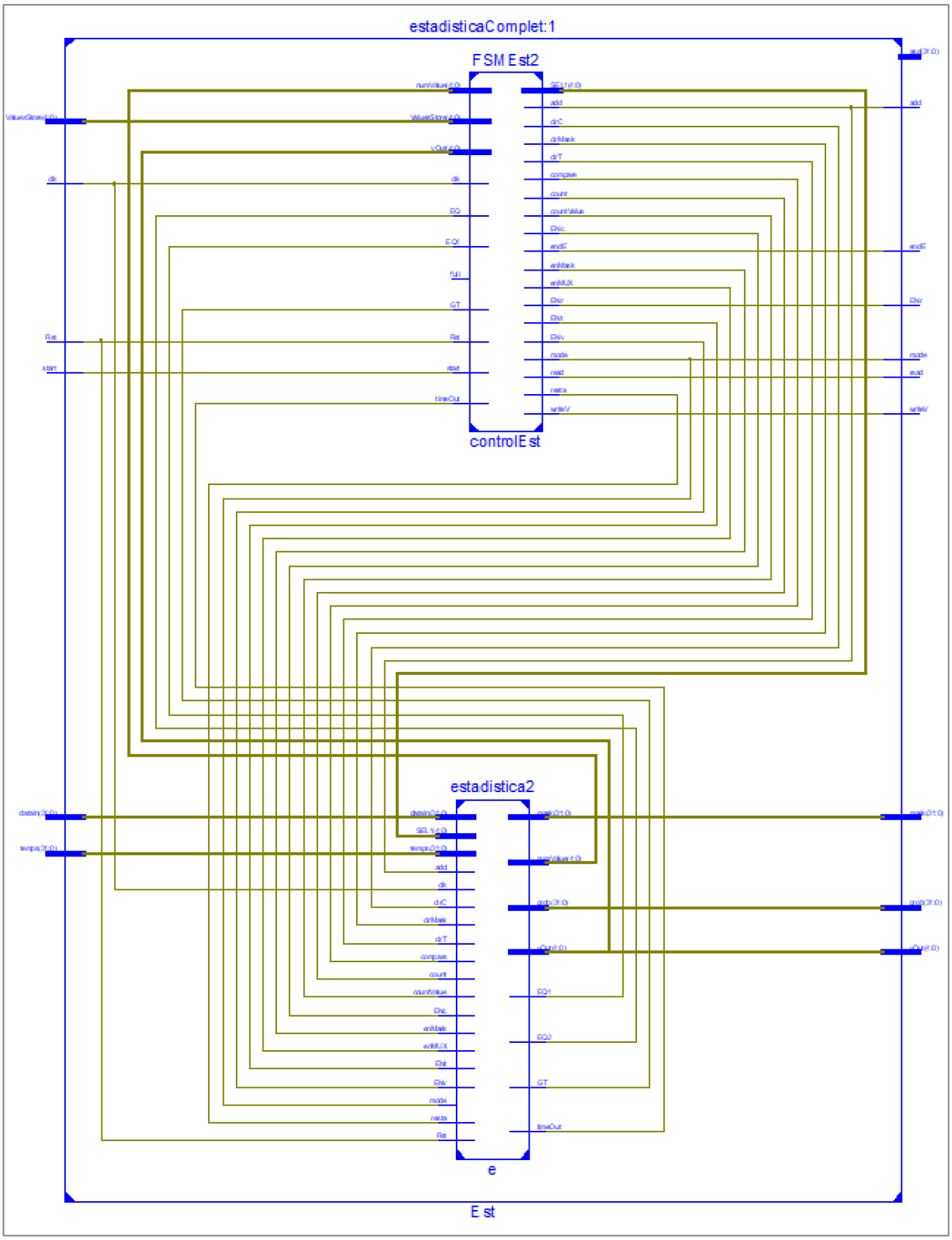

Figura 71. RTL del mòdul "estadisticaComplet"

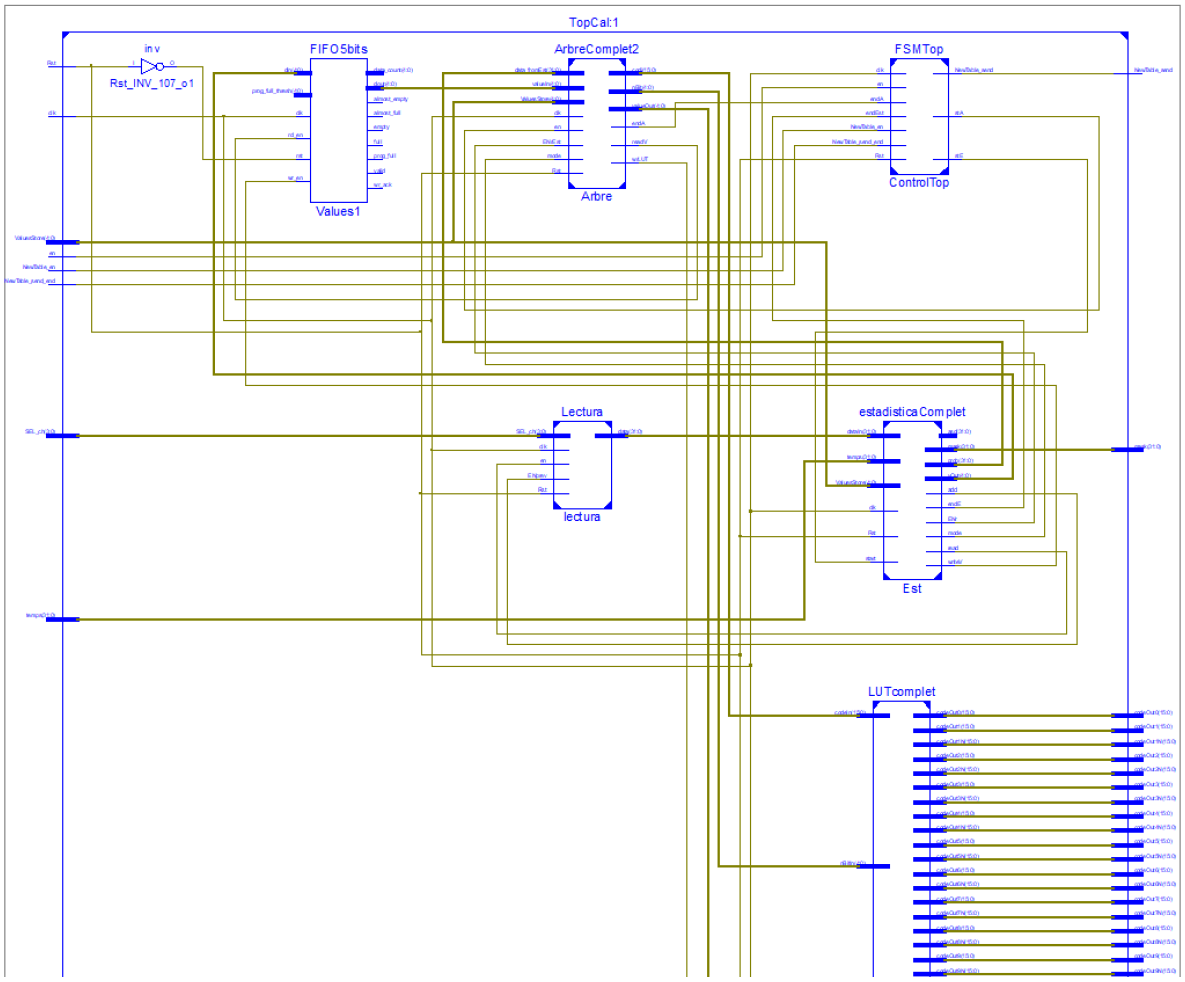

Figura 72. RTL del mòdul "TopCal"

DISSENY I IMPLEMENTACIÓ D'UN ARBRE DE CODIFICACIÓ HUFFMAN EN TEMPS REAL EN FPGA PER A L'OPTIMITZACIÓ DE LA COMPRESSIÓ DE DADES DEL PLA D'ENERGIA DE L'EXPERIMENT NEXT

Annex núm. 2: Detalls de la verificació

1. INTRODUCCIÓ

En el present annex, es detallarà el banc de proves utilitzat per a simular el circuit implementat del projecte. S'explicarà els mòduls que el componen, així com el camí que segueix la simulació i el perquè dels estímuls que s'envien al circuit.

2. BANC DE PROVES

Abans de continuar amb les simulacions, es procedeix a explicar els anomenats "Test Bench" (Banc de Proves). Per poder dur a terme les simulacions, el programa necessita un entorn contextualitzat, és a dir, se li ha de proporcionar valors a les entrades en un procés temporal. Açò es fa mitjançant una programació comportamental en Verilog, la qual consta de la instanciació del mòduls necessaris (Figura 73), un "Procedural Block" de tipus "initial" en el que es crea el rellotge del sistema (que per a aquest projecte ha de ser de 40 MHz) i un altre de tipus "always" (Figura 74), on es proporcionen els valor de les entrades en un moment donat del procés, a part de la durada d'aquest. Les variables utilitzades al Test es mostren a la Figura 75.

```
ReadFile #(.Bits(12)) readFile(
   .clk(clk), .en(read),
   .read_ch1(read_ch1), .read_ch2(read_ch2), .read_ch3(read_ch3), .read_ch4(read_ch4),
   .read_ch5(read_ch5), .read_ch6(read_ch6), .read_ch7(read_ch7), .read_ch8(read_ch8),
   .read_ch9(read_ch9), .read_ch10(read_ch10), .read_ch11(read_ch11), .read_ch12(read_ch12)
   );

TopCal TC(
   .clk(clk), .Rst(Rst), .en(en),
   .NewTable_en(NewTable_en), .NewTable_send_end(NewTable_send_end), .datach12_in(datach12_in),
   .SEL_ch(SEL_ch), .temps(temps), .ValuesStore(ValuesStore),
   .NewTable_send(NewTable_send), .mask(mask), .read_data(read),
   .codeOut16N(codeOut16N), .codeOut15N(codeOut15N), .codeOut14N(codeOut14N), .codeOut13N(codeOut13N),
   .codeOut12N(codeOut12N), .codeOut11N(codeOut11N), .codeOut10N(codeOut10N), .codeOut9N(codeOut9N),
   .codeOut8N(codeOut8N), .codeOut7N(codeOut7N), .codeOut6N(codeOut6N), .codeOut5N(codeOut5N),
   .codeOut4N(codeOut4N), .codeOut3N(codeOut3N), .codeOut2N(codeOut2N), .codeOut1N(codeOut1N),
   .codeOut0(codeOut0), .codeOut1(codeOut1), .codeOut2(codeOut2), .codeOut3(codeOut3),
   .codeOut4(codeOut4), .codeOut5(codeOut5), .codeOut6(codeOut6), .codeOut7(codeOut7),
   .codeOut8(codeOut8), .codeOut9(codeOut9), .codeOut10(codeOut10), .codeOut11(codeOut11),
   .codeOut12(codeOut12), .codeOut13(codeOut13), .codeOut14(codeOut14), .codeOut15(codeOut15),

   .nBitOut16N(nBitOut16N), .nBitOut15N(nBitOut15N), .nBitOut14N(nBitOut14N), .nBitOut13N(nBitOut13N),
   .nBitOut12N(nBitOut12N), .nBitOut11N(nBitOut11N), .nBitOut10N(nBitOut10N), .nBitOut9N(nBitOut9N),
   .nBitOut8N(nBitOut8N), .nBitOut7N(nBitOut7N), .nBitOut6N(nBitOut6N), .nBitOut5N(nBitOut5N),
   .nBitOut4N(nBitOut4N), .nBitOut3N(nBitOut3N), .nBitOut2N(nBitOut2N), .nBitOut1N(nBitOut1N),
   .nBitOut0(nBitOut0), .nBitOut1(nBitOut1), .nBitOut2(nBitOut2), .nBitOut3(nBitOut3),
   .nBitOut4(nBitOut4), .nBitOut5(nBitOut5), .nBitOut6(nBitOut6), .nBitOut7(nBitOut7),
   .nBitOut8(nBitOut8), .nBitOut9(nBitOut9), .nBitOut10(nBitOut10), .nBitOut11(nBitOut11),
   .nBitOut12(nBitOut12), .nBitOut13(nBitOut13), .nBitOut14(nBitOut14), .nBitOut15(nBitOut15)
   );
```

Figura 73. Mòduls utilitzats per al Banc de Proves: lectura i el mòdul Top del projecte

```
    initial
    begin
        $display("Running Testbench");
        Rst=1'b0; clk=1'b0; en=1'b0; NewTable_en=1'b0; NewTable_send_end=1'b0;
        SEL_ch=4'h1; temps=32'h0;
        ValuesStore=5'h11;
        repeat(5) @(posedge clk);
        #55 Rst=1'b1;
        repeat(5) @(posedge clk);
        en=1'b1; NewTable_en=1'b1;
        temps=32'h0007_EF40;     //h19640
        @(posedge clk);
        en=1'b0;
        repeat(680000) @(posedge clk);
        NewTable_send_end=1'b1;
        repeat(5) @(posedge clk);
        Rst=1'b0;
        repeat(3) @(posedge clk);
        $stop;
    end

    always
    #12.5 clk=!clk;

endmodule
```

Figura 74. Blocs de funcionament del Banc de Proves

```
`timescale 1ns / 1ps
module TestTop2();

   reg clk, Rst, en;
   reg NewTable_en, NewTable_send_end;
   reg [4:0] ValuesStore;
   reg [31:0] temps;
   reg [3:0] SEL_ch;

   wire NewTable_send, read;
   wire [11:0] read_ch1, read_ch2, read_ch3, read_ch4;
   wire [11:0] read_ch5, read_ch6, read_ch7, read_ch8;
   wire [11:0] read_ch9, read_ch10, read_ch11, read_ch12;
   wire [31:0] mask;
   wire [143:0] datach12_in;

   assign datach12_in[(1*12)-1:0] = read_ch1;
   assign datach12_in[(2*12)-1:1*12] = read_ch2;
   assign datach12_in[(3*12)-1:2*12] = read_ch3;
   assign datach12_in[(4*12)-1:3*12] = read_ch4;
   assign datach12_in[(5*12)-1:4*12] = read_ch5;
   assign datach12_in[(6*12)-1:5*12] = read_ch6;
   assign datach12_in[(7*12)-1:6*12] = read_ch7;
   assign datach12_in[(8*12)-1:7*12] = read_ch8;
   assign datach12_in[(9*12)-1:8*12] = read_ch9;
   assign datach12_in[(10*12)-1:9*12] = read_ch10;
   assign datach12_in[(11*12)-1:10*12] = read_ch11;
   assign datach12_in[(12*12)-1:11*12] = read_ch12;

   wire [15:0] codeOut16N, codeOut15N, codeOut14N, codeOut13N;
   wire [15:0] codeOut12N, codeOut11N, codeOut10N, codeOut9N;
   wire [15:0] codeOut8N, codeOut7N, codeOut6N, codeOut5N;
   wire [15:0] codeOut4N, codeOut3N, codeOut2N, codeOut1N;
   wire [15:0] codeOut0, codeOut1, codeOut2, codeOut3;
   wire [15:0] codeOut4, codeOut5, codeOut6, codeOut7;
   wire [15:0] codeOut8, codeOut9, codeOut10, codeOut11;
   wire [15:0] codeOut12, codeOut13, codeOut14, codeOut15;

   wire [4:0] nBitOut16N, nBitOut15N, nBitOut14N, nBitOut13N;
   wire [4:0] nBitOut12N, nBitOut11N, nBitOut10N, nBitOut9N;
   wire [4:0] nBitOut8N, nBitOut7N, nBitOut6N, nBitOut5N;
   wire [4:0] nBitOut4N, nBitOut3N, nBitOut2N, nBitOut1N;
   wire [4:0] nBitOut0, nBitOut1, nBitOut2, nBitOut3;
   wire [4:0] nBitOut4, nBitOut5, nBitOut6, nBitOut7;
   wire [4:0] nBitOut8, nBitOut9, nBitOut10, nBitOut11;
   wire [4:0] nBitOut12, nBitOut13, nBitOut14, nBitOut15;
```

Figura 75. Variables internes del Banc de Proves

A la Figura 73, s'observa que hi ha un mòdul anomenat "readFile" al Test[7]. Aquest és un mòdul proporcionat pel Tutor per poder llegir fitxers des del programa. Com ja s'ha comentat, el programa rebrà dades de d'un altre mòdul del que no es disposa; per aquest motiu, era necessari un mòdul com aquest que permetera llegir dades. Per a adaptar l'eixida d'aquest mòdul a l'entrada del programa del projecte, estan els "assign" que es veuen a la Figura 75, que construeixen una sola paraula a partir de les 12 eixides. Aquestes són separades després a l'interior del programa, ja que sols es realitza el procés sobre un canal.

La Figura 74 mostra el bucle "always" per a una simulació de 5 esdeveniments i un arbre de 17 valors. Les diverses simulacions realitzades varien aquests dos valors del Banc (esdeveniments processats i quantitat de valors a l'arbre). La variació del segon és òbvia, mitjançant la variable "ValuesStore", però la primera es fa a l'interior del mòdul "readFile"; a més, s'ha de modificar la variable "temps". Els fitxers

[7] Aquest mòdul és explicat al següent Annex

utilitzats tenen 52000 valors per esdeveniment. La variable temps va amb el rellotge del sistema i cada dos cicles fa el recompte d'un valor. És a dir, per a un esdeveniment amb 52 000 valors, seran necessaris 104 000 cicles (equival a 2,6 ms, el rellotge té un període de 25 ns). En el cas de la Figura 74 on hi ha 5 esdeveniments, seran necessaris 520 000 cicles (7EF40 en hexadecimal).

Per a les simulacions d'aquest projecte s'han utilitzat pocs esdeveniments en comparació amb el total. El temps de processat de la totalitat de les dades és elevat, per la qual cosa s'ha posat aquesta variable de temps. El sistema final sols tindrà en compte uns quants esdeveniments també.

DISSENY I IMPLEMENTACIÓ D'UN ARBRE DE CODIFICACIÓ HUFFMAN EN TEMPS REAL EN FPGA PER A L'OPTIMITZACIÓ DE LA COMPRESSIÓ DE DADES DEL PLA D'ENERGIA DE L'EXPERIMENT NEXT

Annex núm. 3: Programes utilitzats

1. INTRODUCCIÓ

A la llarga d'aquest projecte s'han anat presentant diversos programes utilitzat que no en formen part d'aquest. Seria convenient per açò, realitzar una breu presentació d'aquests per fer més sòlides les explicacions i verificacions realitzades. Els programes no formen part del projecte final que anirà incorporat a l'FPGA, però han sigut de gran ajuda per al desenvolupament del treball.

2. MÒDUL "read_file"

Les dades d'entrada al projecte vénen d'un mòdul d'adquisició de dades ja implementat. Com no es disposa d'aquest, s'ha hagut de fer ús del mòdul "read_file", que permet la lectura d'un fitxer de dades i passar-les aquestes a un format de 12 variables pertanyents a cadascun dels PMT de l'experiment. La programació és la presentada a la Figura 76 i a la Figura 77. Al circuit implementat, posteriorment se seleccionarà amb un multiplexor d'entrada el canal pel qual es llegeixen les dades.

```
1  `timescale 1ns / 1ps
2  module ReadFile #(parameter Bits=5)(
3     input clk, en,
4     output reg [Bits-1:0] read_ch1, read_ch2, read_ch3, read_ch4,
5     output reg [Bits-1:0] read_ch5, read_ch6, read_ch7, read_ch8,
6     output reg [Bits-1:0] read_ch9, read_ch10, read_ch11, read_ch12
7     );
8
9     integer input_file, err_read;
10    parameter EOF=32'hFFFF_FFFF; // Fin de fichero
11
12    always @(posedge clk)
13    begin
14       //every clock cycle an input value is stored in D reg
15       if (en)
16       begin
17       #4 err_read=$fscanf(input_file,
18       "%d %d %d %d %d %d %d %d %d %d %d %d",
19                                                    read_ch1,
20                                                    read_ch2,
21                                                    read_ch3,
22                                                    read_ch4,
23                                                    read_ch5,
24                                                    read_ch6,
25                                                    read_ch7,
26                                                    read_ch8,
```

Figura 76. Mòdul "read_file" part 1

```
 9     integer input_file, err_read;
10     parameter EOF=32'hFFFF_FFFF; // Fin de fichero
11
12     always @(posedge clk)
13     begin
14        //every clock cycle an input value is stored in D reg
15        if (en)
16        begin
17        #4 err_read=$fscanf(input_file,
18        "%d %d %d %d %d %d %d %d %d %d %d %d",
19                                                    read_ch1,
20                                                    read_ch2,
21                                                    read_ch3,
22                                                    read_ch4,
23                                                    read_ch5,
24                                                    read_ch6,
25                                                    read_ch7,
26                                                    read_ch8,
27                                                    read_ch9,
28                                                    read_ch10,
29                                                    read_ch11,
30                                                    read_ch12
31                                                    );
32
33           if (err_read == EOF) //Si llego al final del archivo, vuelvo al principio
34           begin
35           // $stop;
36              $fclose(input_file);
37              input_file=$fopen("C:/Users/Guillem/Documents/TFM/6203_Kr_Th/6203_0000_0_0to5.txt","r");
38              err_read=$fscanf(input_file,
39              "%d %d %d %d %d %d %d %d %d %d %d %d",
40                                                    read_ch1,
41                                                    read_ch2,
42                                                    read_ch3,
43                                                    read_ch4,
44                                                    read_ch5,
45                                                    read_ch6,
46                                                    read_ch7,
47                                                    read_ch8,
48                                                    read_ch9,
49                                                    read_ch10,
50                                                    read_ch11,
51                                                    read_ch12
52                                                    );
53
54           end
55        end
56     end
57  endmodule
```

Figura 77. Mòdul "read_file" part 2

3. PROGRAMA DE PYTHON

El programa escrit en llenguatge Python ha sigut utilitzat en primera instància per entendre el funcionament de l'algorisme i després, per realitzar la cerca dels valors més probables i fer les comprovacions pertinents amb els arbres que codifica. Les figures següents presenten les parts més importants del programa que s'han utilitzat.

La Figura 78 mostra les llibreries que han sigut utilitzades per al programa. Aquestes són necessàries per fer els diversos càlculs i per poder organitzar les dades en l'estructura en forma d'arbre.

```
from glob import glob
import tables as tb
import numpy as np
import matplotlib.pylab as plt
import collections
%matplotlib inline
import heapq
from heapq import heappush, heappop
```

Figura 78. Llibreries utilitzades al programa

A la següent imatge, Figura 79, està programat el tractament inicial de les dades. Primerament s'obtenen les dades des d'un fitxer. Després, s'obté la diferència entre la dada anterior i l'actual. Aquest programa s'ha adaptat per llegir solament d'un canal, tal i com ho fa el circuit implementat al treball.

```
diffs = []

i = 0

refch = 0
nevt = 150 #data.shape[0]
channel = 1 #data.shape[1]

for fname in files:
    # Read file
    h5in = tb.open_file(fname)
    data = h5in.root.RD.pmtrwf[:]

    #Compute baselines, aplica la moda a les dades adquirides anteriors
    baselines = np.apply_along_axis(mode, 2, data)

    #Sustract baselines
    for evt in range(nevt):

        #for sns in range(channel):
            #diffs.append(data[evt][sns] - baselines[evt][sns])
            diffs.append(data[evt][0][:data.shape[2]-1] - data[evt][0][1:])
            #if (sns==refch):
             #   diffs.append(data[evt][sns] - baselines[evt][sns])
            #else:
             #   diffs.append(data[evt][sns] - baselines[evt][sns] - (data[ev

diffs = np.concatenate(diffs)
```

Figura 79. Càlcul dels valors diferencials

Les tres figures següents són les funcions utilitzades per obtenir l'arbre de codis. La Figura 80 és la funció que calcula les probabilitats de cadascun dels valors que hi ha al fitxer. Aquesta funció és la que ha sigut utilitzada per obtenir el rang de dades seleccionat al circuit implementat al projecte. La Figura 81 construeix l'arbre a partir de les probabilitats obtingudes a l'anterior i amb un nombre de valors determinat abans. Per últim, la Figura 82 fa la codificació de l'arbre.

```
def get_probabilities(content):
    total = len(content) + 1 # Agregamos uno por el caracter FINAL
    c = collections.Counter(content)
    res = {}
    #sortres = {}

    for ch,count in c.items():
        res[ch] = float(count)/total
        print(ch,",",count)
    #res[10000] = 1.0/total

    #sortres = res.items()
    #print(sortres)
    #sortres.sort(sortres, key=lambda x:x[1])

    sortres = sorted(res.items(), key=operator.itemgetter(1))
    #sortres = sortres.reverse()
    return sortres, res
```

Figura 80. Funció per calcular la probabilitat d'una dada

```
def make_tree(probs):
    q = []
    # Agregamos todos los símbolos a la pila
    for ch,pr in probs:# .items():
        # La fila de prioridad está ordenada por
        # prioridad y profundidad
        heapq.heappush(q,(pr,0,ch))

    # Empezamos a mezclar símbolos juntos
    # hasta que la fila tenga un elemento
    while len(q) > 1:
        e1 = heapq.heappop(q) # El símbolo menos probable
        e2 = heapq.heappop(q) # El segundo menos probable

        # Este nuevo nodo tiene probabilidad e1[0]+e2[0]
        # y profundidad mayor al nuevo nodo. Augmenta la profu
        #els valors emmagatzemats són els dels fills
        nw_e = (e1[0]+e2[0],max(e1[1],e2[1])+1,[e1,e2])
        heapq.heappush(q,nw_e)

    return q[0]# Devolvemos el arbol sin la fila
```

Figura 81. Funció per crear l'estructura en forma d'arbre

```
def make_dictionary(tree):

    prefix = ''
    res = {} # La estructura que vamos a devolver
    search_stack = [] # Pila para DFS
    # El último elemento de la lista es el prefijo!
    search_stack.append(tree+("",))
    while len(search_stack) > 0:
        elm = search_stack.pop()
        #print("elm[2] ",elm[2])

        if type(elm[2]) == list:
            # En este caso, el nodo NO es una hoja del árbol,
            # es decir que tiene nodos hijos
            prefix = elm[-1]

            # El hijo derecho tiene "0" en el prefijo
            search_stack.append(elm[2][1]+(prefix+"0",))
            #print("elm der",elm[2][1], "prefix ",prefix+"0")


            # El hijo izquierdo tiene "1" en el prefijo
            search_stack.append(elm[2][0]+(prefix+"1",))
            #print("elm izq",elm[2][0], "prefix ",prefix+"1")


            continue
        else:
            # El nodo es una hoja del árbol, así que
            # obtenemos el código completo y lo agregamos
            code = elm[-1]
            res[elm[2]] = code


        pass
    return res
```

Figura 82. Funció per codificar l'arbre

DISSENY I IMPLEMENTACIÓ D'UN ARBRE DE CODIFICACIÓ HUFFMAN EN TEMPS REAL EN FPGA PER A L'OPTIMITZACIÓ DE LA COMPRESSIÓ DE DADES DEL PLA D'ENERGIA DE L'EXPERIMENT NEXT

Annex núm. 4: Figures ampliades

Figura 55

```
//MUXs----------------
// This multiplexor changes its input depending on the results of the comparator below, it remains the higher one
   MUX33_2 mux1(
      .en(enMUX), .Rst(Rst), .SEL(SEL1),
      .A(prob0), .B(prob1), .b(prob1n), .C(prob2), .c(prob2n), .D(prob3), .d(prob3n), .E(prob4), .e(prob4n),
      .F(prob5), .f(prob5n), .G(prob6), .g(prob6n), .H(prob7), .h(prob7n), .I(prob8), .i(prob8n),
      .J(prob9), .j(prob9n), .K(prob10), .k(prob10n), .L(prob11), .l(prob11n), .M(prob12), .m(prob12n),
      .N(prob13), .n(prob13n), .O(prob14), .o(prob14n), .P(prob15), .p(prob15n), .q(prob16n),
      .value(value1), .Out(data1)
      );

// This multiplexor changes its input depending on the following counter in order to search the next higher value
   MUX33 mux2(
      .en(enMUX), .Rst(Rst), .SEL(SEL2),
      .A(prob0), .B(prob1), .b(prob1n), .C(prob2), .c(prob2n), .D(prob3), .d(prob3n), .E(prob4), .e(prob4n),
      .F(prob5), .f(prob5n), .G(prob6), .g(prob6n), .H(prob7), .h(prob7n), .I(prob8), .i(prob8n),
      .J(prob9), .j(prob9n), .K(prob10), .k(prob10n), .L(prob11), .l(prob11n), .M(prob12), .m(prob12n),
      .N(prob13), .n(prob13n), .O(prob14), .o(prob14n), .P(prob15), .p(prob15n), .q(prob16n),
      .value(value2), .Out(data2)
      );

   comptParam MUXcounter(.A(count), .Rst(Rst), .clr(clrC), .en(enMUX), .C(SEL2));   //Counter for 'mux2' selector
   Comparador MUXcomp(.en(count), .Rst(Rst), .A(SEL2), .B(5'h1f), .EQ(EQ));         //Comparator to not exceed the
   Registrel #(.Bits(1)) rcomp(.en(ENc), .Rst(Rst), .clk(clk), .A(EQ), .Z(EQ2));
//--------------------
```

Figura 56

```
//Comparator that searches the higher value amongst the counted ones with the previous counters
   Comparador #(.Bits(32)) DataComp(.en(compare), .Rst(Rst), .A(data1), .B(data2), .GT(GT), .EQ(EQ1), .Out(data3), .C(value1), .D(value2), .Out2(value3));
   Registrel r1(.en(ENv), .Rst(Rst), .clk(clk), .A(value3), .Z(vOut));                    //Value register
   Registrel #(.Bits(32)) r2(.en(compare), .Rst(Rst), .clk(clk), .A(data3), .Z(prob));    //Probability register
//--------------------

//Stored data counter
   comptParam #(.Bits(6)) DataCount(.A(countValue), .Rst(Rst), .en(enMUX), .resta(resta), .C(numValue));

//Mask
   Huff_mask mask_module(.en(enMask), .Rst(Rst), .clr(clrMask), .dataIn(vOut), .dataOut(mask));
```

Figura 61

```
module Register32bit #(parameter Bits=32)(
    input Rst, clk, en, clr,
    input mode,    //Variable from the module 'estadistica', enables the entry of the 'prevReg'
    input [4:0] StoragePosition, StoragePositionIn,
    input [Bits-1:0] dataIn, data_prevReg, data_nextReg, dataReg2,    //dataReg2 is the one from the following of the following register
    output [Bits-1:0] dataOut
    );

    wire [Bits-1:0] dataToReg;
    wire [1:0] SEL;

    assign SEL =  mode ? 2'b01 : (StoragePositionIn==StoragePosition ? 2'b00 : (StoragePositionIn<StoragePosition ? 2'b10 : (StoragePositionIn>StoragePosition ? 2'b11 : 2'bzz)));

    MUX4 #(.Bits(Bits)) mux(.en(1'b1), .SEL(SEL), .Rst(Rst), .A(dataIn), .B(data_prevReg), .C(data_nextReg), .D(dataReg2), .Out(dataToReg));

    Registrel #(.Bits(Bits)) ProbR(.en(en), .Rst(Rst), .clk(clk), .clr(clr), .A(dataToReg), .Z(dataOut));

endmodule
```