DISEÑO DE UN MÓDULO RTL PARA LA ETAPA DE GENERACIÓN DE CLAVES DEL ALGORITMO CRIPTOGRÁFICO McELIECE EN FPGA

Damian Guillermo Morales Cruz

Trabajo de Grado para optar al título de Ingeniero Electrónico

Director
William Alexander Salamanca Becerra
Ingeniero Electrónico PhD

Codirector
Carlos Augusto Fajardo Ariza
Ingeniero Electrónico PhD

Universidad Industrial de Santander
Facultad de Ingenierías Fisicomecánicas
Escuela de Ingenieria Eléctrica, Electrónica y de Telecomunicaciones
Ingeniería Electrónica
Bucaramanga
2026

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Lista de Apéndices

Resumen

Titulo: DISEÑO DE UN MÓDULO RTL PARA LA ETAPA DE GENERACIÓN DE CLAVES DEL ALGORITMO CRIPTOGRÁFICO McELIECE EN FPGA [1]

Autor: Damian Guillermo Morales Cruz [2]

Palabras Clave: Algoritmo, McEliece, post-cuántico, diseño RTL, FPGA.

Descripción: La criptografía actualmente está presente en prácticamente todas las comunicaciones relacionadas con la seguridad y protección de la información. Existen sistemas criptográficos ampliamente utilizados como RSA, AES y ElGamal, considerados seguros en la actualidad; sin embargo, el avance de la computación cuántica ha generado preocupación sobre su seguridad futura, debido a la mayor capacidad de procesamiento de estas computadoras. Por ello, organizaciones como NIST, ISO, IETF y ETSI han impulsado la búsqueda y estandarización de algoritmos post-cuánticos. La iniciativa más relevante ha sido el concurso del NIST, que culminó con la estandarización de CRYSTALS-Kyber y la selección de otros algoritmos, entre ellos McEliece, foco principal de este proyecto.

Dado que el desarrollo de las computadoras cuánticas presenta un riesgo de seguridad para los computadores clásicos, el estudio de algoritmos post-cuánticos ha aumentado. El NIST creó un concurso para estandarizar estos algoritmos, con el objetivo de que se utilicen en el futuro para defender los datos contra los computadores cuánticos. Este proyecto tiene como objetivo diseñar y analizar el algoritmo McEliece en una FPGA, contribuyendo al estudio de estos algoritmos y proporcionando una plataforma para futuras investigaciones criptográficas.

[1] Trabajo de Grado

[2] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y teleco- municaciones. Director:William Alexander Salamanca Becerra, Ingeniero Electrónico PhD.

Abstract

Title: RTL MODULE DESIGN FOR THE KEY GENERATION STAGE OF THE McELIECE CRYPTOGRAPHIC ALGORITHM ON FPGA [1]

Author: Damian Guillermo Morales Cruz [2]

Keywords: Algorithm, McELIECE, post-quantum, RTL design, FPGA.

Description: Cryptography is currently present in virtually all communications related to information security and protection. There are several widely used cryptographic systems, such as RSA, AES, and ElGamal, which are considered secure today; however, the advancement of quantum computing has raised concerns about their future security due to the greater processing capabilities of quantum computers. Consequently, organizations such as NIST, ISO, IETF, and ETSI have promoted the search for and standardization of post-quantum algorithms. The most significant initiative has been the NIST competition, which resulted in the standardization of CRYSTALS-Kyber and the selection of other algorithms, including McEliece, which is the main focus of this project.

Since the development of quantum computers poses a security risk to classical computers, the study of post-quantum algorithms has increased. NIST created a competition to standardize these algorithms with the aim of using them in the future to protect data against quantum computers. This project aims to design and analyze the McEliece algorithm on an FPGA, contributing to the study of these algorithms and providing a platform for future cryptographic research.

[1] Undergraduate Thesis

[2] Faculty of Physical-Mechanical Engineering. School of Electrical, Electronic and Telecommunications Engineering. Advisor: William Alexander Salamanca Becerra, Electronic Engineer, PhD.

Introducción

La criptografía actualmente está presente en prácticamente todas las comunicaciones en lo que se refiere a la seguridad y protección de la información, hay varios sistemas criptográficos que son muy utilizados como el RSA, AES,ElGamal entre otros. Dichos sistemas y algoritmos actualmente son bastante seguros, aunque con la inminente llegada de la tecnología cuántica en las computadoras se ha generado preocupación e incertidumbre sobre la seguridad de la criptografía en un futuro cercano, ya que las computadoras cuánticas serían mucho más rápidas y potentes, con lo cual hace más fácil y eficiente romper la seguridad de los sistemas criptográficos.

Teniendo en cuenta lo anterior, varias organizaciones y organismos dedicados a la criptografía se han dado a la tarea de buscar estándares para algoritmos post cuánticos (resistentes a la computación cuántica). Algunos organismos son el ISO, ITU-T SC 17, IETF,ETSI, ENISA, CACR, entre otros; la iniciativa de estandarización más significativa está desarrollada por el NIST (National Institute of Standards and Technology) la cual consta de un concurso abierto en el cual se recibieron propuestas de algoritmos que podrían ser post cuánticos. Dicho concurso consta de 4 rondas, donde actualmente terminó con la estandarización del algoritmo CRYSTALS-kyber. Sin embargo, se obtuvieron 4 algoritmos próximos a estandarizar teniendo posteriores mejoras. Entre estos 4 finalistas se encuentra el algoritmo McEliece, en el cual se centra la implementación a realizar en este proyecto.

El algoritmo McEliece fue propuesto por R. J. McEliece en 1978, basado en la teoría de los códigos de Goppa y en los códigos de corrección de errores, el algoritmo pretende ocultar el mensaje añadiendo errores; dichos errores solo pueden ser corregidos conociendo la matriz generadora y las matrices de permutación empleadas en el proceso, así como ciertas características del código de Goppa utilizado. Todos los datos se proporcionan en la generación de las claves, esto hace que el algoritmo sea bastante seguro incluso ante ataques con el algoritmo de Shor.

Actualmente hay pocas implementaciones en FPGA (Field Programable Gate Array) ya que la mayoría se realizan en GPU (Graphics Processing Unit) para realizar pruebas de seguridad ante ataques, debido a esto y a que el algoritmo McEliece es un finalista en el proceso de estandarización de algoritmos de cifrado post-cuántico del NIST, es necesario estudiarlo y probarlo en las diferentes plataformas de cómputo disponibles en la actualidad. La finalidad de este proyecto

es justamente poder estudiar el comportamiento de McEliece en una FPGA para poder ayudar al entendimiento de este algoritmo y, a su vez, revisar que tan útil sería en una implementación de este estilo.

1. Objetivos

1.1. Objetivo General

Diseñar el módulo RTL para la generación de claves del criptosistema McEliece.

1.2. Objetivos Específicos

1. Realizar una revisión de los principios matemáticos del algoritmo McEliece y, con base en esta, desarrollar una primera implementación en software.
2. Proponer una arquitectura que establezca las formas de representación de los datos y los módulos requeridos para la generación de claves en el FPGA.
3. Desarrollar y verificar los módulos en RTL, validando su funcionamiento en simulación.
4. Evaluar la utilización de recursos y el desempeño de la generación de claves en el FPGA.

2. Marco Teórico

En esta sección se definirán conceptos que son fundamentales para comprender de manera más precisa el contenido del proyecto de investigación, facilitando así la interpretación de los métodos, resultados y discusiones que se presentan a lo largo del estudio.

2.1. Estructuras Matemáticas Fundamentales

Antes de estudiar códigos correctores de errores y criptografía, es necesario comprender algunas estructuras algebraicas básicas que constituyen el lenguaje matemático sobre el cual se construyen estos sistemas: los anillos y los campos.

2.1.1. Anillos

Un **anillo** es un conjunto A provisto de dos operaciones: la suma $(+)$ y la multiplicación $(\cdot)$, que satisfacen las siguientes propiedades:

1. **Cerradura:** Si se toman dos elementos del conjunto (A) y se suman o multiplican, el resultado también pertenece al mismo conjunto..

$$\forall a,b \in A: \quad a+b \in A, \quad a \cdot b \in A.$$

2. **Estructura aditiva abeliana:** La suma es asociativa y conmutativa, existe un elemento neutro aditivo 0 y todo elemento tiene inverso aditivo.

$$\forall a,b,c \in A, \quad (a+b)+c = a+(b+c) = b+(c+a),$$

$$\exists 0 \in A: \quad \forall a \in A,\ a+0 = a,$$

$$\forall a \in A,\ \exists (-a) \in A: \quad a+(-a) = 0.$$

3. **Asociatividad de la multiplicación:**

$$\forall a,b,c \in A, \quad (a \cdot b) \cdot c = a \cdot (b \cdot c) = b \cdot (a \cdot c).$$

4. **Distributividad de la multiplicación:** La multiplicación distribuye sobre la suma por ambos lados.

$$\forall a,b,c \in A, \quad a \cdot (b+c) = a \cdot b + a \cdot c, \quad (a+b) \cdot c = a \cdot c + b \cdot c.$$

Un anillo puede o no tener **elemento neutro multiplicativo** (denotado 1). Cuando existe, se dice que el anillo es *con unidad*. Si además la multiplicación es conmutativa, se habla de un **anillo conmutativo con unidad**. Ejemplos típicos son los enteros $\mathbb{Z}$ y el anillo de polinomios $K[x]$ sobre un cuerpo K.

2.1.2. Campos

Un **campo** es un anillo conmutativo con unidad $1 \neq 0$ en el que todo elemento distinto de cero tiene inverso multiplicativo. Es decir, en un campo se pueden realizar las cuatro operaciones aritméticas básicas (suma, resta, multiplicación y división por elementos distintos de cero).

Formalmente, un conjunto F es un campo si satisface las propiedades de anillo conmutativo con unidad y además:

$$\forall a \in F,\ a \neq 0 \Rightarrow \exists a^{-1} \in F : \quad a \cdot a^{-1} = 1.$$

Ejemplos de campos son $\mathbb{Q}$, $\mathbb{R}$ y $\mathbb{C}$. En aplicaciones discretas, los campos finitos son de especial interés.

2.2. Campos Finitos

Los **campos finitos** o **campos de Galois** son campos con un número finito de elementos. Se denotan por $GF(q)$ o $\mathbb{F}_q$, donde q es el número de elementos del campo. Una propiedad fundamental es que q debe ser una potencia de un primo:

$$q = p^n, \quad p \text{ primo}, \ n \in \mathbb{Z}_{>0}.$$

2.2.1. Campo primo $GF(p)$

Si $n = 1$ en $q = p^n$, entonces $q = p$ y se obtiene un **campo primo** $GF(p)$. Sus elementos son

$$\{0, 1, 2, \ldots, p-1\},$$

Las operaciones aritméticas básicas se realizan **módulo** p. Para todo $a, b \in GF(p)$:

$$\textit{Suma} : (a+b) \mod p = r, \qquad \text{Multiplicación: } (a \cdot b) \mod p = S,$$

$$\text{Inverso multiplicativo: } \forall a \in F,\ a \neq 0 \Rightarrow \exists a^{-1} \in F : \quad (a \cdot a^{-1}) \mod p \equiv 1.$$

Un ejemplo importante es $GF(2) = \{0, 1\}$, el campo binario, ampliamente usado en teoría de la información y criptografía.

2.2.2. Extensiones y campos $GF(p^n)$

Para $n > 1$ se construyen campos finitos de orden p^n como extensiones del campo primo $GF(p)$. Una construcción estándar es tomar el anillo de polinomios $GF(p)[x]$ generado por un polinomio irreducible $f(x)$ de grado n:

$$GF(p^n) \cong GF(p)[x]/\langle f(x) \rangle.$$

Los elementos de $GF(p^n)$ pueden representarse como de polinomios de grado menor que n con coeficientes en $GF(p)$. La multiplicación se realiza tomando el producto de polinomios y reduciéndolo módulo $f(x)$.

2.2.3. Propiedades relevantes

Los campos finitos poseen dos propiedades estructurales fundamentales: la ciclicidad de su grupo multiplicativo y la construcción de extensiones mediante polinomios irreducibles.

2.2.3.1. Grupo multiplicativo cíclico. Dado un campo finito $GF(q)$, se denota por $GF(q)^\times$ al conjunto de todos los elementos no nulos del campo:

$$GF(q)^\times = GF(q) \setminus \{0\}.$$

El cero se excluye porque carece de inverso multiplicativo. Este conjunto, que contiene $q-1$ elementos, forma un grupo abeliano bajo la multiplicación y tiene la propiedad de ser **cíclico**: existe al menos un elemento $\alpha \in GF(q)^\times$, llamado **elemento primitivo** o **generador**, cuyas potencias sucesivas generan todos los elementos del grupo:

$$GF(q)^\times = \langle \alpha \rangle = \{\alpha^0, \alpha^1, \ldots, \alpha^{q-2}\}.$$

El exponente máximo es $q-2$ porque el grupo tiene $q-1$ elementos y la secuencia comienza en $\alpha^0 = 1$.

2.2.3.2. Polinomios irreducibles y extensiones. La construcción de un campo de extensión $GF(p^n)$ a partir de un campo primo $GF(p)$ se realiza mediante un **polinomio irreducible** de grado n con coeficientes en $GF(p)$. Un polinomio irreducible es aquel que no puede descomponerse como producto de polinomios de grado menor en el mismo campo.

La extensión se define como el conjunto de polinomios de grado menor que n con coeficientes en $GF(p)$, donde las operaciones se efectúan módulo el polinomio irreducible elegido:

$$GF(p^n) \cong GF(p)[x]/\langle f(x) \rangle.$$

Cuando la raíz del polinomio irreducible genera además todo el grupo multiplicativo del campo extendido, el polinomio se denomina **polinomio primitivo**.

2.3. Campo finito $GF(2^m)$

Denominado *campo binario extendido*, puede verse como un espacio vectorial de dimensión m sobre $GF(2)$ en el cual existen m elementos $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ que forman una base de $GF(2^m)$ sobre $GF(2)$, de modo que cada elemento $c \in GF(2^m)$ se escribe de forma única como

combinación lineal:

$$c = \sum_{i=0}^{m-1} \alpha_i \beta_i,$$

donde $\alpha_i \in \{0,1\}$.

2.3.1. Representación en base polinomial

Sea $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ un polinomio irreducible de grado m sobre $GF(2)$, con $f_i \in \{0,1\}$. La construcción del campo de extensión $GF(2^m)$ se expresa formalmente como el anillo cociente:

$$GF(2^m) \cong GF(2)[x]/\langle f(x)\rangle.$$

donde:

- $GF(2)[x]$ denota el conjunto de todos los polinomios con coeficientes en el campo binario $\{0,1\}$.
- $\langle f(x)\rangle$ representa el ideal generado por el polinomio irreducible $f(x)$, es decir, el conjunto de todos los múltiplos polinomiales de $f(x)$.
- El cociente $GF(2)[x]/\langle f(x)\rangle$ agrupa los polinomios en **clases de equivalencia** módulo $f(x)$: dos polinomios pertenecen a la misma clase si su diferencia es un múltiplo de $f(x)$.

En términos prácticos, esta construcción significa que los elementos de $GF(2^m)$ se representan como polinomios de grado estrictamente menor que m:

$$a_{m-1}x^{m-1} + \cdots + a_1 x + a_0, \qquad a_i \in \{0,1\},$$

y todas las operaciones aritméticas se realizan tomando el resultado módulo $f(x)$. Usualmente estos elementos se escriben de forma compacta como cadenas de bits: $a_{m-1}a_{m-2}\ldots a_1 a_0$.

2.3.1.1. Operaciones en base polinomial.

- **Adición:** Se define como la operación lógica XOR componente a componente. Si $a =$

$(a_{m-1},\ldots,a_0)$ y $b=(b_{m-1},\ldots,b_0)$, entonces

$$a+b=(c_{m-1},\ldots,c_0),\quad c_i=a_i\oplus b_i.$$

• **Multiplicación:** Se define como el producto de polinomios seguido de una reducción módulo $f(x)$. Si $a=(a_{m-1},\ldots,a_0)$ y $b=(b_{m-1},\ldots,b_0)$ representan los polinomios $a(x)=\sum a_i x^i$ y $b(x)=\sum b_i x^i$, entonces

$$a\cdot b\equiv a(x)\,b(x)\pmod{f(x)}.$$

• **Inversión:** Para $a\neq 0$ se busca a^{-1} tal que $a\cdot a^{-1}\equiv 1 \pmod{f(x)}$. El inverso puede obtenerse mediante el algoritmo extendido de Euclides aplicado a $a(x)$ y $f(x)$.

2.3.1.2. Ejemplo: $GF(2^4)$ con $f(x)=x^4+x+1$. Este campo está compuesto por los 16 elementos listados en la tabla 1, correspondientes a todos los polinomios de grado menor que 4. Adicionalmente, se muestra entre paréntesis la representación binaria de cada elemento.

Tabla 1

Elementos del campo $GF(2^4)$ con $f(x)=x^4+x+1$

0 (0000)	1 (0001)	x (0010)	$x+1$ (0011)
x^2 (0100)	x^2+1 (0101)	x^2+x (0110)	x^2+x+1 (0111)
x^3 (1000)	x^3+1 (1001)	x^3+x (1010)	x^3+x+1 (1011)
x^3+x^2 (1100)	x^3+x^2+1 (1101)	x^3+x^2+x (1110)	x^3+x^2+x+1 (1111)

Operaciones de ejemplo:

Suma, operación lógica XOR:

$$(x^3+x^2+1)+(x^3+1)=(1101)\oplus(1001)=(0100).$$

Multiplicación:

$$(x^3+x^2+1)\cdot(x^3+1)=x^6+x^5+x^2+1\equiv x^3+x^2+x+1\pmod{x^4+x+1},$$

por tanto $(1101)\cdot(1001) = (1111)$.

Inverso:

$$(1101)^{-1} = (0100),$$

ya que $(1101)\cdot(0100) = (0001)$.

2.4. Códigos Correctores de Errores

En sistemas digitales, los datos pueden alterarse debido a ruido o interferencias. Para mitigar esto se emplean códigos correctores de errores, que añaden redundancia al mensaje original antes de transmitirlo. Esto permite detectar y, en muchos casos, corregir errores.

2.4.1. Conceptos básicos

- **Palabra de código:** Vector transmitido que incluye información y redundancia.

- **Distancia de Hamming:** Para dos palabras u, v de igual longitud, la distancia de Hamming $d(u,v)$ es el número de posiciones en las que difieren. La distancia mínima $d_{\text{mín}}$ de un código determina su capacidad de detección y corrección.

- **Capacidad de corrección:** Un código con distancia mínima $d_{\text{mín}}$ puede detectar hasta $d_{\text{mín}} - 1$ errores y corregir hasta $\left\lfloor \frac{d_{\text{mín}}-1}{2} \right\rfloor$ errores.

2.4.2. Familias relevantes

Entre los códigos más utilizados en criptografía y teoría de la información están los códigos de Reed–Solomon, códigos BCH y códigos de Goppa. Muchos de estos códigos se construyen usando la aritmética en campos finitos, especialmente en $GF(2^m)$ o en $GF(p^n)$.

2.5. Códigos de Goppa Binarios

2.5.1. Definición

Los **códigos de Goppa binarios** son códigos lineales definidos sobre $GF(2)$ y construidos a partir de un polinomio de Goppa $g(x)$ con coeficientes en $GF(2^m)$. Sea $g(x) \in GF(2^m)[x]$ de

grado t y sea $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq GF(2^m)$ tal que L no contiene raíces de $g(x)$. Para un vector $c = (c_1, \ldots, c_n)$ con $c_i \in GF(2)$ se define la función racional

$$R_c(x) = \sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} \in GF(2^m)(x).$$

El código de Goppa asociado se define como

$$\Gamma(L, g(x)) = \{\, c \in GF(2)^n \;:\; R_c(x) \equiv 0 \pmod{g(x)} \,\}.$$

Si se define $\frac{1}{x-\alpha_i} \equiv P_i(x) \pmod{g(x)}$ con

$$P_i(x) = P_{i,0} + P_{i,1}x + \cdots + P_{i,t-1}x^{t-1},$$

entonces la condición anterior equivale a

$$\sum_{i=1}^{n} c_i P_i(x) \equiv 0 \pmod{g(x)}.$$

Un código de Goppa binario *irreducible* es aquel cuyo polinomio $g(x)$ es irreducible en $GF(2^m)[x]$. Estos códigos admiten algoritmos de decodificación eficientes.

2.5.2. Parámetros

Los parámetros habituales de un código de Goppa son (n, k, d):

- n es la longitud del código (el tamaño de L).
- k es la dimensión del código; se tiene la cota $k \geq n - mt$.
- d es la distancia mínima; se cumple $d \geq t + 1$.

Aquí m es el grado de la extensión $GF(2^m)$ y $t = \deg g(x)$.

2.5.3. Matrices de paridad y generadora

La matriz de paridad H puede construirse a partir de los polinomios $P_i(x)$. Si se escribe cada $P_i(x) = \sum_{j=0}^{t-1} p_{j,i} x^j$ con $p_{j,i} \in GF(2^m)$, entonces

$$H = \begin{bmatrix} p_{0,1} & p_{0,2} & \cdots & p_{0,n} \\ p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{t-1,1} & p_{t-1,2} & \cdots & p_{t-1,n} \end{bmatrix}_{t \times n},$$

y debe cumplirse $H c^T = 0$ para todo $c \in \Gamma(L, g(x))$.

Una forma práctica de construir H es mediante la factorización en tres matrices T, V y D (definidas sobre $GF(2^m)$):

$$T = \begin{bmatrix} -g_t & -g_{t-1} & -g_{t-2} & \dots & -g_1 \\ 0 & -g_t & -g_{t-1} & \dots & -g_2 \\ 0 & 0 & -g_t & \dots & -g_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & -g_t \end{bmatrix}_{t \times t},$$

$$V = \begin{bmatrix} \alpha_1^{t-1} & \alpha_2^{t-1} & \dots & \alpha_n^{t-1} \\ \alpha_1^{t-2} & \alpha_2^{t-2} & \dots & \alpha_n^{t-2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}_{t \times n},$$

$$D = \begin{bmatrix} h_1 & 0 & 0 & \dots & 0 \\ 0 & h_2 & 0 & \dots & 0 \\ 0 & 0 & h_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & h_m \end{bmatrix}_{n \times n},$$

Donde:

g_i son los coeficientes del polinomio de Goppa $g(x)$ de grado t.

a_i los elementos del subconjunto L.

$h_i = g(a_i)^{-1}$.

Entonces una forma de H es:

$$H = T \cdot V \cdot D.$$

La matriz generadora del código G se obtiene como una base del espacio nulo de H, es decir, G satisface $GH^T = 0$ y tiene tamaño $k \times n$.

2.5.4. Codificación y decodificación

La codificación se realiza multiplicando un vector de información $u \in GF(2)^k$ por la matriz generadora G para obtener la palabra de código $c = uG$.

Para la decodificación se requiere un algoritmo eficiente. En códigos de Goppa binarios se utiliza habitualmente el **algoritmo de Patterson**, que permite corregir hasta t errores y se resume en los pasos siguientes.

2.5.4.1. Algoritmo de Patterson (esquema). Sea $y = c + e$ el vector recibido, siendo c la palabra de código y e el vector de errores. El objetivo es recuperar c, para esto se realizan los siguientes pasos.

1. Calcular el síndrome racional

$$R_y(x) = \sum_{i=1}^{n} \frac{y_i}{x-\alpha_i} \equiv \sum_{i=1}^{n} \frac{e_i}{x-\alpha_i} \pmod{g(x)},$$

ya que para $c \in \Gamma(L, g(x))$ se tiene $\sum_i \frac{c_i}{x-\alpha_i} \equiv 0 \pmod{g(x)}$.

2. Calcular

$$T(x) \equiv R_y(x)^{-1} \pmod{g(x)}.$$

3. Calcular

$$v(x) \equiv \sqrt{T(x)+x} \pmod{g(x)},$$

donde la raíz cuadrada se interpreta en $GF(2^m)[x]/\langle g(x)\rangle$ (en característica 2 la operación de elevar al cuadrado es lineal).

4. Resolver la ecuación de clave (key equation) mediante el algoritmo extendido de Euclides o factorización adecuada para obtener polinomios $A(x)$ y $B(x)$ tales que

$$\sigma(x) = A(x)^2 + xB(x)^2$$

es el polinomio localizador de errores. Se tiene las cotas

$$\deg A(x) \leq \left\lfloor \frac{t}{2} \right\rfloor, \qquad \deg B(x) \leq \left\lfloor \frac{t-1}{2} \right\rfloor,$$

y además $A(x) = B(x)\, v(x)$ en el anillo cociente.

5. Encontrar las raíces de $\sigma(x)$. Si $\sigma(\alpha_i) = 0$ entonces hay un error en la posición i.

6. Construir el conjunto de posiciones de error $E = \{\, i \mid \sigma(\alpha_i) = 0 \,\}$ y el vector de errores e con $e_i = 1$ si $i \in E$, $e_i = 0$ en caso contrario.

7. Recuperar la palabra de código $c = y - e$.

El algoritmo de Patterson es eficiente y está especialmente adaptado a la estructura de los códigos de Goppa binarios; su complejidad práctica depende de t y de las operaciones en $GF(2^m)$.

2.6. Descripción del Criptosistema de McEliece

El criptosistema de McEliece es un esquema de criptografía asimétrica, basado en códigos correctores de errores, propuesto por Robert McEliece en 1978. El principio básico es seleccionar un código corrector de errores con un decodificador eficiente y transformar su representación para ocultar su estructura, publicando la versión transformada como clave pública.

Sea $m,t \in \mathbb{N}$. Sea $g(x)$ un polinomio irreducible de grado t sobre $GF(2^m)$ y sea $L \subset GF(2^m)$ con $|L| = n$ tal que L no contiene raíces de $g(x)$. Consideramos el código de Goppa binario irreducible $\Gamma(L, g(x))$ de longitud n, dimensión $k \geq n - mt$ y capacidad de corregir hasta t errores.

A continuación se presenta un ejemplo en el que Alice y Bob establecen una comunicación mediante el criptosistema, asumiendo Alice el rol de generadora de las claves.

2.6.1. Generación de claves

Alice realiza los siguientes pasos:

1. Selecciona un código de Goppa binario $\Gamma(L, g(x))$ capaz de corregir t errores y obtiene su matriz generadora privada G de tamaño $k \times n$.
2. Genera una matriz invertible aleatoria S de tamaño $k \times k$.
3. Genera una matriz de permutación P de tamaño $n \times n$.
4. Calcula la matriz pública $\widehat{G} = SGP$, de tamaño $k \times n$.
5. Publica la clave pública $(\widehat{G}, t)$ y guarda como clave privada el conjunto (S, P, G, A), donde A es el algoritmo de decodificación eficiente (por ejemplo, el algoritmo de Patterson).

Algunos pasos de la etapa de generación de claves pueden cambiar dependiendo del esquema que se utilice, como el classic McEliece que usa la matriz generadora G, o el Niederreiter que usa la matriz H para generar las claves.

2.6.2. Cifrado

Para enviar a Alice un mensaje $m \in GF(2)^k$ (vector de longitud k), Bob realiza:

1. Calcula $c' = m\widehat{G}$.

2. Genera un vector de error $e \in GF(2)^n$ de peso $\leq t$, el cual puede ser aleatorio o determinista, según el criterio de Bob.

3. Calcula el texto cifrado $c = c' + e$ y lo envía a Alice.

2.6.3. Descifrado

Al recibir c, Alice procede:

1. Aplica la permutación inversa: $\widetilde{c} = cP^{-1}$.

2. Usa el decodificador privado A sobre $\widetilde{c}$ para corregir los errores y recuperar $\widetilde{m}$ tal que $\widetilde{m}G = \widetilde{c} - \widetilde{e}$.

3. Recupera el mensaje original aplicando la inversa de S:

$$m = \widetilde{m}S^{-1}.$$

La seguridad del esquema se basa en la dificultad de decodificar un código lineal general sin conocer la estructura oculta (problema de decodificación de códigos lineales), y en que $\widehat{G}$ debe parecer una matriz generadora aleatoria para un atacante.

2.7. Resumen

En este capítulo se han presentado las nociones básicas de anillo y campo, la teoría de campos finitos y la construcción explícita de $GF(2^m)$ en base polinomial, junto con ejemplos operativos. Se introdujeron los conceptos fundamentales de códigos correctores de errores y se desarrolló la teoría y práctica de los códigos de Goppa binarios, incluyendo la construcción de matrices de paridad y generadora y el algoritmo de Patterson para decodificación. Finalmente se describió el criptosistema de McEliece, su generación de claves, cifrado y descifrado, y la relación entre la teoría de campos finitos y la seguridad del esquema.

3. Metodología

3.1. Enfoque metodológico

El desarrollo del presente trabajo sigue un enfoque metodológico basado en una transición progresiva desde el modelo matemático del criptosistema de McEliece hacia su implementación en hardware a nivel *Register Transfer Level* (RTL). Como etapa intermedia, se realiza una primera implementación en software, la cual permite validar el funcionamiento del sistema y esclarecer las operaciones fundamentales necesarias para su posterior descripción en hardware.

3.2. Implementación en software

Para garantizar la correcta comprensión del sistema y validar el comportamiento de los algoritmos involucrados, se realiza inicialmente una simulación funcional utilizando un software libre especializado en matemáticas llamado *SageMath*. Este entorno permite trabajar de forma nativa con estructuras algebraicas como campos finitos, anillos de polinomios y códigos correctores de errores.

El desarrollo se divide en las siguientes etapas:

1. **Definición del código de Goppa**

    a) Definición de los parámetros del campo finito.

    b) Construcción del campo $GF(2^m)$.

    c) Definición del polinomio de Goppa.

2. **Generación de claves del criptosistema**

    a) Generación del conjunto soporte L.

    b) Construcción de la matriz de paridad H.

    c) Obtención de la matriz generadora G.

3. **Definición del proceso criptográfico**

    a) Definición del algoritmo de decodificación.

b) Generación de las matrices auxiliares del criptosistema.

c) Codificación y decodificación.

Si bien el diseño RTL comprende únicamente la etapa de generación de claves, la implementación en software abarca el criptosistema completo. A continuación se describe cada una de dichas etapas.

3.2.1. Definición de los parámetros del sistema

El primer paso consistió en seleccionar los parámetros fundamentales del código de Goppa y del campo finito sobre el cual se construyó el criptosistema.

El parámetro principal es m, el cual corresponde al grado de extensión del campo finito. A partir de este valor se definió el campo $GF(2^m)$, el cual contiene n elementos, donde $n = 2^m$; a continuación, se estableció el grado del polinomio de Goppa: $t = m - 1$, donde t representa la cantidad máxima de errores que pueden ser corregidos en una palabra de código.

3.2.2. Construcción del campo finito

El punto de partida para la construcción del campo finito es definir el anillo de polinomios y sobre este construir el campo de extensión; a continuación se muestra un ejemplo numérico que se utilizó en la simulación en SageMath, se tomó $m = 4$ y a partir de esto se definió el anillo de polinomios sobre $GF(2)$:

$$GF(2)[x]$$

Posteriormente se construyó el campo de extensión

$$GF(2^4) = GF(2)[x]/f(x)$$

donde $f(x)$ es un polinomio al cual no se puede factorizar en polinomios más simples, denominado polinomio irreducible, de grado m, que se utiliza como base del campo. En el entorno SageMath este polinomio se selecciona automáticamente al crear el campo finito, en este caso es $x^4 + x + 1$, lo que permite definir directamente los elementos del campo como polinomios módulo

$f(x)$, este campo contiene todos los polinomios de grado menor que 4, como se puede ver en la tabla 1.

3.2.3. Definición del polinomio de Goppa

Una vez definido el campo $GF(2^4)$, se procedió a definir el polinomio de Goppa

$$g(x) \in GF(2^4)[x]$$

Este polinomio debe ser irreducible y tener grado t, para este ejemplo $t = 4 - 1 = 3$. Teniendo esto en cuenta se seleccionó el polinomio $x^3 + x + 1$, el cual determina la estructura del código corrector de errores y su capacidad de corrección.

3.2.4. Generación del conjunto soporte L

El siguiente paso consistió en definir el conjunto soporte del código:

$$L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$$

donde

$$\alpha_i \in GF(2^m)$$

y se cumple que

$$g(\alpha_i) \neq 0$$

La longitud del código está dada por

$$n = 2^m = 2^4 = 16$$

Por lo tanto, el conjunto soporte contiene todos los elementos del campo que no son raíces del polinomio de Goppa:

$$L = [1, x, x^2, x^3, x+1, x^2+x, x^3+x^2, x^3+x+1,$$
$$x^2+1, x^3+x, x^2+x+1, x^3+x^2+x,$$
$$x^3+x^2+x+1, x^3+x^2+1, x^3+1, 1]$$

3.2.5. Construcción de la matriz de paridad H

Con el polinomio de Goppa y el conjunto soporte definidos, es posible construir la matriz de paridad del código.

Para ello se utilizan tres matrices auxiliares:

$$T, V, D$$

3.2.5.1. Matriz T. Es una matriz triangular de tamaño $t \times t$, en este caso 3×3, construida a partir de los coeficientes del polinomio de Goppa

$$g(x) = x^t + g_{t-1}x^{t-1} + \cdots + g_1 x + g_0$$

Los elementos de la matriz se definen de la siguiente manera:

Si $i = j$:

$$T[i, j] = 1$$

Si $i < j$:

$$T[i, j] = 0$$

Si $i > j$:

$$T[i, j] = g_{t-(i-j)}$$

Para nuestro ejemplo la matriz T quedó definida como:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}_{3\times 3},$$

3.2.5.2. Matriz V. Es una matriz de tipo Vandermonde de tamaño $t \times n$, en este caso 3×16 construida a partir de los elementos del conjunto soporte L. Cada elemento se define como:

$$V[i, j] = (L_j)^i$$

donde L_j es el elemento j-ésimo del conjunto soporte.

Para nuestro ejemplo la matriz V quedó definida como:

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & x & x^2 & x^3 & x+1 & x^2+x & x^3+x^2 & x^3+x+1 \\ 1 & x^2 & x+1 & x^3+x^2 & x^2+1 & x^2+x+1 & x^3+x^2+x+1 & x^3+1 \end{bmatrix}$$

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ x^2+1 & x^3+x & x^2+x+1 & x^3+x^2+x & x^3+x^2+x+1 & x^3+x^2+1 & x^3+1 & 1 \\ x & x^3 & x^2+x & x^3+x+1 & x^3+x & x^3+x^2+x & x^3+x^2+1 & 1 \end{matrix}\Bigg]_{3\times 16}$$

3.2.5.3. Matriz D. Es una matriz diagonal de tamaño $n \times n$, en este caso 16×16 cuyos elementos se definen como:

$$D[i, i] = \frac{1}{g(L_i)}$$

donde $g(L_i)$ corresponde a la evaluación del polinomio de Goppa en el elemento L_i del conjunto soporte.

Para nuestro ejemplo la matriz D quedó definida como:

$$D = diag[1, x^2+1, x, x^3+x^2+x, x^2, x^2+x+1, x^3+x+1, x^2+x+1,$$
$$x+1, x^3+x^2+1, x^2+x, x^2+x, x^3+1, x^2+x+1, x^2+x, 1]$$

3.2.5.4. Matriz de paridad. Una vez definidas las matrices T, V y D, la matriz de paridad del código se obtiene mediante la multiplicación

$$H = T \cdot V \cdot D$$

la matriz resultante tiene dimensiones $t \times n$, en este caso 3×16 y sus elementos pertenecen al campo $GF(2^4)$.

$$H = \begin{bmatrix} 1 & x^2+1 & x & x^3+x^2+x & x^2 & x^2+x+1 & x^3+x+1 & x^2+x+1 \\ 1 & x^3+x & x^3 & x^3+1 & x^3+x^2 & 1 & x^3+x^2+1 & x^2 \\ 0 & x & x^2 & x^3+x & x+1 & 1 & x^3 & x^3+x^2+1 \end{bmatrix}$$

$$\begin{bmatrix} x+1 & x^3+x^2+1 & x^2+x & x^2+x & x^3+1 & x^2+x+1 & x^2+x & 1 \\ x^3+x^2+x+1 & x^3+x+1 & 1 & x & x^3+x^2+x & x^2+1 & x+1 & 1 \\ x^2+1 & x^3+x^2+x+1 & 1 & x^3+1 & x^3+x^2 & x^3+x+1 & x^3+x^2+x & 0 \end{bmatrix}_{3\times 16}$$

Para seguir con el procedimiento, esta matriz se transformó en su representación binaria, donde cada elemento del campo se expresa como un vector de 4 bits, cada vector está representado en forma vertical en la matriz resultante:

$$H_b = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}_{12\times 16}$$

3.2.6. Obtención de la matriz generadora G

A partir de la matriz de paridad en su forma binaria H_b, se calcula la matriz generadora del código G, la cual satisface la relación $GH^T = 0$. En SageMath la mejor forma de resolver esta relación es obteniendo el kernel o núcleo de la matriz H y generando una matriz base con dicho kernel, obteniendo así la matriz G, de dimensión $k \times n$, en este caso 4×16.

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}_{4\times 16}$$

3.2.7. Selección del algoritmo de decodificación

Para el proceso de decodificación se seleccionó el algoritmo de Patterson, el cual es ampliamente utilizado para la decodificación de códigos de Goppa binarios debido a su eficiencia y adecuación a la estructura algebraica de estos códigos. Este algoritmo permite localizar y corregir errores a partir del cálculo de síndromes y la resolución de ecuaciones polinomiales en el campo finito $GF(2^m)$.

La elección de este algoritmo se fundamenta en que presenta una complejidad computacional moderada y una formulación basada en operaciones sobre polinomios, tales como inversión modular y factorización. Además, el algoritmo de Patterson garantiza la corrección de hasta t errores, en concordancia con la capacidad de corrección del código de Goppa definido previamente.

3.2.8. Generación de las matrices auxiliares del criptosistema.

Para ocultar la estructura del código de Goppa se generan dos matrices adicionales.

3.2.8.1. Matriz *S*. Es una matriz aleatoria invertible de tamaño $k \times k$, en este caso 4×4 definida sobre $GF(2)$:

$$S = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}_{4\times 4}$$

3.2.8.2. Matriz *P*. Es una matriz de permutación de tamaño $n \times n$ en este caso 16×16 definida sobre $GF(2)$:

$$P = \begin{bmatrix}
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0
\end{bmatrix}_{16\times 16}$$

3.2.9. Codificación y decodificación

La clave pública se define como $G' = S \cdot G \cdot P$:

$$G' = \begin{pmatrix}
1&0&0&1&0&1&0&1&1&1&0&1&0&0&1&1\\
0&1&1&0&1&0&1&0&0&0&1&0&1&1&0&0\\
0&1&1&1&1&1&1&0&1&1&1&1&1&1&1&1\\
0&1&0&0&1&0&0&1&0&1&1&0&0&1&1&1
\end{pmatrix}_{4\times 16}$$

mientras que la clave privada está formada por (S,G,P,A) donde A es el algoritmo de

decodificación para los códigos de Goppa.

Finalmente, se verificó el proceso de codificación y decodificación.

Dado un mensaje $u \in GF(2)^k$:

$$u = [1,0,0,0]$$

la palabra de código se obtiene como $c = uG'$:

$$c = [1,1,0,0,1,0,0,1,1,1,1,1,1,0,0,0]$$

Posteriormente se introduce un vector de errores $e \in GF(2)^n$:

$$e = [0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0]$$

y se obtiene el mensaje transmitido $y = c + e$:

$$y = [1,1,0,0,1,0,0,1,0,0,1,1,1,0,0,0]$$

El proceso de decodificación permite identificar las posiciones de error (8,9) y recuperar el mensaje original (1,0,0,0).

3.3. Diseño RTL

Una vez validada una implementación funcional en software, se procedió a realizar un análisis más detallado con el objetivo de identificar las operaciones fundamentales involucradas y establecer una arquitectura hardware adecuada. Este análisis permite traducir las operaciones algebraicas del dominio teórico hacia bloques funcionales que puedan ser implementados a nivel RTL.

A partir de este estudio, se identificaron y clasificaron los módulos necesarios para el desarrollo de cada etapa, los cuales corresponden a operaciones bien definidas sobre cuerpos finitos y estructuras matriciales.

1. **Generación de claves:** Esta etapa concentra la mayor complejidad computacional, ya

que involucra tanto operaciones aritméticas en cuerpos finitos como transformaciones matriciales.

a) Operaciones en $\mathbb{F}_{2^m}$: suma, resta, división, multiplicación y potenciación.

b) Construcción de matrices estructurales: T, V, D y H

c) Inversión y reducción de matrices mediante el algoritmo de Gauss-Jordan

2. **Codificación:** Consiste en la transformación del mensaje mediante la clave pública.

a) Aplicación de la matriz de permutación P

b) Multiplicación por la matriz secreta S

3. **Decodificación:** Se basa en la recuperación del mensaje original a partir del texto cifrado.

a) Implementación del algoritmo de Patterson para decodificación de códigos Goppa

Con el fin de facilitar la implementación hardware, se adoptó una representación binaria de los polinomios en $\mathbb{F}_{2^m}$, donde cada polinomio se modela como una cadena de bits, en la cual cada bit corresponde a un coeficiente. Esta representación resulta especialmente conveniente para arquitecturas digitales, ya que permite mapear directamente las operaciones algebraicas a lógica combinacional y secuencial.

Definida esta base, se procede a la selección de parámetros del sistema. En el criptosistema de McEliece, los parámetros fundamentales son m, n, t y k, los cuales determinan tanto el nivel de seguridad como el costo computacional y de almacenamiento.

Tabla 2

Parámetros del sistema

m	$n = 2^m$	$t = m - 1$	$k = n - m * t$
32	4.294.967.296	31	4.294.966.304
16	65.536	15	65.296
8	256	7	200
4	16	3	4

A partir de los parámetros de la tabla 2, se estimó el tamaño de las matrices involucradas en el sistema, expresado en bits. Este análisis es crucial, ya que el tamaño de dichas estructuras impacta directamente en los requerimientos de memoria y en la viabilidad de implementación sobre hardware reconfigurable.

Tabla 3

Tamaño de matrices en bits

m	T_{txt}	V_{txn}	D_{nxn}	G_{kxn}	P_{nxn}	S_{kxk}
32	961	$4{,}26 \times 10^{12}$	$5{,}9 \times 10^{20}$	$1{,}84 \times 10^{19}$	$1{,}89 \times 10^{19}$	$1{,}84 \times 10^{19}$
16	225	$1{,}57 \times 10^{7}$	$6{,}87 \times 10^{10}$	$4{,}28 \times 10^{9}$	$4{,}29 \times 10^{9}$	$4{,}26 \times 10^{9}$
8	49	$1{,}43 \times 10^{4}$	$5{,}24 \times 10^{5}$	$5{,}12 \times 10^{4}$	$6{,}55 \times 10^{4}$	4×10^{4}
4	9	192	$1{,}02 \times 10^{3}$	64	256	16

Como se observa en la tabla 3, el crecimiento exponencial de los parámetros con respecto a m genera requerimientos de memoria extremadamente elevados para valores grandes, lo cual dificulta su implementación directa en plataformas FPGA de recursos limitados. Considerando las restricciones de hardware disponibles —particularmente la capacidad de memoria de las FPGA del laboratorio— se decidió trabajar inicialmente con $m = 4$ como caso de prueba, permitiendo validar la arquitectura y depurar el diseño.

Posteriormente, se plantea utilizar $m = 8$ como configuración objetivo, ya que representa un compromiso adecuado entre complejidad, consumo de recursos y escalabilidad del sistema. Es importante destacar que, aunque estos valores no corresponden a parámetros criptográficamente seguros, el diseño fue concebido de manera parametrizable, de modo que pueda escalarse a configuraciones más robustas en plataformas con mayores capacidades.

Esta decisión establece una transición natural hacia el diseño RTL, en el cual se prioriza la modularidad, reutilización de bloques aritméticos y optimización del uso de memoria, aspectos clave para una implementación eficiente del criptosistema en hardware.

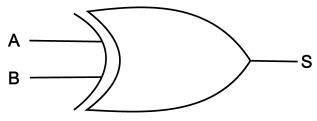
3.3.1. Operaciones en $\mathbb{F}_{2^m}$

3.3.1.1. Suma y resta. Al considerar los polinomios en su representación binaria sobre $\mathbb{F}_2$, las operaciones de suma y resta son equivalentes, pudiendo implementarse directamente me-

diante una compuerta XOR:

Figura 1

Compuerta XOR



A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

3.3.1.2. División. Se diseñó primero la división ya que para las demás operaciones es necesario realizar la reducción polinomial en la base del campo. El módulo `division` implementa un algoritmo de división binaria a nivel hardware, basado en el método clásico de división larga, adaptado para operar sobre representaciones polinomiales en forma binaria.

Los parámetros del módulo son `WN`, `WD` y `WCR`, longitudes del numerador, denominador, cociente y residuo, respectivamente. Con las entradas `N`,`D`, y salidas `Residuo` y `Cociente`. El principio de funcionamiento consiste en realizar una serie de restas sucesivas (implementadas como operaciones XOR) entre el dividendo y el divisor, acompañadas de desplazamientos hacia la izquierda. Inicialmente, el dividendo *N* es extendido mediante ceros en sus bits menos significativos, formando el vector `NN`, lo que permite alinear correctamente el proceso de división, se toma una ventana de bits del residuo parcial (`NNN`) y se compara con el resultado de aplicar una operación XOR con el divisor *D*. Dependiendo de esta comparación, el algoritmo decide si se realiza la resta (XOR) o si se conserva el valor actual. Este proceso es equivalente a determinar si el divisor "cabe" en el residuo parcial en cada paso de la división.

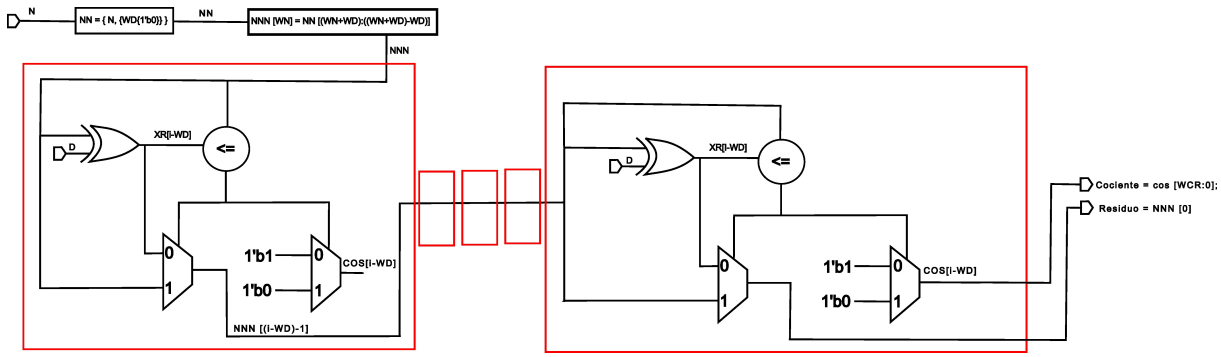
Simultáneamente, se construye el cociente bit a bit en el vector `cos`, donde cada bit indica si en la iteración correspondiente se efectuó la resta. El residuo se actualiza en cada etapa mediante un desplazamiento e incorporación del siguiente bit del dividendo extendido.

Al finalizar el proceso iterativo, el módulo entrega como salidas el cociente (`Cociente`), formado por los bits más significativos calculados, y el residuo final (`Residuo`), correspondiente al valor restante tras la última iteración.

El diagrama presentado en la figura 2 es funcionalmente equivalente a la descripción en Verilog, los cuadros resaltados en rojo hacen referencia a una etapa del proceso, donde el contenido

Figura 2

División



en cada uno es exactamente igual. No obstante, es importante destacar que el diseño resultante es completamente combinacional, donde todas las etapas se encuentran desplegadas en paralelo.

3.3.1.3. Multiplicación. El módulo `Multiplicacion` implementa la multiplicación de dos polinomios en el campo finito $\mathbb{F}_{2^m}$, seguida de una reducción modular respecto a un polinomio base irreducible.

Los parámetros del módulo son `W`, `WBAS` y `WCR`, longitud de los elementos a multiplicar, longitud de la base polinomial, longitud del resultado, respectivamente. Con las entradas `A`,`B`, como los factores, `BAS` como la base del campo usada para la reducción modular, por último la salida `Multiplicacion` como el resultado final:

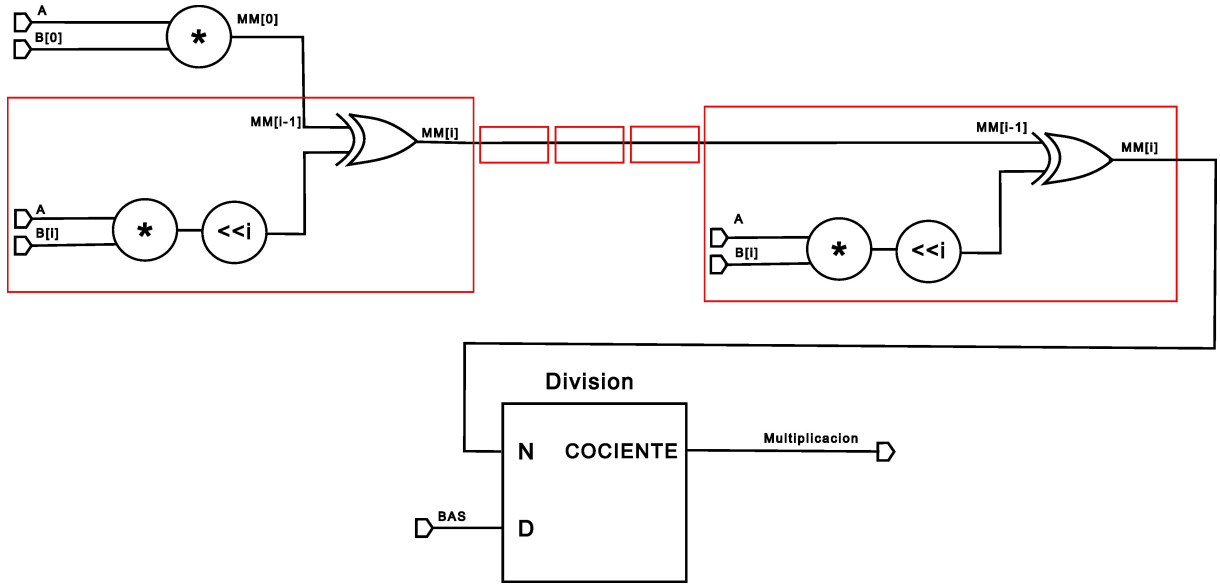
En la primera etapa, se realiza la multiplicación de los operandos `A` y `B` utilizando una estructura iterativa basada en productos parciales. Cada bit de `B` actúa como selector del operando `A`, generando un término parcial desplazado. Estos términos se acumulan mediante operaciones XOR, lo cual corresponde a la suma en $\mathbb{F}_2$. El resultado de esta etapa es un polinomio intermedio de grado mayor, almacenado en `MM[W]`.

En la segunda etapa, se lleva a cabo la reducción modular del polinomio resultante utilizando el polinomio base `BAS`, que define el campo finito. Este proceso consiste en eliminar los términos de mayor grado mediante operaciones XOR condicionadas, equivalentes a restar múltiplos del polinomio base. La reducción se implementa con el módulo de `division`; finalmente, el resultado reducido se obtiene seleccionando los bits menos significativos del último valor inter-

medio, correspondientes al tamaño del campo, y se asigna a la salida `Multiplicacion`.

Figura 3

Multiplicación



El diagrama presentado en la figura 3 también es totalmente combinacional.

3.3.1.4. Potenciación. El módulo `Pot` implementa la operación de potenciación en el campo finito $\mathbb{F}_{2^m}$ utilizando el método de exponenciación binaria (square-and-multiply).

Los parámetros del módulo son `W`, `WBAS` y `WCR`, longitud de los elementos a multiplicar, longitud de la base polinomial, longitud del resultado, respectivamente. Con la entrada `A` como el elemento a operar, `BAS` como la base del campo usada para la reducción modular, `EXP` como el exponente, por último la salida `Potenciacion` como el resultado final:

El diseño se estructura en tres etapas principales. En la primera etapa, se generan potencias sucesivas del operando `A` mediante elevaciones al cuadrado iterativas. Esto se realiza utilizando instancias del módulo `Multi`, donde cada resultado se obtiene como el cuadrado del anterior, produciendo los términos A^{2^i}. Estos valores se almacenan en el arreglo `P`, representando las potencias base necesarias para la descomposición binaria del exponente.

En la segunda etapa, se realiza una selección condicional de dichas potencias en función del valor del exponente `EXP`. Para cada bit del exponente, si este es igual a 1, se selecciona la

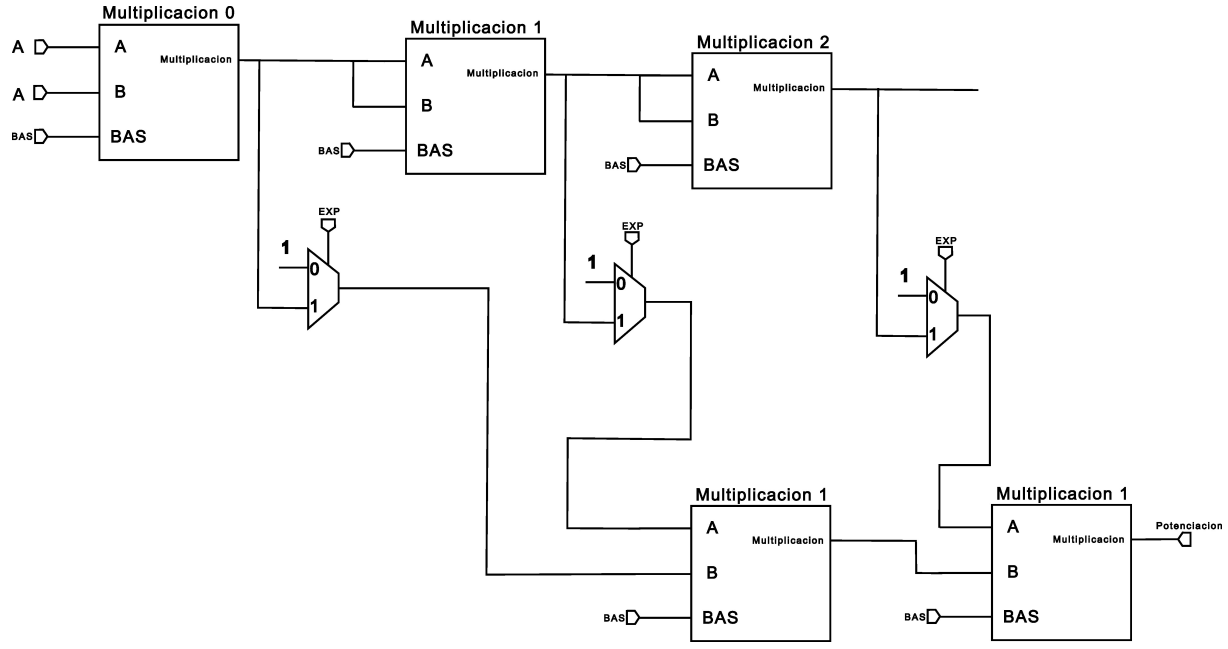
potencia correspondiente; en caso contrario, se selecciona el elemento neutro multiplicativo (valor 1). Este proceso genera el conjunto de operandos parciales almacenados en `P2`.

Finalmente, en la tercera etapa, se lleva a cabo la acumulación de los términos seleccionados mediante una cadena de multiplicaciones sucesivas, nuevamente utilizando el módulo `Multiplicacion`. Este proceso implementa la multiplicación de las potencias activadas, reconstruyendo así el resultado de la exponenciación.

El resultado final corresponde a la salida `Potenciacion`, que representa A^{EXP} en el campo finito definido por el polinomio base `BAS`.

Figura 4

Potenciación



El diagrama presentado en la figura 4 también es totalmente combinacional.

3.3.1.5. Consideraciones de diseño y alternativas. El diseño de los módulos presentados se realizó priorizando una arquitectura completamente combinacional, con el objetivo de minimizar la latencia de cada operación. No obstante, durante el desarrollo se consideraron diversas alternativas que, aunque no fueron implementadas, resultan relevantes desde el punto de vista de optimización, escalabilidad y flexibilidad del sistema.

En primer lugar, se evaluó la posibilidad de implementar arquitecturas secuenciales para operaciones como la división, multiplicación y potenciación. Este enfoque permitiría reducir significativamente el uso de recursos hardware mediante la reutilización de unidades funcionales, a costa de incrementar el número de ciclos de reloj requeridos para obtener un resultado. De manera complementaria, para la operación de potenciación se analizaron esquemas más avanzados de exponenciación, como el uso de ventanas deslizantes o técnicas de precomputación, los cuales disminuyen el número de multiplicaciones necesarias, pero introducen un mayor costo en términos de almacenamiento intermedio y complejidad en la lógica de control.

Adicionalmente, una vez definidos los módulos de operaciones básicas, se exploró como alternativa el diseño de una arquitectura basada en RISC-V orientada al procesamiento de operaciones en campos finitos. Esta propuesta contemplaba el desarrollo de una unidad aritmético-lógica (ALU) especializada, junto con la definición de un conjunto de instrucciones dedicado, permitiendo implementar el flujo criptográfico completo a partir de la ejecución de dichas instrucciones, inicialmente en lenguaje ensamblador y posteriormente en lenguaje C. Este enfoque ofrecía una mayor flexibilidad y programabilidad del sistema, a costa de un incremento significativo en la complejidad del diseño a nivel de arquitectura y control.

No obstante, dichas alternativas fueron descartadas con el fin de mantener el enfoque del proyecto en la integración de módulos específicos para cada etapa del criptosistema. En este sentido, se optó por una implementación completamente desplegada, que favorece el paralelismo, reduce la latencia y simplifica la verificación funcional. Asimismo, se prestó especial atención a la parametrización de los módulos, con el propósito de facilitar su reutilización y adaptación a distintos tamaños de campo y configuraciones. Estas decisiones permiten contextualizar el diseño propuesto y abren la posibilidad de futuras optimizaciones orientadas a diferentes escenarios de aplicación.

3.3.2. Construcción de la matriz de paridad H

Una vez establecidos los módulos correspondientes a las operaciones básicas en $\mathbb{F}_{2^m}$, se cuenta con los bloques fundamentales necesarios para abordar etapas de mayor nivel dentro del criptosistema. En particular, se procede al diseño de los módulos encargados de la generación de la matriz de paridad.

A partir de este punto, el enfoque del diseño se orienta a la optimización del uso de recursos hardware, buscando minimizar el consumo de área sin comprometer la funcionalidad. En este contexto, se evita el almacenamiento explícito de las matrices auxiliares involucradas en la construcción de la matriz H. En su lugar, dichos componentes se conciben como módulos capaces de calcular dinámicamente un único elemento de la matriz en cada instante, a partir de los índices i y j correspondientes.

Este enfoque permite reducir significativamente los requerimientos de memoria, a costa de un incremento en la lógica combinacional asociada al cálculo de cada posición, manteniendo así la coherencia con la estrategia general de diseño adoptada en la etapa anterior.

3.3.2.1. Matriz T. El módulo `Matrix_T` implementa la generación de un elemento de la matriz de transformación T a partir de la comparación entre dos índices y un vector de coeficientes del polinomio de Goppa utilizado para generar los códigos de Goppa.

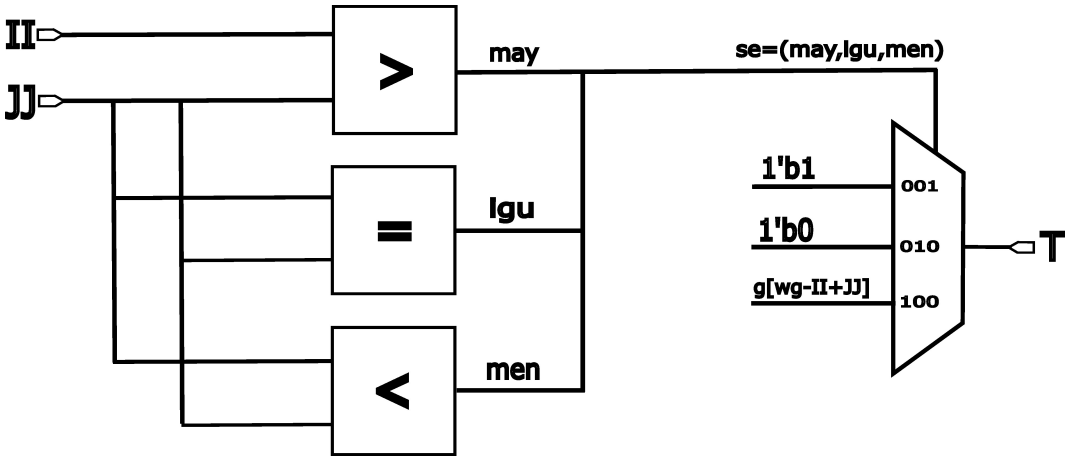
Los parámetros del módulo son `WN` y `Wg`, que definen la longitud de los índices y del vector de coeficientes, respectivamente. Como entradas, el módulo recibe `g`, que corresponde a un vector de coeficientes, y los valores `II` y `JJ`, que representan los índices de fila y columna de la matriz. La salida `T` corresponde al valor calculado para la posición $T(i, j)$.

El principio de funcionamiento del módulo se basa en la comparación entre los índices `II` y `JJ`. Para ello, se generan tres señales booleanas que indican si `II` es mayor, igual o menor que `JJ`. Estas señales se agrupan en el vector `se`, el cual permite seleccionar el comportamiento del módulo.

Cuando `II <JJ`, el valor de salida se define como cero, lo que implica que la matriz es estrictamente triangular en esa región. En el caso `II == JJ`, la salida toma el valor uno, estableciendo la diagonal principal de la matriz. Finalmente, cuando `II >JJ`, el valor de salida se obtiene a partir del vector `g`, indexando la posición correspondiente mediante la expresión `g[Wg - II + JJ]`. Esta operación corresponde a tomar el coeficiente del polinomio de Goppa respectivo para la posición i y j.

Figura 5

Matriz T



El diseño del módulo mostrado en la figura 5 es completamente combinacional, ya que la salida depende únicamente de las entradas actuales sin requerir elementos de memoria. Adicionalmente, se define un valor por defecto para la salida, con el fin de evitar estados indeterminados durante la evaluación lógica.

3.3.2.2. Matriz V. El módulo `Matrix_V` implementa el cálculo de un elemento de una matriz auxiliar V definida en función de una operación de potenciación en el campo finito $\mathbb{F}_{2^m}$.

Los parámetros del módulo son `WN`, `WBAS` y `WCR`, que corresponden a la longitud de los índices de entrada, la longitud del polinomio base del campo y la longitud del resultado, respectivamente. Como entradas, el módulo recibe `II` y `JJ`, que representan los índices de la posición (i, j) dentro de la matriz. La salida `V` corresponde al valor calculado para dicha posición.

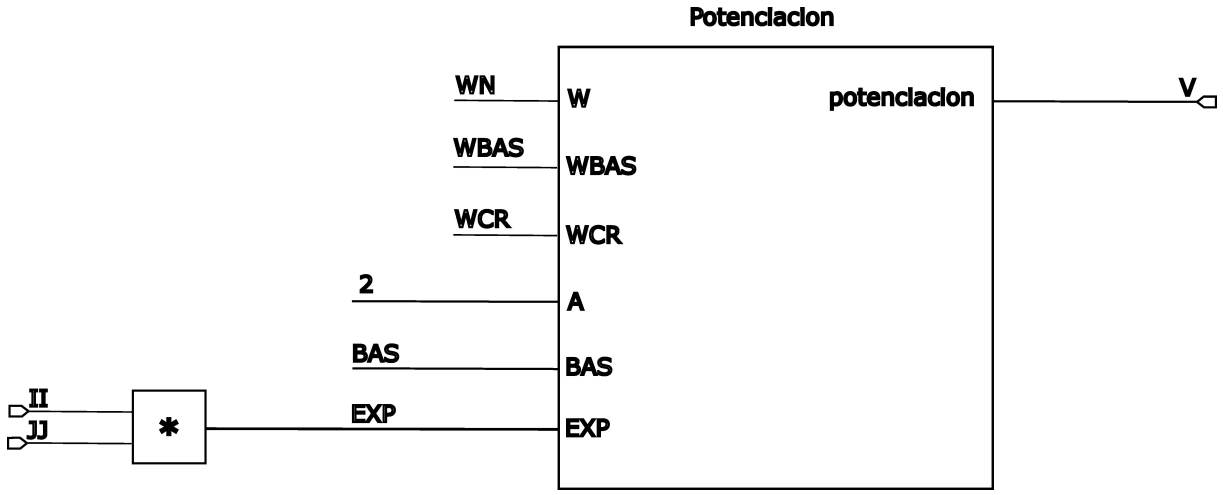
El funcionamiento del módulo se basa en la generación de un exponente a partir del producto de los índices de entrada, definido como `EXP = II * JJ`. Este valor determina la potencia a la cual se eleva un elemento constante del campo, en este caso el valor decimal 2, que en binario es 10, representando el elemento x que actúa como generador, ver 2.2.3.1.

La operación de potenciación se realiza mediante la instancia del módulo `Pot`, el cual implementa la exponenciación en el campo finito utilizando reducción modular respecto al polinomio base `BAS`. De esta manera, el valor de salida `V` corresponde a $2^{(i \cdot j)}$ dentro del campo definido. Esta

operación corresponde a la generación de los elementos del subconjunto soporte L, mencionado en la sección 2.5.1, y visto de manera extendida en las secciones 3.2.4 y 3.2.5.2.

Figura 6

Matriz V



El diseño del módulo presentado en la figura 6 es completamente combinacional, ya que la salida depende únicamente de las entradas actuales y del resultado de la operación de potenciación. Este enfoque permite calcular dinámicamente cada elemento de la matriz sin necesidad de almacenamiento intermedio.

3.3.2.3. Matriz D. El módulo `Matrix_D` implementa el cálculo de un elemento de la matriz diagonal D en el campo finito $\mathbb{F}_{2^m}$, cuyo valor depende tanto de un conjunto de coeficientes como de operaciones de potenciación.

Los parámetros del módulo son `WN`, `WBAS`, `WCR` y `Wg`, que corresponden a la longitud de los índices, la longitud del polinomio base del campo, la longitud del resultado y el tamaño del vector de coeficientes, respectivamente. Como entradas, el módulo recibe el vector `g`, que contiene los coeficientes del polinomio de Goppa, y los índices `II` y `JJ`, que definen la posición (i, j) dentro de la matriz. La salida `D` corresponde al valor calculado para dicha posición.

El funcionamiento del módulo se basa en la construcción de un elemento dependiente del índice `II`, seguido de una combinación ponderada de sus potencias. Inicialmente, se calcula el

valor L como una potencia de un elemento generador del campo, específicamente $L = 2^{II}$, mediante el uso del módulo Pot.

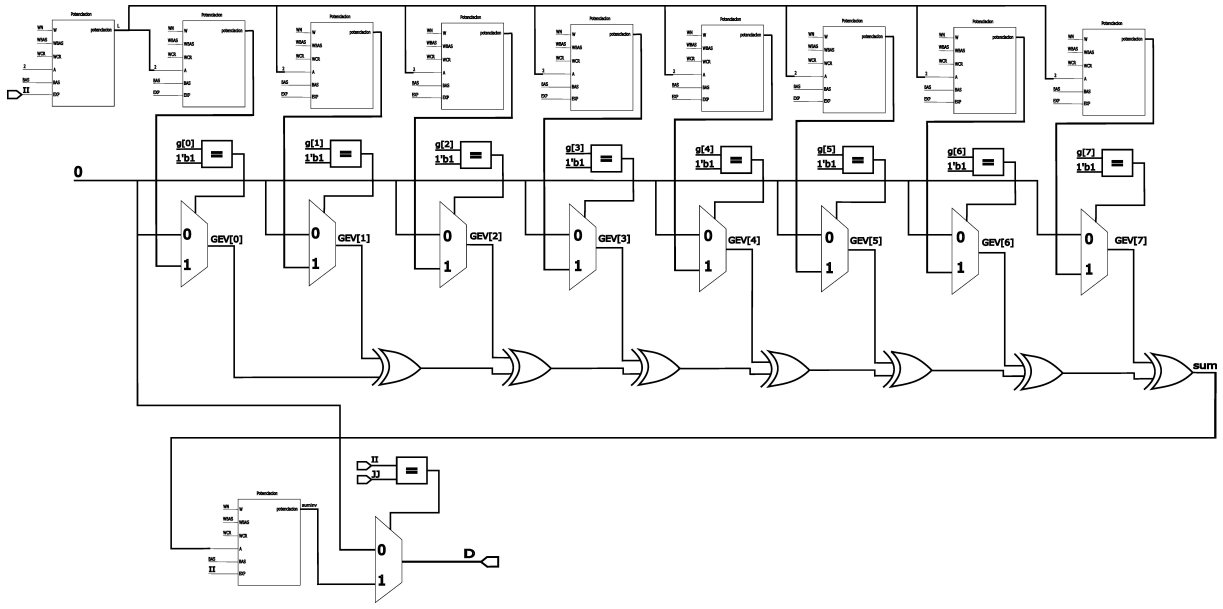
A partir de este valor, se generan múltiples potencias sucesivas de L, almacenadas en el arreglo GEV, esto equivale a evaluar el polinomio g para el valor de L. Posteriormente, estas potencias son filtradas mediante el vector de coeficientes g, de tal forma que únicamente se consideran aquellos términos cuyo coeficiente asociado es distinto de cero. Este proceso da lugar al arreglo GEV1.

Los elementos seleccionados se combinan mediante operaciones XOR, obteniendo así un valor acumulado SUM, que representa una suma en el campo $\mathbb{F}_2$. A continuación, se calcula el inverso multiplicativo de este valor en el campo finito, utilizando nuevamente el módulo Pot, elevando SUM a la potencia fija 2^{m-2} que corresponde al inverso en el campo definido por el polinomio base BAS, esta es una propiedad muy útil de los campos finitos generados con polinomios irreducibles, ya que son cíclicos, mencionado en la sección 2.2.3. El resultado de esta operación se almacena en SUMINV.

Finalmente, el módulo asigna el valor de salida en función de la relación entre los índices II y JJ. Si ambos índices son iguales, es decir, sobre la diagonal principal, la salida D toma el valor SUMINV. En caso contrario, la salida es cero, lo que define la estructura diagonal de la matriz.

Figura 7

Matriz D



El diseño del módulo presentado en la figura 7 es completamente combinacional, integrando múltiples instancias del módulo `Pot` y operaciones lógicas en paralelo. Este enfoque permite calcular dinámicamente cada elemento de la matriz sin requerir almacenamiento intermedio, manteniendo la consistencia con la estrategia general de optimización de recursos adoptada en la generación de matrices auxiliares.

3.3.2.4. Matriz H. El módulo `Matrix_H` implementa el cálculo de un elemento de la matriz de paridad H, a partir de la combinación de las matrices auxiliares previamente definidas. Este módulo integra las estructuras `Matrix_T`, `Matrix_V` y `Matrix_D`, consolidando el proceso de construcción de la matriz dentro del campo finito $\mathbb{F}_{2^m}$.

Los parámetros del módulo son `WN`, `WBAS`, `WCR` y `Wg`, que definen las longitudes de los índices, del polinomio base del campo, del resultado y del vector de coeficientes, respectivamente. Como entradas, el módulo recibe `II` y `JJ`, que representan los índices de la posición (i, j) dentro de la matriz. La salida `H` corresponde al valor calculado para dicha posición.

El funcionamiento del módulo se basa en la combinación de productos entre los elementos de las matrices auxiliares. En una primera etapa, se generan múltiples instancias de los módulos

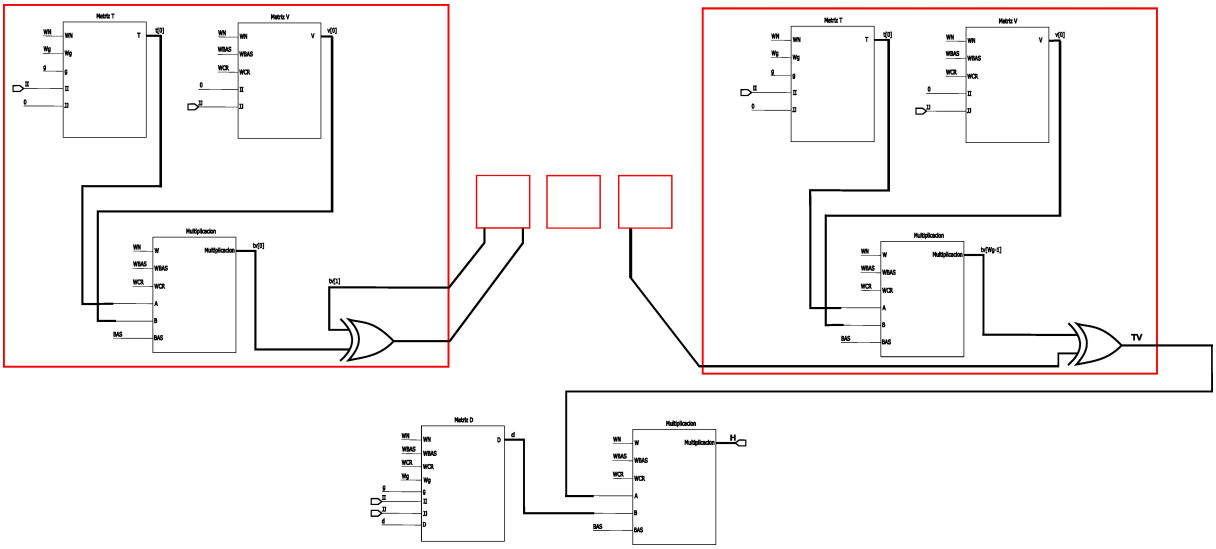
`Matrix_T` y `Matrix_V`. Posteriormente, estos valores son multiplicados en el campo finito utilizando el módulo `Multiplicacion`, generando los términos parciales `tv[k]`.

En una segunda etapa, los productos parciales se combinan mediante operaciones XOR, obteniendo un valor acumulado `TV`, que corresponde a la suma en $\mathbb{F}_2$ de los términos generados. Este proceso implementa una forma de producto matricial entre las estructuras `T` y `V`.

Adicionalmente, se calcula un factor mediante el módulo `Matrix_D`; finalmente, el resultado acumulado `TV` es multiplicado por el factor `d` utilizando nuevamente el módulo `Multiplicacion`, obteniendo así el valor final de la salida `H`. De este modo, cada elemento de la matriz de paridad se calcula como la combinación de estructuras auxiliares y operaciones en el campo finito.

Figura 8

Matriz H



El diseño del módulo presentado en la figura 8 es completamente combinacional y se encuentra totalmente desplegado, lo que permite calcular cualquier elemento de la matriz *H* de forma directa a partir de sus índices, sin requerir almacenamiento intermedio. Esta estrategia mantiene la coherencia con el enfoque general del diseño, orientado a minimizar la latencia y optimizar el uso de memoria mediante el cálculo dinámico de cada posición.

3.3.3. Generación de claves

Una vez completada la generación de la matriz de paridad H, se da paso a la etapa de generación de claves dentro del criptosistema de McEliece. En este contexto, la matriz generadora G juega un papel fundamental, ya que permite la construcción de las claves públicas a partir de transformaciones lineales y permutaciones que ocultan la estructura del código subyacente.

Con el fin de mantener la flexibilidad del diseño y permitir su aplicación en diferentes variantes del criptosistema, el alcance del módulo se establece hasta la obtención de la matriz G. A partir de este punto, los procedimientos pueden variar dependiendo del esquema considerado. Por ejemplo, en el criptosistema de Niederreiter se emplea una formulación basada en matrices de paridad y síndromes, mientras que en Classic McEliece se adoptan parámetros y estructuras optimizadas para seguridad post-cuántica.

Sin embargo, independientemente de estas diferencias, las técnicas implementadas para la construcción de H y G constituyen una base común sólida, sobre la cual es posible desarrollar cualquiera de estos esquemas criptográficos.

3.3.3.1. Generación de la matriz G. Como se mencionó en la Sección 3.2.6, a partir de la matriz de paridad H se obtiene la matriz generadora del código G, la cual satisface la relación $G \cdot H^T = 0$. En el entorno software, este proceso se realiza de manera directa mediante herramientas como SageMath, que disponen de funciones específicas para el manejo y transformación de matrices en campos finitos.

No obstante, para su implementación en hardware es necesario adoptar un enfoque algorítmico que permita resolver esta relación de forma estructurada. En este trabajo se emplea el método de Gauss-Jordan para la reducción de matrices, aplicado sobre la matriz H. Cabe destacar que, en general, H no es una matriz cuadrada, por lo que el procedimiento se orienta a llevarla a una forma reducida por filas.

Al aplicar el método de Gauss-Jordan, la matriz resultante adopta una forma sistemática, en la cual se obtiene una submatriz identidad concatenada con otra matriz que contiene los coeficientes restantes. Esta representación permite identificar directamente la matriz generadora G en su forma sistemática.

Por lo tanto, el proceso de obtención de G en hardware se basa en la implementación del

algoritmo de Gauss-Jordan sobre la matriz H, constituyendo un paso fundamental dentro del diseño del criptosistema.

3.3.3.2. Algoritmo de Gauss-Jordan. El módulo `Gauss_Jordan` implementa el algoritmo de reducción por filas de Gauss-Jordan sobre una matriz definida en el campo finito $\mathbb{F}_2$, con el objetivo de obtener su forma reducida y, en particular, su representación sistemática.

Los parámetros del módulo son `f`, `c` y `b`, que corresponden al número de filas, número de columnas y ancho de palabra, respectivamente. Como entradas, el módulo recibe la señal de reloj `clk`, la señal de reinicio `reset`, una señal de inicio `start`, y la matriz de entrada `matrix_in`, representada como un vector unidimensional. La salida `matrix_out` contiene la matriz resultante tras la reducción, junto con la señal `done`, que indica la finalización del proceso.

Internamente, la matriz se almacena en una estructura bidimensional `matrix`, y se emplean vectores de mapeo `row_map` y `col_map` para representar permutaciones de filas y columnas sin necesidad de modificar físicamente los datos en memoria. Este enfoque permite realizar intercambios de manera eficiente, reduciendo el costo en hardware.

El funcionamiento del módulo está controlado mediante una máquina de estados finitos (FSM), la cual coordina las distintas etapas del algoritmo de Gauss-Jordan:

- **LOAD**: Se inicializan los registros internos, incluyendo la matriz y los vectores de mapeo. El módulo permanece en este estado hasta que la señal `start` se activa.

- **PIVOTR_INIT**: Se inicializa la búsqueda de un pivote en la columna actual, comenzando desde la fila actual.

- **PIVOTR_SEARCH**: Se recorre la matriz en busca de una fila que contenga un elemento distinto de cero en la columna actual. Si se encuentra, se almacena su índice; en caso contrario, se continúa con la búsqueda de pivote en columnas.

- **PIVOTC_INIT**: Se inicializa la búsqueda de un pivote en la fila actual, recorriendo las columnas restantes.

- **PIVOTC_SEARCH**: Se busca una columna que contenga un elemento distinto de cero en la fila actual. Si no se encuentra un pivote válido, el sistema transita al estado de error.

- **SWAPR**: Se realiza el intercambio lógico de filas mediante la actualización del vector `row_map`, alineando el pivote encontrado con la fila actual.

- **SWAPC**: De manera análoga, se intercambian columnas utilizando el vector `col_map`, alineando el pivote con la columna actual.

- **ELIMINATE**: Se lleva a cabo la eliminación gaussiana. Para cada fila distinta de la fila pivote, si el elemento en la columna actual es distinto de cero, se realiza una operación XOR con la fila pivote. Este proceso elimina los valores en la columna, generando ceros en todas las posiciones excepto en el pivote. Posteriormente, se incrementan los índices de fila y columna para continuar con la siguiente iteración.

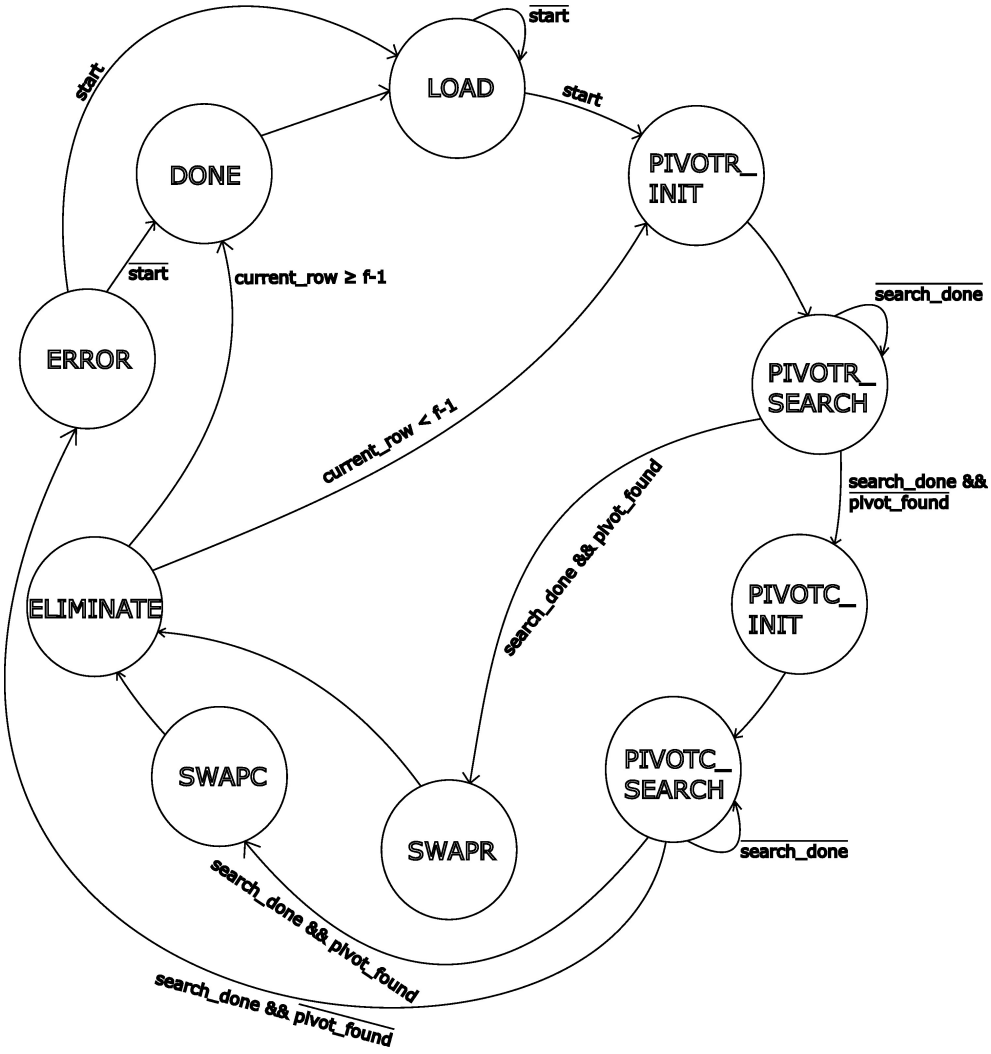
- **ERROR**: Se activa cuando no es posible encontrar un pivote válido, indicando que la matriz no puede ser reducida completamente bajo las condiciones esperadas.

- **DONE**: Indica la finalización del proceso. En este estado, se reconstruye la matriz de salida `matrix_out` aplicando los mapeos de filas y columnas, y se activa la señal `done`.

Las transiciones entre estados están determinadas principalmente por el resultado de las búsquedas de pivote (`pivot_found`) y la finalización de los recorridos (`search_done`). En cada iteración, el algoritmo asegura la existencia de un pivote válido mediante intercambios de filas o columnas, permitiendo avanzar progresivamente hacia una forma reducida por filas.

Figura 9

FSM Gauss Jordan



El diseño del módulo presentado en la figura 9 es secuencial y orientado a control, donde cada estado de la FSM corresponde a una etapa específica del algoritmo. Este enfoque permite manejar matrices de mayor tamaño con un uso eficiente de recursos, en contraste con implementaciones completamente combinacionales, ya que se intentó realizar un diseño combinacional, sin embargo el costo lógico y de recursos es enorme y no es viable para su uso.

3.3.3.3. Matriz G. El módulo `Generaciónacion_de_claves` implementa el proceso completo de obtención de la matriz generadora G a partir de la matriz de paridad H. Este

bloque integra la generación dinámica de la matriz H junto con su posterior reducción mediante el algoritmo de Gauss-Jordan, constituyendo una etapa central dentro del criptosistema.

Los parámetros del módulo son `F` y `C`, que corresponden al número de filas y columnas de la matriz H, respectivamente. Como entradas, el módulo recibe las señales `clk`, `reset` y `start`. La salida `matrix_out` contiene la matriz resultante tras el proceso de reducción, mientras que la señal `done` indica la finalización de la operación.

El funcionamiento del módulo se divide en dos etapas principales: la generación de la matriz de paridad y su posterior transformación a forma sistemática.

En la primera etapa, se construye la matriz H de manera secuencial utilizando la instancia del módulo `Matrix_H`. Para ello, se emplean los índices `II` y `JJ`, que recorren las filas y columnas de la matriz. En cada iteración, se calcula un elemento $H(i, j)$, el cual es almacenado en el buffer `matrix_buffer`. Este proceso evita la necesidad de almacenar matrices intermedias completas, alineándose con la estrategia de optimización de recursos planteada previamente.

Una vez completado el llenado de la matriz, el módulo inicia la segunda etapa, en la cual se aplica el algoritmo de Gauss-Jordan mediante la instancia del módulo `Gauss_Jordan`. Este proceso transforma la matriz H en una forma sistemática, permitiendo obtener la matriz generadora G.

El comportamiento del módulo está gobernado por una máquina de estados finitos (FSM), cuyas etapas se describen a continuación:

- **IDLE**: Estado inicial. El módulo espera la activación de la señal `start`. Al activarse, se inicializan los índices `II` y `JJ`.

- **GEN_H**: Se calcula el valor del elemento $H(i, j)$ mediante el módulo `Matrix_H`. Este estado representa la evaluación combinacional del elemento actual.

- **STORE**: El valor calculado se almacena en la posición correspondiente del buffer `matrix_buffer`. La asignación se realiza considerando la representación interna de los elementos del campo.

- **NEXT**: Se actualizan los índices `II` y `JJ` para avanzar al siguiente elemento de la matriz. El recorrido se realiza de forma secuencial por filas.

- **GAUSS**: Una vez completado el llenado de la matriz, se activa la señal `start_gj` para

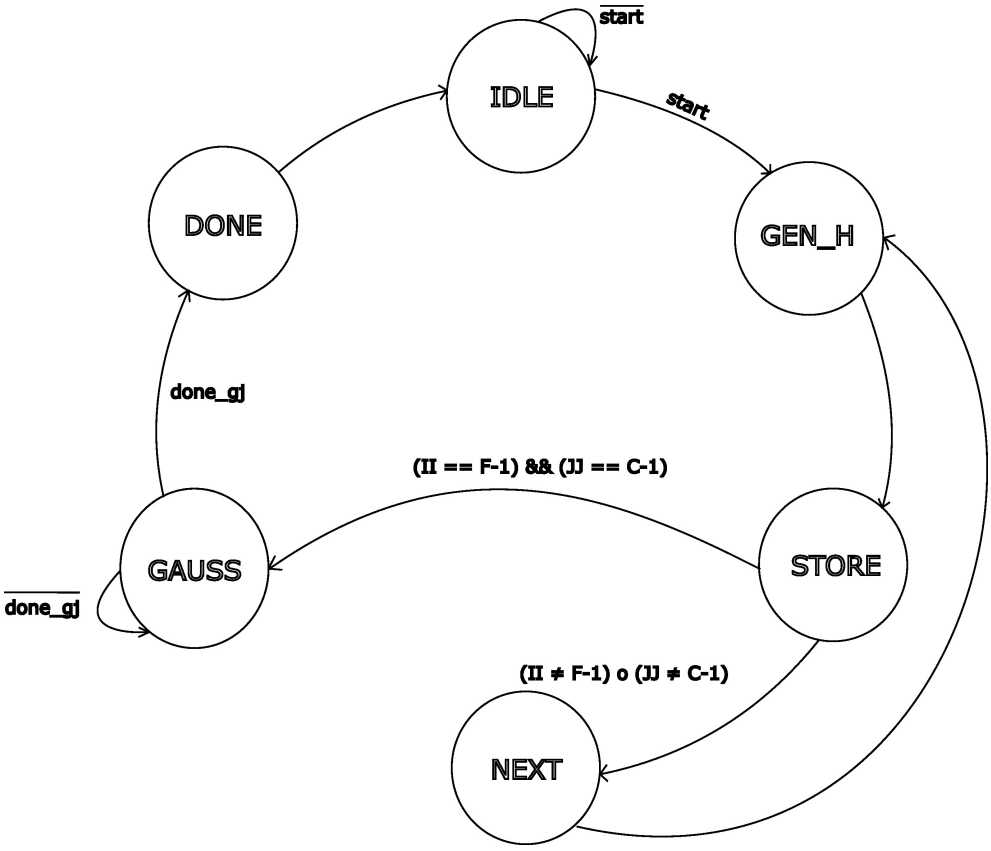
iniciar el módulo `Gauss_Jordan`. El sistema permanece en este estado hasta que la señal `done_gj` indica la finalización del proceso.

• **DONE**: Se almacena el resultado final en `matrix_out` y se activa la señal `done`, indicando que la matriz generadora ha sido obtenida.

Las transiciones de estado están determinadas por el avance de los índices y la finalización del módulo de reducción. En particular, el paso del estado `STORE` al estado `GAUSS` ocurre cuando se ha completado el recorrido de toda la matriz, es decir, cuando `II = F-1` y `JJ = C-1`. Posteriormente, el sistema espera la señal de finalización del módulo de Gauss-Jordan para concluir el proceso.

Figura 10

Generación de la matriz G



El diseño del módulo presentado en la figura 10 es secuencial y orientado a control, permitiendo la construcción progresiva de la matriz y su posterior procesamiento. Este enfoque equilibra

el uso de recursos y la complejidad del sistema, integrando de manera eficiente los bloques desarrollados previamente para la generación de la matriz H y su transformación en la matriz generadora G.

3.4. Resumen

En este capítulo se presentó el desarrollo del criptosistema de McEliece mediante un enfoque progresivo que parte de su formulación matemática y culmina en su implementación en hardware a nivel RTL.

Inicialmente, se realizó una implementación en software utilizando SageMath, lo que permitió definir el campo finito $GF(2^m)$, el polinomio de Goppa y el conjunto soporte. A partir de estos elementos, se construyó la matriz de paridad H y se obtuvo la matriz generadora G mediante el cálculo de su núcleo. Asimismo, se generaron las matrices auxiliares S y P, y se validó el proceso completo de codificación y decodificación empleando el algoritmo de Patterson.

Posteriormente, se abordó el diseño en hardware, comenzando con la implementación de las operaciones básicas en $\mathbb{F}_{2^m}$. Estos bloques fueron utilizados para construir de manera modular las matrices auxiliares T, V y D, permitiendo la generación dinámica de la matriz H sin requerir almacenamiento completo. Finalmente, se implementó el algoritmo de Gauss-Jordan para obtener la forma sistemática de la matriz y derivar la matriz generadora G.

El resultado es una arquitectura modular que articula de manera coherente los distintos niveles del diseño, desde la descripción algebraica hasta su realización en hardware. Este enfoque no solo garantiza la correcta funcionalidad del sistema, sino que también establece una base sólida para su análisis, validación y posible optimización en futuras etapas.

4. Análisis y resultados

En este capítulo se presentan y analizan los resultados obtenidos a lo largo del desarrollo del proyecto. Se evalúa el desempeño de los módulos individuales en síntesis e implementación, centrándose en la frecuencia máxima de operación, el consumo de potencia y la utilización de recursos de hardware, con el fin de caracterizar el comportamiento del diseño en términos de eficiencia y escalabilidad.

Para ello, se realiza un desglose de los resultados en función del parámetro m, el cual determina el grado de extensión del campo finito $GF(2^m)$ y, en consecuencia, el tamaño de las matrices involucradas y la complejidad de las operaciones aritméticas. Se consideran valores desde $m = 4$ hasta $m = 8$, lo que permite observar el impacto del crecimiento del campo en el rendimiento global del sistema.

4.1. Análisis de resultados por módulo

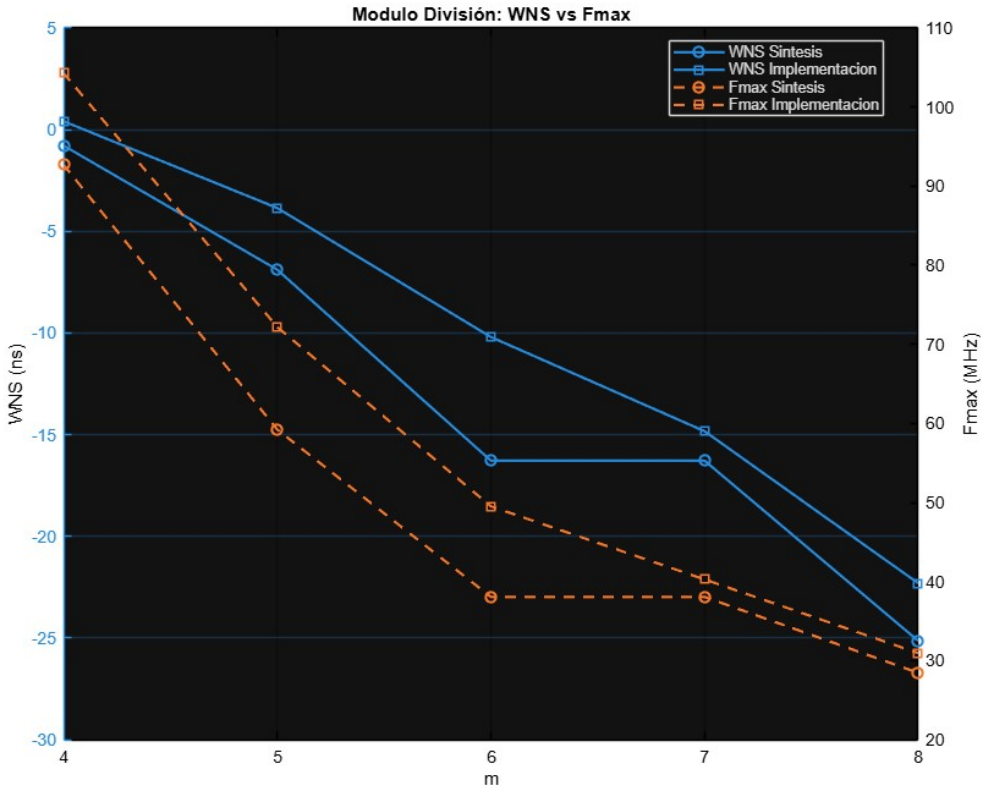
Previo al análisis detallado de cada módulo, es importante establecer algunas consideraciones generales del proceso de validación e implementación. Todos los módulos fueron verificados mediante simulación utilizando testbench, comparando sus resultados con una implementación de referencia en SageMath, lo que permitió comprobar su correcto funcionamiento. Dado que la mayoría de los módulos presentan un comportamiento netamente combinacional, para efectos de la síntesis e implementación —y con el fin de obtener reportes temporales— estos fueron encapsulados entre registros. Esto permitió medir el retardo de propagación de las señales y, en consecuencia, estimar la frecuencia máxima de operación. Por lo tanto, en los resultados se observará un consumo de registros que no corresponde directamente al comportamiento intrínsecamente combinacional de los módulos, sino a la estrategia empleada para su caracterización temporal.

La FPGA seleccionada para la síntesis e implementación fue la XC7A200T-SB484-1, perteneciente a la familia Artix-7 de Xilinx. El entorno de desarrollo utilizado fue Vivado 2025.2, empleando las estrategias de síntesis e implementación configuradas por defecto. Adicionalmente, fue necesario definir restricciones temporales para la generación de los reportes, estableciendo un periodo de reloj de 10.00 ns, lo que corresponde a una frecuencia objetivo de 100 MHz para el diseño.

4.1.1. División

Figura 11

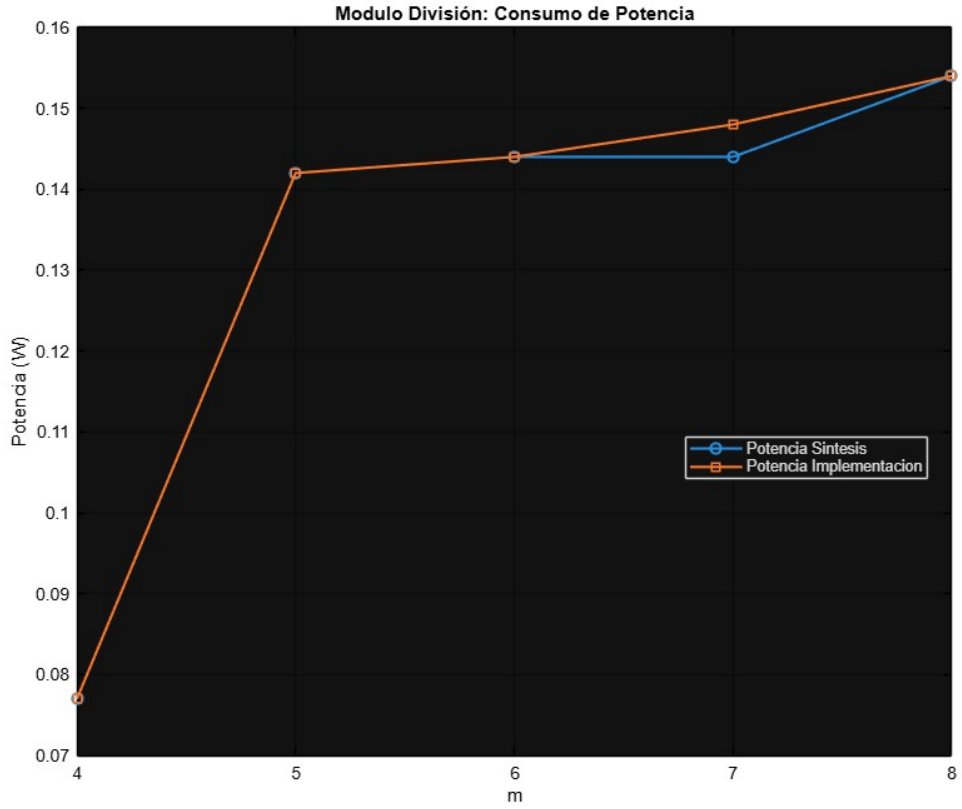
Frecuencia máxima de operación del módulo de división en función del parámetro m, comparando resultados de síntesis e implementación



En términos temporales, en la figura 11 se observa que los valores de *Worst Negative Slack* (WNS) evidencian una degradación significativa a medida que aumenta el parámetro *m*. Para valores pequeños de *m*, la implementación aún logra cumplir parcialmente las restricciones temporales; sin embargo, conforme *m* crece, el WNS se vuelve progresivamente más negativo tanto en síntesis como en implementación. Este comportamiento indica que el retardo crítico del circuito aumenta de forma considerable, reduciendo la frecuencia máxima de operación alcanzable.

Figura 12

Consumo de potencia total del módulo de división en función del parámetro m, para síntesis e implementación



En cuanto al consumo de potencia, en la figura 12 se observa un incremento moderado conforme aumenta *m*, manteniéndose en un rango relativamente bajo. Este crecimiento controlado sugiere que, aunque el número de operaciones lógicas se incrementa, la actividad de conmutación no escala de forma abrupta, lo que es consistente con una arquitectura combinacional sin realimentaciones ni estructuras altamente dinámicas.

Tabla 4

Consumo de recursos del módulo de división

m	LUTs		Registers		Bonded IOB		BUFGCTRL	
	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp
4	68	57	23	23	29	29	1	1
5	82	96	32	33	37	37	1	1
6	93	107	39	40	45	45	1	1
7	93	154	39	46	45	53	1	1
8	201	214	53	54	61	61	1	1

Desde el punto de vista de utilización de recursos, como se puede observar en la tabla 4, el módulo presenta un crecimiento general en el consumo de LUTs conforme aumenta m, con un incremento particularmente pronunciado en la etapa de implementación para valores altos (especialmente en $m = 7$ y $m = 8$). Esta diferencia entre síntesis e implementación sugiere la presencia de optimizaciones y reestructuraciones internas realizadas por la herramienta, que pueden introducir mayor uso de lógica para cumplir restricciones físicas y de ruteo.

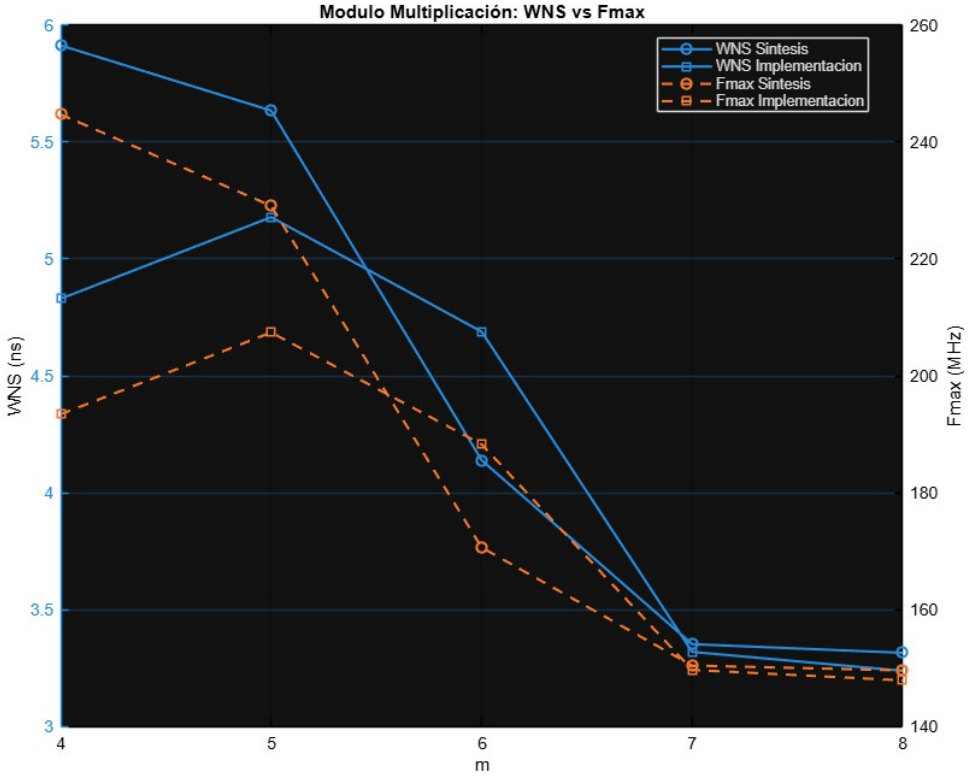
El uso de registros también muestra un incremento progresivo, aunque este comportamiento está influenciado por la estrategia de encapsulamiento utilizada para caracterizar temporalmente el diseño, más que por la naturaleza intrínseca del módulo. Por su parte, los recursos de entrada/salida (IOB) crecen de manera proporcional al tamaño de los operandos, mientras que el uso de recursos de reloj (BUFGCTRL) permanece constante.

En conjunto, estos resultados permiten identificar al módulo de división como uno de los componentes más críticos desde el punto de vista temporal. Su escalabilidad en términos de frecuencia es limitada bajo una arquitectura completamente combinacional, lo que lo convierte en un potencial cuello de botella para valores elevados de m. Este comportamiento sugiere que futuras optimizaciones, como la introducción de segmentación mediante *pipeline* o el uso de arquitecturas alternativas, podrían ser necesarias para mejorar su desempeño en sistemas de mayor tamaño.

4.1.2. Multiplicación

Figura 13

Frecuencia máxima de operación del módulo de multiplicación en función del parámetro m, comparando resultados de síntesis e implementación



En términos temporales, el módulo de multiplicación presenta un comportamiento estable como se logra observar en la figura 13, con valores de WNS positivos tanto en síntesis como en implementación para todos los valores de *m*. Aunque se observa una ligera disminución del margen temporal a medida que aumenta *m*, este decrecimiento es gradual y no compromete el cumplimiento de las restricciones de tiempo.

Este comportamiento sugiere que la profundidad lógica del módulo crece de manera controlada, lo cual es consistente con una arquitectura basada en operaciones paralelas que no generan rutas críticas excesivamente largas. En consecuencia, la frecuencia máxima de operación se mantiene relativamente alta incluso para valores mayores de *m*.

Ahora bien, considerando la arquitectura del módulo de multiplicación (ver Figura 3), es importante destacar que este incorpora una operación equivalente a una división al final del proceso, con el fin de realizar la reducción polinomial. No obstante, a diferencia del módulo de división completo, esta operación no introduce los mismos retardos críticos.

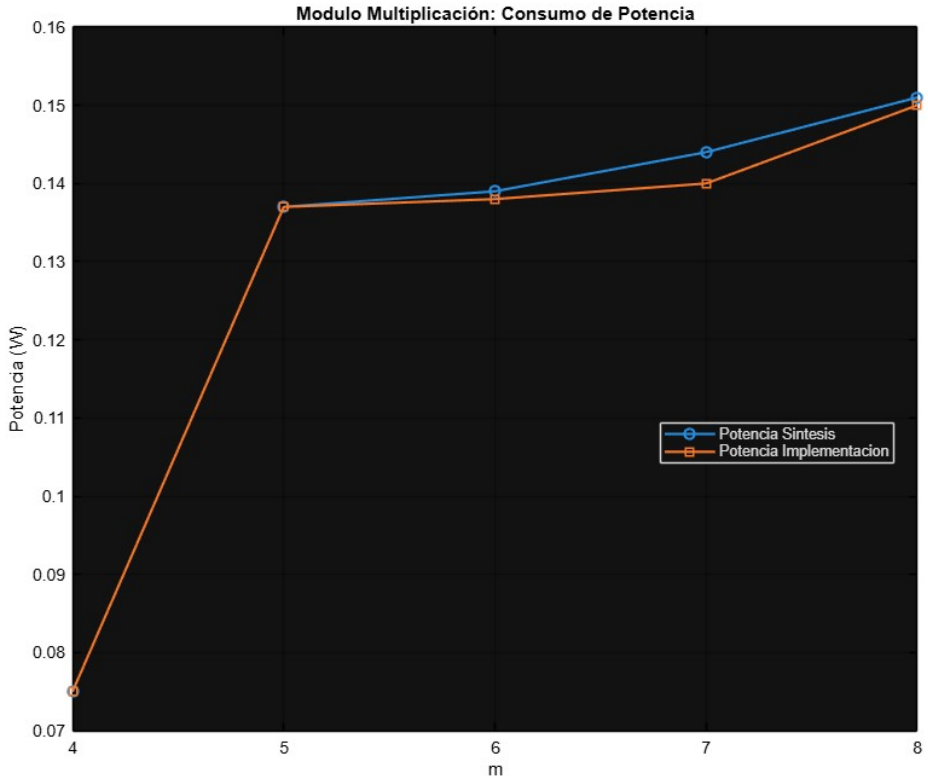
Esta diferencia se debe a que el módulo de división, como se observa en la Figura 2, incluye una etapa previa de procesamiento de datos, en la cual se realiza una extensión de bits y una reorganización de los operandos antes de ejecutar la operación principal. Esta etapa adicional incrementa la profundidad lógica total y, en consecuencia, afecta negativamente el desempeño temporal.

Por el contrario, en el módulo de multiplicación no se emplea el módulo de división completo, sino únicamente la lógica necesaria para efectuar la reducción. Esto es posible debido a que el procesamiento de los datos ya se lleva a cabo implícitamente durante la propia operación de multiplicación, evitando así etapas adicionales de preprocesamiento.

En este sentido, la comparación entre ambos módulos permite identificar con mayor claridad que la principal limitación temporal del módulo de división no radica únicamente en la operación aritmética en sí, sino en las etapas adicionales de preparación de datos que incrementan la longitud de la ruta crítica.

Figura 14

Consumo de potencia total del módulo de multiplicación en función del parámetro m, para síntesis e implementación



En cuanto al consumo de potencia, en la figura 14 se observa un incremento progresivo con el aumento de *m*, aunque con una pendiente moderada. Este crecimiento está asociado al aumento en el número de operaciones lógicas requeridas, pero sin evidenciar un incremento abrupto en la actividad de conmutación.

Tabla 5

Consumo de recursos del módulo de multiplicación

m	LUTs		Registers		Bonded IOB		BUFGCTRL	
	Syn	**Imp**	**Syn**	**Imp**	**Syn**	**Imp**	**Syn**	**Imp**
4	21	16	22	22	26	26	1	1
5	30	30	28	28	33	33	1	1
6	46	46	34	34	40	40	1	1
7	63	63	40	40	47	47	1	1
8	109	88	46	46	54	54	1	1

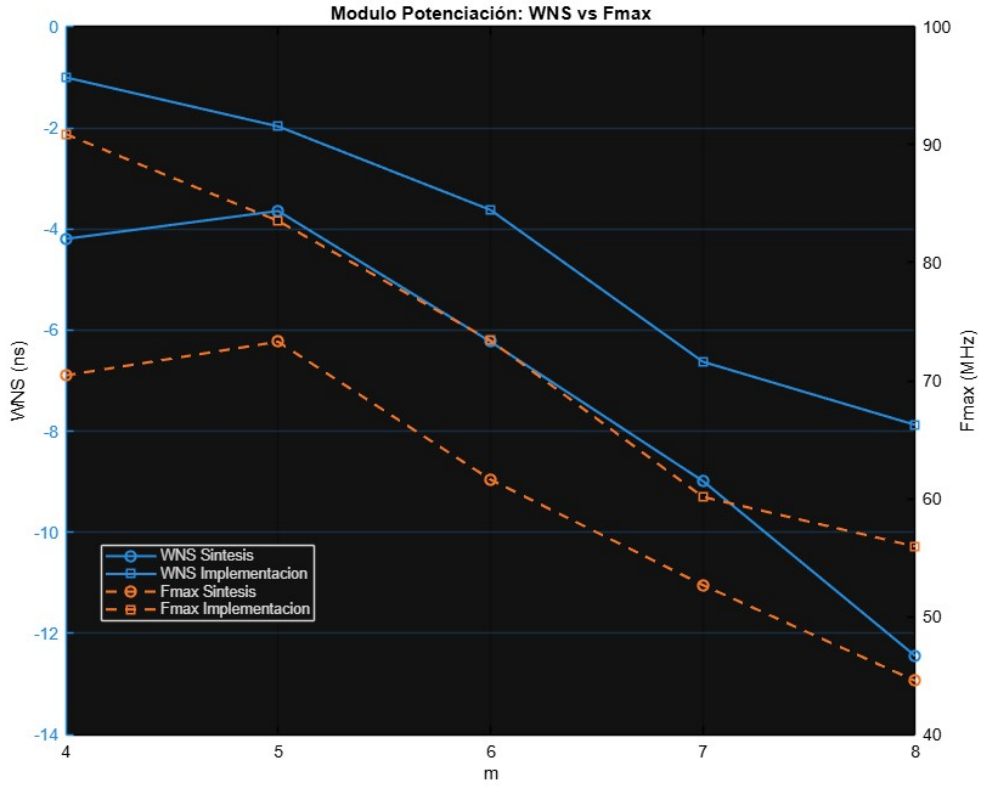
Desde el punto de vista de recursos, en la tabla 5 se aprecia que el consumo de LUTs crece de forma aproximadamente lineal, reflejando la escalabilidad del algoritmo implementado. A diferencia del módulo de división, no se observan incrementos abruptos en la etapa de implementación, lo que indica una mejor correspondencia entre la descripción lógica y su mapeo físico.

En conjunto, estos resultados posicionan al módulo de multiplicación como un bloque eficiente y escalable, con un comportamiento predecible tanto en términos de tiempo como de recursos.

4.1.3. Potenciación

Figura 15

Frecuencia máxima de operación del módulo de potenciación en función del parámetro m, comparando resultados de síntesis e implementación

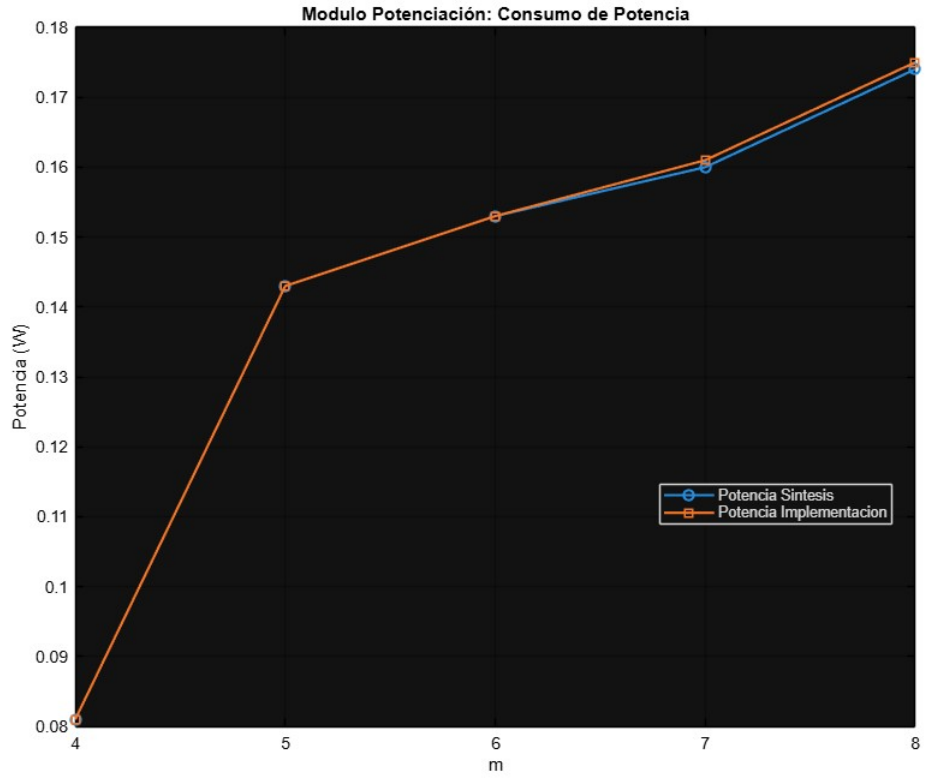


El análisis temporal del módulo de potenciación que se puede observar en la figura 15 revela un comportamiento crítico, caracterizado por valores de WNS negativos incluso para valores pequeños de *m*, los cuales se degradan progresivamente a medida que el tamaño del campo aumenta. Esto indica que el módulo no cumple las restricciones temporales bajo las condiciones evaluadas, limitando significativamente su frecuencia máxima de operación. Este comportamiento se explica por la naturaleza del algoritmo de potenciación, que implica la ejecución secuencial de múltiples multiplicaciones. Al estar implementado de forma completamente combinacional, el retardo total corresponde a la acumulación de los retardos de cada operación intermedia, generando

una ruta crítica considerablemente larga.

Figura 16

Consumo de potencia total del módulo de potenciación en función del parámetro m, para síntesis e implementación



El consumo de potencia que se puede observar en la figura 16 presenta un incremento más pronunciado en comparación con otros módulos, especialmente para valores altos de *m*. Este comportamiento está directamente relacionado con el elevado número de operaciones internas y el aumento en la actividad de conmutación asociada, sin embargo el consumo a manera general es bajo.

Tabla 6

Consumo de recursos del módulo de potenciación

m	LUTs		Registers		Bonded IOB		BUFGCTRL	
	Syn	**Imp**	**Syn**	**Imp**	**Syn**	**Imp**	**Syn**	**Imp**
4	176	179	24	34	25	25	1	1
5	151	186	20	23	21	21	1	1
6	312	346	23	27	24	24	1	1
7	395	403	26	31	27	27	1	1
8	862	864	45	78	45	45	1	1

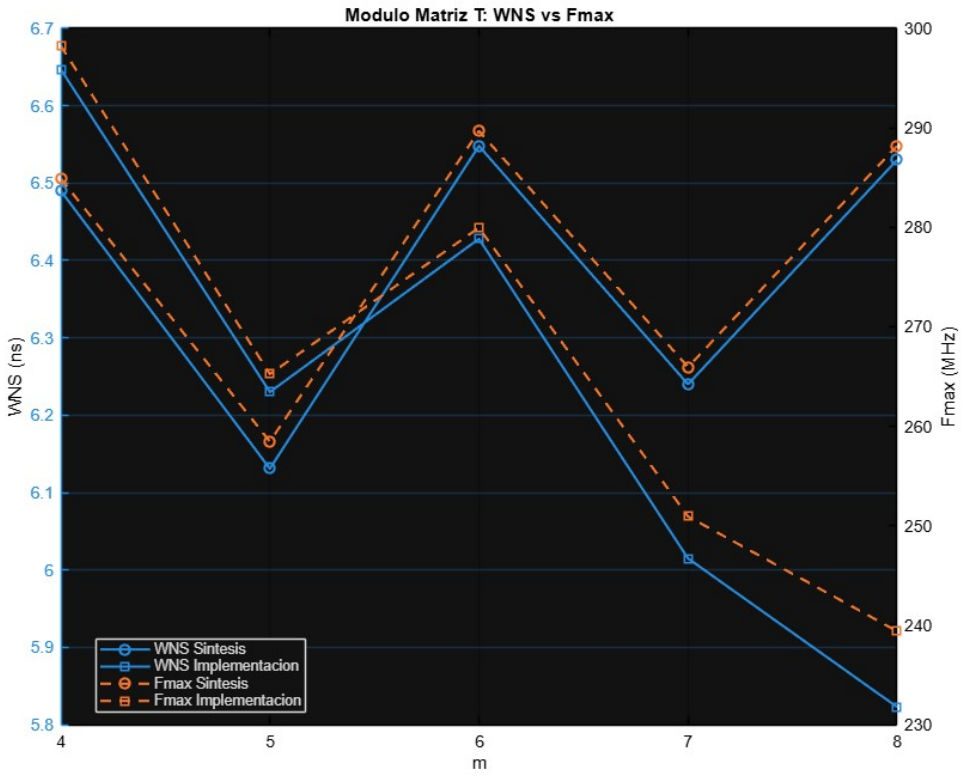
En términos de recursos, en la tabla 6 se aprecia que el crecimiento en el uso de LUTs es significativamente mayor que en otros módulos, alcanzando valores elevados para $m = 8$. Esto refleja la complejidad inherente del módulo, donde múltiples instancias de operaciones aritméticas se combinan en una única estructura combinacional.

En conjunto, el módulo de potenciación se identifica como uno de los más exigentes tanto en términos de tiempo como de recursos, constituyendo un claro candidato a optimización mediante técnicas como reutilización de hardware o segmentación temporal.

4.1.4. Matriz T

Figura 17

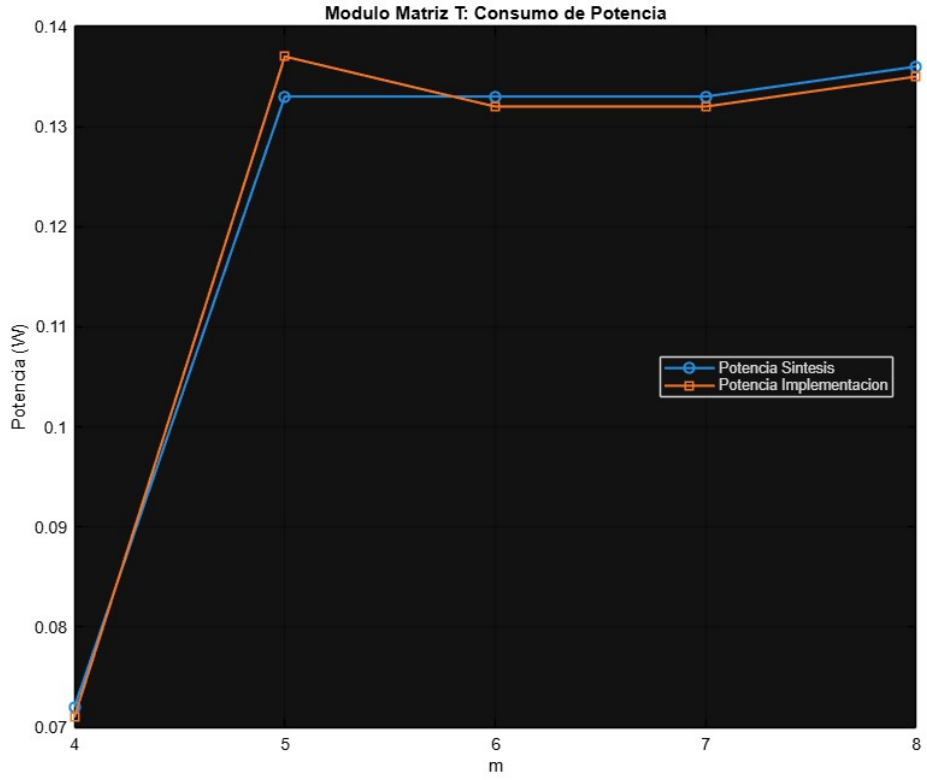
Frecuencia máxima de operación del módulo Matriz T en función del parámetro m, comparando resultados de síntesis e implementación



Como se puede observar en la figura 17, el modulo presenta un comportamiento temporal altamente favorable, con valores de WNS positivos y estables para todos los valores de *m*. La variación observada es mínima, lo que indica que la profundidad lógica del circuito es reducida y no crece significativamente con el tamaño de los parametros.

Figura 18

Consumo de potencia total del módulo Matriz T en función del parámetro m, para síntesis e implementación



El consumo de potencia observado en la figura 18 se mantiene bajo y prácticamente constante, con ligeras variaciones atribuibles al incremento en el tamaño de los datos. Esto sugiere una baja actividad de conmutación y una estructura lógica sencilla.

Tabla 7

Consumo de recursos del módulo Matriz T

m	LUTs		Registers		Bonded IOB		BUFGCTRL	
	Syn	**Imp**	**Syn**	**Imp**	**Syn**	**Imp**	**Syn**	**Imp**
4	15	14	19	19	26	26	1	1
5	17	17	24	24	33	33	1	1
6	21	21	29	29	40	40	1	1
7	25	25	34	34	47	47	1	1
8	27	26	39	39	54	54	1	1

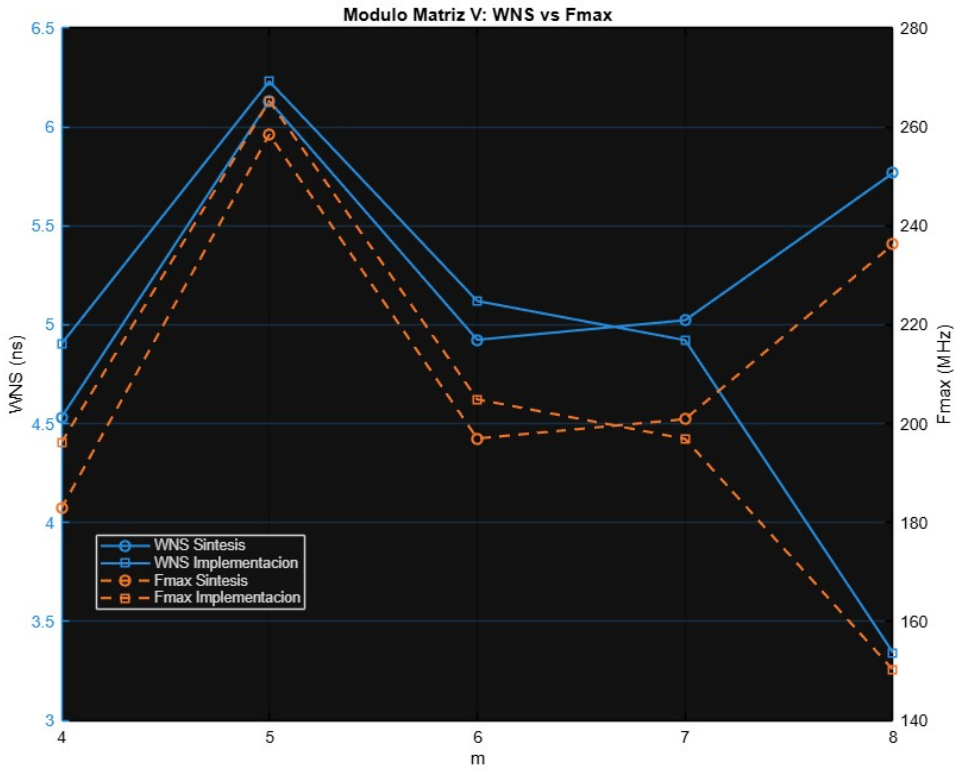
En términos de recursos, en la tabla 7 se observa que el crecimiento en LUTs y registros es leve y lineal, confirmando que se trata de un módulo de baja complejidad. En consecuencia, la matriz T no representa un componente crítico dentro del sistema.

En conjunto el modulo presenta un comportamiento favorable para su implementación incluso con valores de *m* mas grandes.

4.1.5. Matriz V

Figura 19

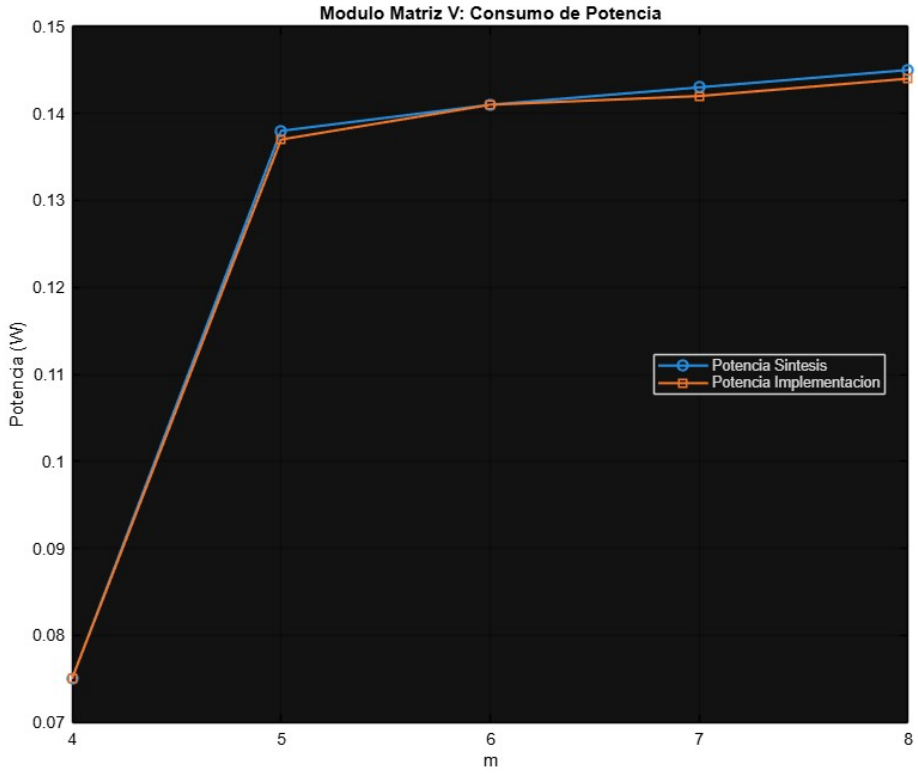
Frecuencia máxima de operación del módulo Matriz V en función del parámetro m, comparando resultados de síntesis e implementación



Con base en la figura 19 el comportamiento del módulo matriz V es similar al de la matriz T, aunque con una ligera mayor variabilidad en los valores de WNS. Aun así, todos los valores se mantienen en el rango positivo, garantizando el cumplimiento de las restricciones temporales.

Figura 20

Consumo de potencia total del módulo Matriz V en función del parámetro m, para síntesis e implementación



En cuanto al consumo de potencia, en la figura 20 se observa un incremento progresivo conforme aumenta el parámetro *m*, aunque con una pendiente moderada, la ausencia de estructuras complejas o altamente interconectadas limita la actividad de conmutación, lo que contribuye a mantener el consumo en niveles relativamente bajos y estables.

Tabla 8

Consumo de recursos del módulo Matriz V

m	LUTs		Registers		Bonded IOB		BUFGCTRL	
	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp
4	15	15	14	14	18	18	1	1
5	15	15	17	17	18	18	1	1
6	17	17	21	21	22	22	1	1
7	19	18	23	23	24	24	1	1
8	27	26	25	25	26	26	1	1

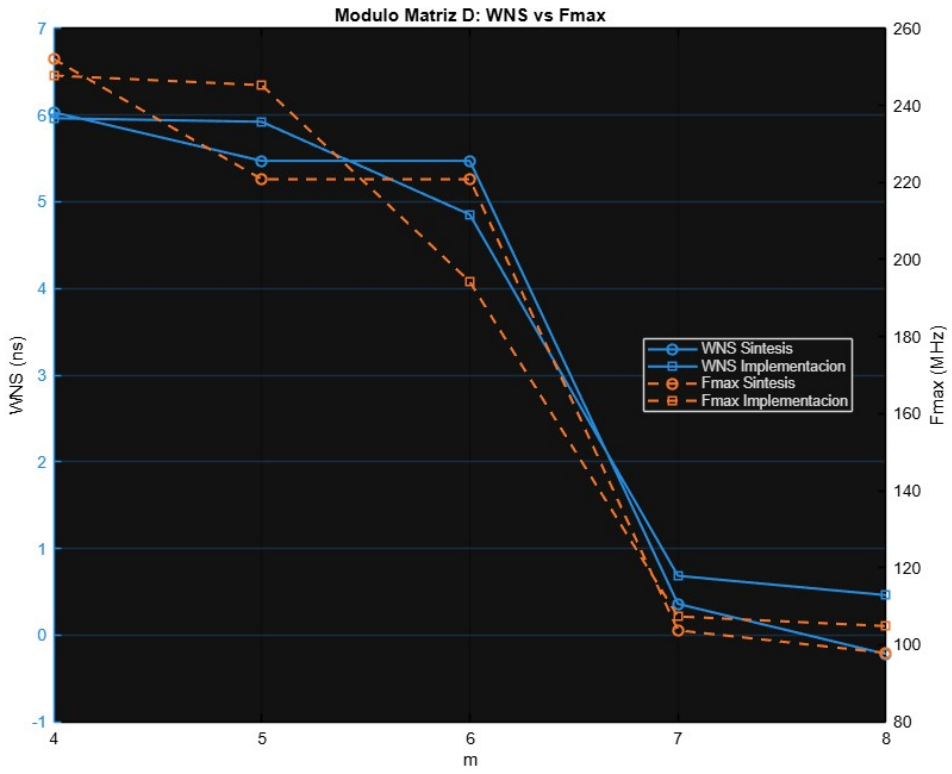
Desde el punto de vista de utilización de recursos, como se observa en la tabla 8 el módulo presenta un crecimiento lineal tanto en LUTs como en registros. Este comportamiento es consistente con la expansión del tamaño de los datos de entrada, sin evidenciar incrementos abruptos ni ineficiencias en el mapeo a hardware. Además, la similitud entre los resultados de síntesis e implementación sugiere que la herramienta logra una correspondencia adecuada entre la descripción lógica y su realización física, sin necesidad de introducir optimizaciones adicionales costosas.

En conjunto, estos resultados confirman que el módulo matriz V es eficiente y altamente escalable, manteniendo un balance favorable entre desempeño temporal, consumo de potencia y utilización de recursos, sin introducir limitaciones relevantes en el sistema global.

4.1.6. Matriz D

Figura 21

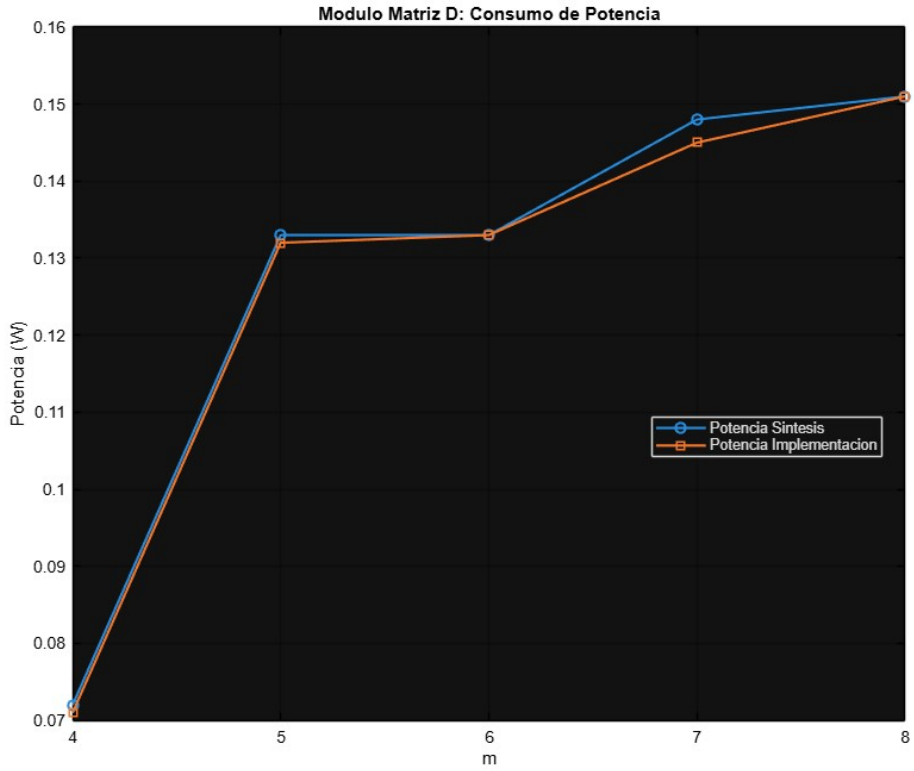
Frecuencia máxima de operación del módulo Matriz D en función del parámetro m, comparando resultados de síntesis e implementación



Con base en la figura 21, se observa que el módulo matriz D mantiene valores de WNS positivos para la mayoría de los casos, aunque se observa una degradación conforme aumenta *m*, llegando a valores cercanos a cero o ligeramente negativos en síntesis para valores altos, este comportamiento sugiere un incremento en la complejidad lógica que comienza a impactar la ruta crítica, aunque sin llegar a niveles tan restrictivos como en módulos más complejos.

Figura 22

Consumo de potencia total del módulo Matriz D en función del parámetro m, para síntesis e implementación



En cuanto al consumo de potencia, en la figura 22 se observa una tendencia creciente con el incremento de m, la cual se mantiene relativamente moderada para valores pequeños y medios, pero comienza a acentuarse a partir de $m = 7$. Este aumento está asociado no solo al mayor número de elementos lógicos involucrados, sino también a un incremento en la actividad de conmutación, derivado de una mayor interconexión interna entre señales. A diferencia de módulos más simples como las matrices T y V, la estructura de la matriz D implica una mayor densidad de operaciones, lo que se traduce en un consumo energético ligeramente superior y menos uniforme.

Tabla 9

Consumo de recursos del módulo Matriz D

m	LUTs		Registers		Bonded IOB		BUFGCTRL	
	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp
4	21	19	22	22	26	26	1	1
5	20	20	31	31	32	32	1	1
6	20	33	31	37	32	38	1	1
7	130	131	43	43	44	44	1	1
8	200	150	49	49	50	50	1	1

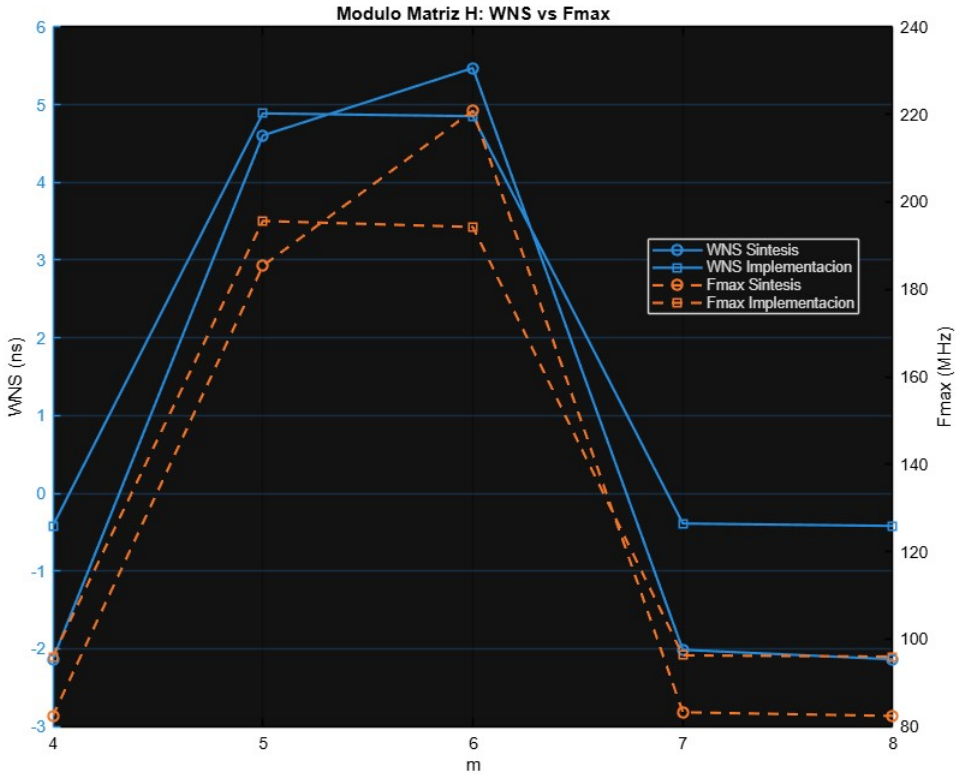
Desde el punto de vista de utilización de recursos, en la tabla 9 se observa que el módulo presenta un comportamiento no estrictamente lineal. Para valores bajos de m, el consumo de LUTs se mantiene relativamente contenido; sin embargo, a partir de $m = 7$ se evidencia un incremento considerable, especialmente en la etapa de implementación. Este crecimiento sugiere que las herramientas de síntesis y ruteo requieren introducir lógica adicional para satisfacer restricciones físicas, lo que puede estar relacionado con una mayor complejidad en la conectividad del diseño.

En conjunto, estos resultados indican que el módulo matriz D presenta una escalabilidad más limitada en comparación con otras estructuras matriciales más simples. Aunque no constituye un cuello de botella crítico en todos los escenarios, su comportamiento para valores altos de m evidencia una tendencia hacia la saturación de recursos y degradación temporal, lo que lo posiciona como un candidato relevante para futuras optimizaciones.

4.1.7. Matriz H

Figura 23

Frecuencia máxima de operación del módulo Matriz H en función del parámetro m, comparando resultados de síntesis e implementación

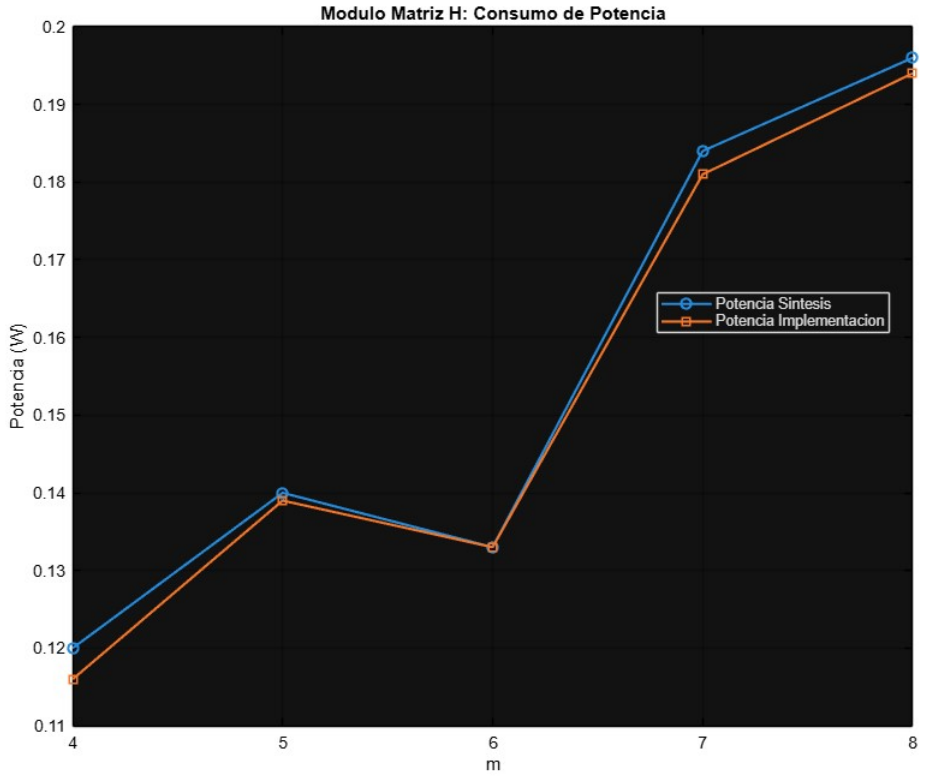


Como se observa en la figura 23, el módulo presenta un comportamiento temporal irregular, con valores de WNS que alternan entre positivos y negativos dependiendo del valor de *m*. Esto indica que el cumplimiento de las restricciones temporales no es consistente, especialmente para valores altos.

Este comportamiento puede atribuirse a una estructura lógica más compleja, con mayor interdependencia entre señales, lo que incrementa la variabilidad en la ruta crítica, esto podría mejorar cambiando la estrategia de síntesis e implementación, a una más especializada para la optimización de rutas críticas del diseño.

Figura 24

Consumo de potencia total del módulo Matriz H en función del parámetro m, para síntesis e implementación



En cuanto al consumo de potencia, en la figura 24 se observa una tendencia creciente conforme aumenta el parámetro *m*, con un incremento más notable a partir de valores intermedios. Este comportamiento está asociado al aumento en la cantidad de operaciones internas y a una mayor actividad de conmutación dentro del módulo. A diferencia de matrices más simples, la estructura de la matriz H implica una mayor interacción entre sus elementos, lo que contribuye a un consumo energético más elevado y a una menor eficiencia en términos de potencia a medida que escala el tamaño del problema. Sin embargo cabe aclarar que el consumo es relativamente bajo dentro del chip, y es más que aceptable para una buena implementación del módulo.

Tabla 10

Consumo de recursos del módulo Matriz H

m	LUTs		Registers		Bonded IOB		BUFGCTRL	
	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp
4	18	19	16	16	20	20	1	1
5	37	36	22	22	23	23	1	1
6	89	88	27	27	28	28	1	1
7	227	281	31	31	32	32	1	1
8	303	285	35	35	36	36	1	1

Partiendo de la tabla 10 se observa que el uso de recursos, particularmente de LUTs, tiene un crecimiento significativo con respecto a *m*, pasando de valores relativamente bajos en configuraciones pequeñas a un incremento considerable en los casos de mayor tamaño. Este crecimiento no es estrictamente lineal, lo que sugiere que la complejidad del módulo aumenta de forma más acelerada debido a la cantidad de operaciones lógicas y a la forma en que estas son sintetizadas e implementadas. Adicionalmente, se observa cierta discrepancia entre los resultados de síntesis e implementación, indicando que las herramientas de diseño realizan optimizaciones y ajustes que impactan el uso final de recursos.

En cuanto a los registros, su incremento es moderado y está principalmente influenciado por la estrategia de encapsulamiento utilizada para el análisis temporal, mientras que los recursos de entrada/salida crecen de manera proporcional al tamaño de los datos. El uso de recursos de reloj permanece constante.

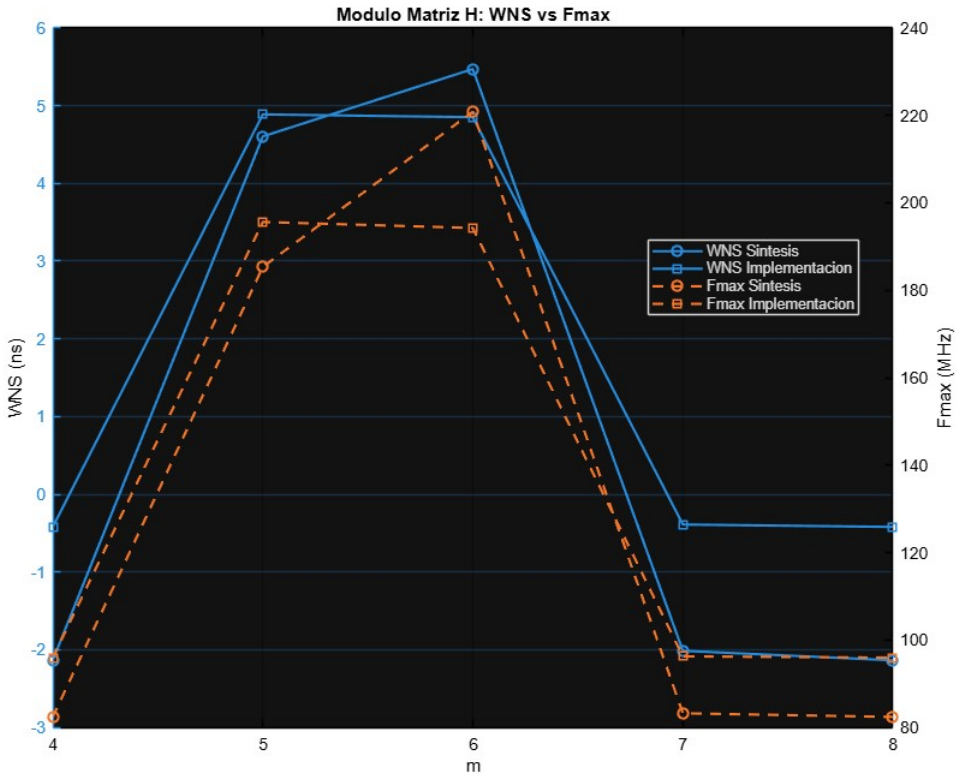
En conjunto, el módulo matriz H se posiciona como uno de los más demandantes dentro de los bloques matriciales, con implicaciones tanto en tiempo como en recursos, especialmente para valores elevados de *m*, donde su impacto en el desempeño global del sistema se vuelve más significativo.

4.1.8. Gauss Jordan

La evaluación de este módulo se restringió a un máximo de $m = 6$. La decisión responde a una limitación técnica insalvable en el flujo de diseño: para configuraciones con $m \geq 7$, la explosión en el número de Look-Up Tables (LUTs) y la congestión de enrutamiento asociada sobrepasan la capacidad de la herramienta de síntesis e implementación, provocando que el proceso de Place and Route colapse sin generar una solución válida. En consecuencia, resulta imposible obtener métricas de utilización y temporización para valores superiores de m bajo la arquitectura propuesta.

Figura 25

Frecuencia máxima de operación del módulo Gauss Jordan en función del parámetro m, comparando resultados de síntesis e implementación



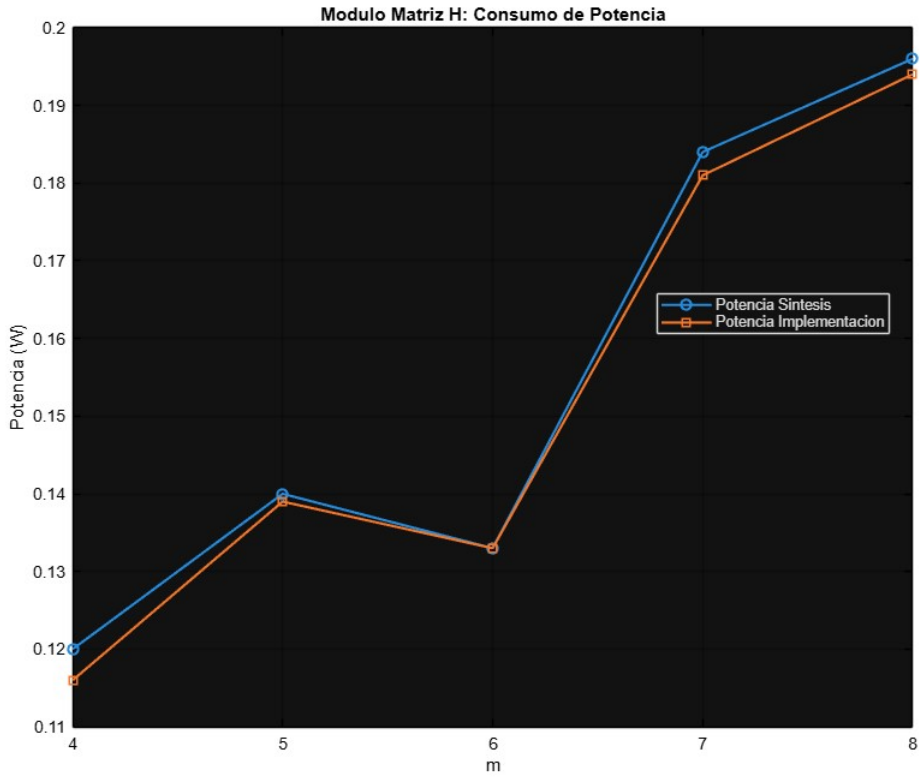
En la figura 25 se presenta el comportamiento temporal del módulo de eliminación Gauss-Jordan. Se evidencia una degradación severa y sostenida del WNS a medida que se incrementa el parámetro m. Para $m = 4$, el diseño aún presenta un margen temporal positivo en síntesis (0.209 ns)

que se torna ligeramente negativo en implementación (-0.280 ns). Sin embargo, a partir de $m = 5$, los valores de WNS caen abruptamente a rangos negativos significativos tanto en síntesis como en implementación, alcanzando un valor crítico de -7.355 ns para $m = 6$.

Este comportamiento indica que el módulo Gauss-Jordan constituye una ruta crítica extrema dentro del sistema. La caída abrupta de la frecuencia máxima de operación se atribuye a la profundidad lógica masiva generada por la naturaleza iterativa y dependiente del algoritmo, el cual, al ser implementado de forma combinacional o con lógica de control compleja no segmentada, acumula retardos de propagación prohibitivos. A diferencia de módulos aritméticos simples, aquí la lógica de decisión y las múltiples etapas de normalización y eliminación contribuyen a un camino de datos extremadamente largo.

Figura 26

Consumo de potencia total del módulo Gauss Jordan en función del parámetro m, para síntesis e implementación



En cuanto al consumo de potencia, observado en la figura 26, se registra un incremento exponencial en función de m. Pasando de 0.158 W en síntesis para $m = 4$ a 0.676 W para $m = 6$. Este aumento drástico es coherente con la explosión en la utilización de recursos lógicos mostrada en la tabla 11.

Tabla 11

Consumo de recursos del módulo Gauss Jordan

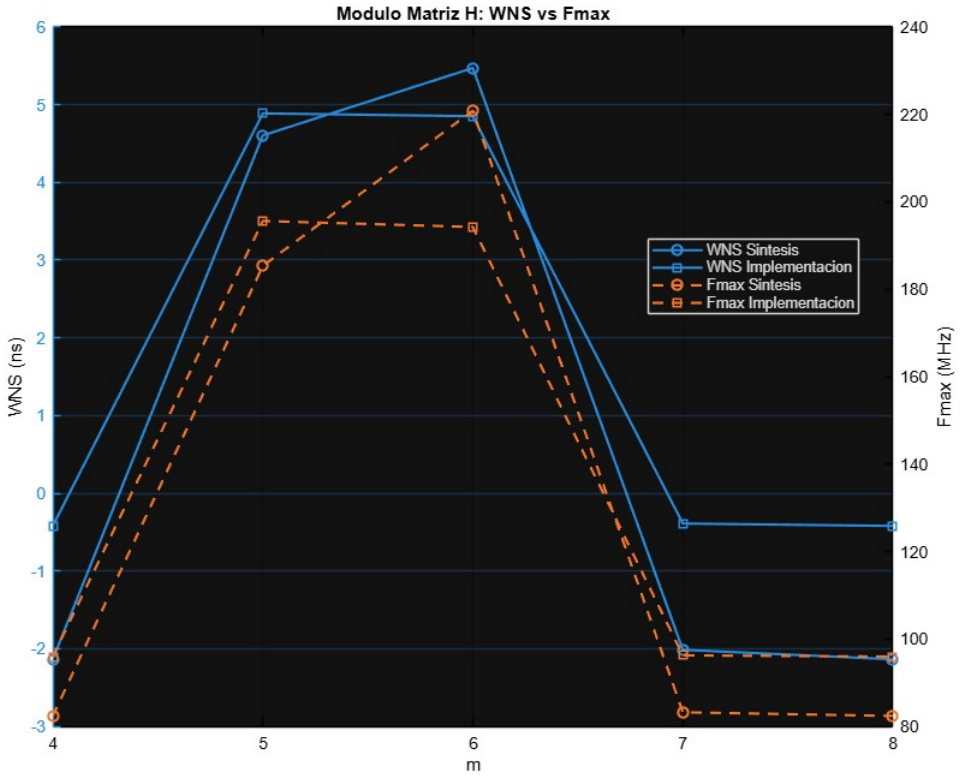
m	LUTs		Registers		F7 Muxes		F8 Muxes		Bonded IOB		BUFGCTRL	
	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp
4	2988	3185	334	334	42	42	5	5	4	4	1	1
5	15043	15774	966	1028	1373	1373	40	40	4	4	1	1
6	57153	57340	2791	2930	5958	6028	59	59	4	4	1	1

Desde el punto de vista de utilización de recursos (Tabla 11), este módulo presenta el crecimiento más agresivo de todo el sistema. El número de LUTs pasa de aproximadamente 3,000 para $m = 4$ a más de 57,000 para $m = 6$, lo que representa un incremento de casi 20 veces. La aparición y aumento de recursos dedicados como los multiplexores F7 y F8 confirma la alta complejidad del ruteo y la lógica de selección involucrada. Este comportamiento no lineal establece a Gauss-Jordan como el factor limitante principal para la escalabilidad del diseño en la FPGA seleccionada.

4.1.9. Matriz G

Figura 27

Frecuencia máxima de operación del módulo Generación de claves en función del parámetro m, comparando resultados de síntesis e implementación

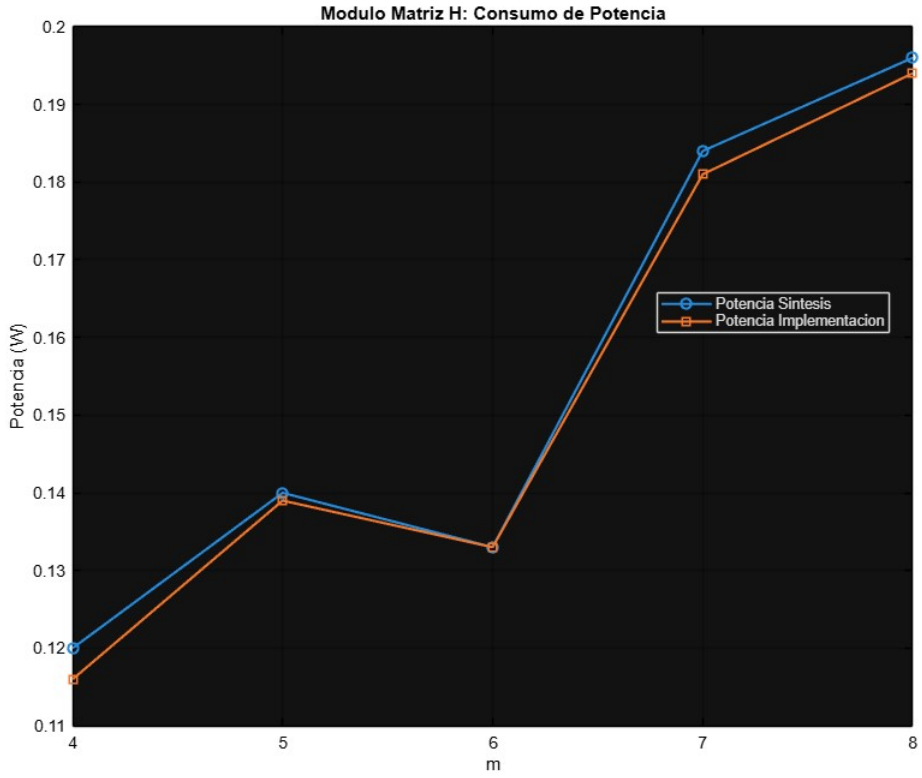


El módulo de Key Generation representa la integración total del sistema, incluyendo el módulo Gauss-Jordan, las matrices derivadas y la lógica de control asociada. Como se observa en la figura 27, el comportamiento temporal refleja y magnifica las limitaciones del submódulo Gauss-Jordan. Aunque para $m = 4$ el slack en síntesis es prácticamente neutro (-0.01 ns), para $m = 6$ el WNS en implementación se desploma hasta -21.659 ns.

Es particularmente notable la diferencia entre síntesis e implementación para $m = 6$; mientras que la síntesis estima un retardo con WNS de -7.820 ns, el proceso de Place and Route (implementación) degrada el rendimiento casi tres veces más.

Figura 28

Consumo de potencia total del módulo Generación de claves en función del parámetro m, para síntesis e implementación



En términos de potencia (Figura 28), el sistema completo alcanza un consumo en implementación de 2.409 W para $m = 6$, un valor de consumo medio para un dispositivo Artix-7. Comparando los datos con el módulo Gauss-Jordan aislado, se aprecia que el consumo incremental del resto de los módulos es casi marginal frente al consumo base del sistema de reducción matricial.

Tabla 12

Consumo de recursos del módulo Generación de claves

m	LUTs		Registers		F7 Muxes		F8 Muxes		Bonded IOB		BUFGCTRL	
	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp	Syn	Imp
4	3358	3611	544	544	47	47	6	6	4	4	1	1
5	15432	16350	1668	1735	1401	1401	45	45	4	4	1	1
6	64545	64884	4892	4959	7314	7314	692	692	4	4	1	1

Finalmente, la tabla 12 confirma la hipótesis planteada: el consumo de LUTs y registros del sistema completo tiene un crecimiento exponencial que está dominado en más de un 88 porciento por el módulo Gauss-Jordan. La diferencia restante corresponde a la lógica de control, registros de interfaz y las matrices T, V, D y H. Este dato es fundamental porque simplifica el análisis de cuellos de botella: cualquier esfuerzo de optimización del sistema completo debe concentrarse exclusivamente en la arquitectura del bloque Gauss-Jordan, ya que los demás módulos presentan una escalabilidad controlada y eficiente.

4.2. Resumen

El análisis experimental sobre la FPGA Artix-7, abarcando configuraciones de $m = 4$ a $m = 8$, arrojó los siguientes comportamientos diferenciados por grupo funcional:

Módulos Aritméticos: La multiplicación demostró ser eficiente y escalable, cumpliendo restricciones temporales en todo el rango evaluado. La división presentó degradación progresiva del slack temporal conforme aumentó m, mientras que la potenciación resultó crítica incluso para valores pequeños, con WNS negativos y un consumo elevado de recursos debido a la acumulación de retardos en su estructura combinacional.

Módulos Matriciales: Las matrices T y V exhibieron un comportamiento óptimo, con márgenes temporales positivos y consumo de recursos lineal y contenido. Las matrices D y H mostraron una escalabilidad aceptable, aunque con ligeras irregularidades en el WNS y un incremento

moderado de LUTs para $m = 7$, sin llegar a comprometer la viabilidad del sistema.

Sistema Integrado: El análisis del módulo Gauss-Jordan y del sistema completo Key Generation solo fue factible hasta $m = 6$, ya que para valores superiores la herramienta de síntesis colapsó ante la congestión de recursos. Se evidenció que el bloque Gauss-Jordan representa aproximadamente el 90 porciento del consumo total de LUTs y registros del sistema, constituyéndose como el cuello de botella determinante. Para $m = 6$, el WNS en implementación alcanzó -21.659 ns y el consumo de potencia ascendió a 2.409 W, confirmando la inviabilidad de una implementación puramente combinacional para campos finitos de mayor tamaño.

5. Conclusiones

• El desarrollo del proyecto permitió consolidar la base matemática requerida para la implementación del algoritmo en FPGA, dando lugar a un modelo de referencia en SageMath orientado a la verificación funcional del diseño. Dicho modelo constituye una herramienta valiosa tanto para la comprensión detallada del funcionamiento del algoritmo como para la evaluación preliminar de diferentes estrategias de implementación en hardware, facilitando la depuración y validación de los módulos RTL desarrollados.

• La representación de datos basada en vectores de bits de ancho m, que supone una representación bastante acertada y la arquitectura modular adoptada demostraron ser el enfoque correcto para abordar el diseño en FPGA. La separación funcional entre módulos aritméticos, matriciales y de control no solo facilitó la verificación incremental del sistema, sino que permitió una clara trazabilidad del impacto de cada bloque sobre el desempeño global. El planteamiento predominantemente combinacional para los módulos aritméticos y las matrices generadoras resultó altamente escalable. Un aspecto particularmente relevante fue la implementación de los módulos matriciales que generan sus salidas on the fly (es decir, calculan cada elemento de la matriz en el mismo ciclo de reloj en que se requiere, sin necesidad de almacenamiento previo ni memorias intermedias), lo cual evitó la necesidad de guardar la mayoría de las matrices y supuso un avance significativo en la eficiencia del diseño a nivel RTL. Los resultados evidencian que el crecimiento en consumo de recursos y potencia se mantiene lineal y controlado con el aumento de m, validando la estrategia arquitectural propuesta. Más allá de su desempeño inmediato, este enfoque combinacional constituye una guía valiosa para futuras mejoras, pues su escalabilidad natural, combinada con técnicas de segmentación como pipeline, se perfila como la ruta más adecuada para implementar eficientemente esta clase de algoritmos en hardware.

• Se logró caracterizar completamente el comportamiento del sistema en términos de frecuencia, recursos y potencia, identificando las condiciones de operación viables en la FPGA seleccionada. Los datos obtenidos proporcionan una línea base cuantitativa para evaluar futuras mejoras arquitecturales.

El trabajo realizado entrega un diseño funcional base y, más importante aún, una caracterización empírica precisa del factor limitante. Se concluye que el sistema es viable para tamaños de clave pequeños, y que la ruta hacia configuraciones mayores requiere necesariamente una reestructuración de la lógica de inversión de matrices, ya sea mediante mayor paralelismo controlado o segmentación profunda de la ruta de datos.

• El bloque de eliminación Gauss-Jordan constituye el cuello de botella determinante del sistema completo. Su implementación como máquina de estados finitos, si bien funcional para m mayor o igual a 6, colapsa las capacidades de síntesis y enrutamiento de la FPGA para tamaños de campo superiores.Este hallazgo delimita con precisión el problema a resolver: la escalabilidad del sistema depende exclusivamente de la optimización de esta etapa. El resto de los módulos presentan márgenes suficientes para operar con parámetros criptográficamente relevantes, por lo que el esfuerzo de diseño futuro debe concentrarse íntegramente en refinar la arquitectura del modulo Gauss-Jordan.

6. Recomendaciones

• Se recomienda mantener la representación de datos basada en vectores de bits y el enfoque combinacional para los módulos aritméticos y matriciales, dado su probado buen desempeño. Estos bloques pueden ser reutilizados como unidades funcionales dentro de arquitecturas más estructuradas.

• Para mejorar la frecuencia de operación del sistema, se recomienda aplicar pipeline en los módulos que presentaron los valores más negativos de WNS, particularmente en los bloques de división y potenciación. Esta técnica permitiría reducir la profundidad lógica de las rutas críticas sin modificar sustancialmente la arquitectura base.

• Se insta a replantear la arquitectura del módulo Gauss-Jordan, bien sea refinando la máquina de estados actual con técnicas más pulidas de control y reutilización de recursos, o explorando esquemas alternativos de inversión de matrices en cuerpos finitos que presenten mejor escalabilidad en hardware.

• Se propone aprovechar los módulos desarrollados como base para una arquitectura RISC-V especializada en operaciones criptográficas. Las operaciones aritméticas en cuerpo finito y los módulos de generación de matrices podrían integrarse como instrucciones personalizadas o unidades funcionales dedicadas, permitiendo que algoritmos como Gauss-Jordan se ejecuten mediante secuencias de instrucciones sobre este hardware especializado, logrando un equilibrio entre la eficiencia del diseño combinacional y la flexibilidad de un entorno programable.

• Si se desea continuar con el desarrollo de las siguientes etapas del criptosistema, tambien se realizo el diseño y descripcion en verilog del los modulos para el algoritmo de Patterson, que pueden ser consultados en el repositorio de GitHub

Referencias Bibliográficas

Chen, S., Lin, H., Huang, W., y Huang, Y. (2022). Hardware design and implementation of classic mceliece post-quantum cryptosystem based on fpga. En *2022 ieee high performance extreme computing conference (hpec)* (pp. 1–7). doi: 10.1109/HPEC55821.2022.9926345

McEliece, R. J. (1978). *A public-key cryptosystem based on algebraic coding theory* (Deep Space Network Progress Report n.^{os} 42–44). NASA Jet Propulsion Laboratory, California Institute of Technology. Descargado de `https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF`

Shoufan, A., Wink, T., Molter, G., Huss, S., y Strentzke, F. (2009). A novel processor architecture for mceliece cryptosystem and fpga platforms. En *2009 20th ieee international conference on application-specific systems, architectures and processors* (pp. 98–105). doi: 10.1109/ASAP.2009.25

Apéndice A: Repositorio en GitHub

Se ha establecido un repositorio en GitHub. con el propósito de facilitar el acceso académico y fomentar la colaboración en el proyecto, este repositorio incluye el código en SageMath como un archivo para jupyter notebook, todos los códigos de descripción de hardware en Verilog, asi como la estructuración del proyecto para Vivado, los diagramas de los módulos, las gráficas y los reportes obtenidos para el análisis de resultados. Esto promueve el aprendizaje y alienta a otros investigadores y estudiantes a contribuir y mejorar el sistema, ampliando así su impacto en la comunidad académica.