DISEÑO E IMPLEMENTACIÓN EN UN ACELERADOR HARDWARE, DE LA TRANSFORMADA TEÓRICO NUMÉRICA (NTT) DEL MECANISMO DE ENCAPSULACIÓN DE CLAVE CRYSTALS-KYBER.

JOSE ALEJANDRO ROJAS DIAZ

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2025

DISEÑO E IMPLEMENTACIÓN EN UN ACELERADOR HARDWARE, DE LA TRANSFORMADA TEÓRICO NUMÉRICA (NTT) DEL MECANISMO DE ENCAPSULACIÓN CLAVE CRYSTALS-KYBER.

JOSE ALEJANDRO ROJAS DIAZ

Trabajo de grado para optar por el título de Ingeniero Electrónico

Directores
Jaime Velasco, Ph. D Claudia Patricia Rentería, Ph. D

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2025

Nota de aceptación

_____

_____

_____

_____

_____

Firma del presidente del jurado

_____

Firma del jurado

_____

Firma del jurado

AGRADECIMIENTOS

Agradezco, en primer lugar, a Dios, por darme la vida, la fortaleza y la sabiduría necesarias para culminar esta etapa.

A mi familia, por su amor incondicional y constante apoyo. En especial a mi madre Ynelda, por su entrega y dedicación; a mi abuela Ana, por sus oraciones y ejemplo de vida; y a mi tía Adriana, por su cariño y respaldo en cada momento.

Extiendo mi agradecimiento a mis directores, Jaime Velasco Medina y Claudia Patricia Rentería Mejía, por su orientación, paciencia y valiosos aportes a lo largo del desarrollo de este trabajo.

A mis amigos, por su compañía, motivación y por hacer este camino más llevadero.

A la Universidad del Valle, en particular a la Escuela de Ingeniería Eléctrica y Electrónica, por brindarme la formación académica y los recursos necesarios para llevar a cabo este proyecto.

Y finalmente, a todos aquellos que, de alguna manera, contribuyeron a la realización de este trabajo. A todos, muchas gracias.

Índice

Índice de figuras

Índice de cuadros

RESUMEN

La computación cuántica ofrece soluciones de alta eficiencia a problemas complejos que exceden las capacidades de la computación clásica. No obstante, también representa una amenaza constante y creciente para los sistemas criptográficos actuales. Algoritmos cuánticos como el de Shor permiten la factorización eficiente de números grandes, comprometiendo la seguridad de esquemas como RSA y ECC, estos esquemas son ampliamente usados en la protección de datos y comunicaciones digitales. Esta amenaza ha impulsado el desarrollo de la criptografía post-cuántica, un conjunto de algoritmos diseñados para resistir ataques incluso con la existencia de computadoras cuánticas. Entre ellos destaca CRYSTALS-Kyber, un mecanismo de encapsulación de claves que ha sido seleccionado por el NIST como parte de los futuros estándares criptográficos.

En este contexto, la presente tesis se enfoca en el diseño e implementación en hardware de la Transformada Teórico-Numérica (NTT), la cual es una pieza fundamental en el rendimiento del algoritmo Kyber. El estudio comprende una fase de apropiación y desarrollo inicial en lenguaje C, seguida de su implementación en el lenguaje de descripción de hardware VHDL para posteriormente sintetizar e implementar en el FPGA EP4CE115F29C7N de la tarjeta DE2 115[4], con el fin de evaluar su eficiencia y viabilidad en sistemas embebidos, por último evalúa y compara el grado de optimización de los recursos y resultados con implementaciones previas.

Este trabajo busca contribuir a la construcción de soluciones criptográficas seguras frente al paradigma cuántico, aportando una implementación optimizada que pueda ser integrada en futuros sistemas de comunicación seguros. Asimismo, se consideran aspectos técnicos y éticos en la transición hacia nuevas formas de proteger la información digital.

Los resultados obtenidos evidencian que la arquitectura paralela total ofrece la menor latencia absoluta en transformadas de 128 y 256 puntos, aunque con un mayor consumo de recursos. En contraste, las arquitecturas tipo pipeline, especialmente las de 64 y 128 mariposas con reducción de Barrett, logran un equilibrio eficiente entre área utilizada y tiempo de ejecución, siendo más competitivas conforme aumenta el tamaño de la transformada. Ademas estas configuraciones alcanzan mejoras de hasta un 60.50 % en tiempo de ejecución y destacan en eficiencia área-tiempo al contrastar los resultados haciendo uso de la función mod, o haciendo uso de la reducción modular usando Barret. Por otro lado, las arquitecturas basadas en reutilización de componentes muestran un desempeño limitado, resultando inviables para aplicaciones criptográficas exigentes. La elección de la arquitectura más adecuada dependerá de las prioridades del sistema: mínima latencia, optimización de área, o un balance entre ambos factores en contextos con restricciones energéticas y de hardware.

ABSTRACT

Quantum computing offers highly efficient solutions to complex problems that exceed the capabilities of classical computing. However, it also poses a constant and growing threat to current cryptographic systems. Quantum algorithms such as Shor's enable efficient factorization of large numbers, compromising the security of widely used schemes such as RSA and ECC, which are essential for protecting digital data and communications.

This threat has driven the development of post-quantum cryptography, a set of algorithms designed to withstand attacks even in the presence of quantum computers. Among these, CRYSTALS-Kyber stands out as a key encapsulation mechanism selected by NIST as part of the upcoming cryptographic standards.

In this context, the present thesis focuses on the hardware design and implementation of the Number Theoretic Transform (NTT), which plays a crucial role in the performance of the Kyber algorithm. The study involves an initial development and understanding phase in the C language, followed by its implementation in the VHDL hardware description language. The design is then synthesized and implemented on the EP4CE115F29C7N FPGA of the DE2-115 board [4], in order to evaluate its efficiency and feasibility in embedded systems. Finally, the results are assessed and compared with previous implementations to determine optimization levels in terms of resource usage and performance.

This work aims to contribute to the development of secure cryptographic solutions against the quantum paradigm by providing an optimized implementation that can be integrated into future secure communication systems. Technical and ethical considerations are also taken into account during the transition to new methods for protecting digital information.

The results show that the fully parallel architecture achieves the lowest absolute latency for 128- and 256-point transforms, albeit with higher resource consumption. In contrast, pipeline architectures—particularly those with 64 and 128 butterflies using Barrett reduction—achieve an efficient balance between area usage and execution time, becoming more competitive as the transform size increases. Additionally, these configurations achieve up to 60.50 % improvement in execution time and stand out in area-time efficiency when comparing implementations using the standard modulus function versus modular reduction with Barrett. On the other hand, architectures based on component reuse show limited performance, rendering them unfeasible for demanding cryptographic applications. The selection of the most suitable architecture will depend on system priorities: minimum latency, area optimization, or a balanced trade-off between both in contexts with energy and hardware constraints.

1. INTRODUCCIÓN

1.1. PLANTEAMIENTO DEL PROBLEMA

En las últimas dos décadas, la evolución computacional y en el área de las comunicaciones, ha cambiado la forma y la facilidad con la que interactúan las personas y con la que acceden a cierta información; en muchos casos, la información se produce, comparte y almacena de forma digital. Como se puede suponer, este nuevo recurso representa un avance sustancial en muchas áreas, pero un desafío en otras. La seguridad en la información y la computación es la clave para una interacción segura y discreta de nuestros datos. Para garantizar esta labor existen técnicas de seguridad en los protocolos de comunicación que involucran procesos de encriptado y desencriptado de datos. De esta forma, aunque el ataque pueda burlar la seguridad primaria y acceder a los datos, estos en apariencia no tendrán un orden lógico o coherente que se pueda interpretar.

En la vanguardia de la revolución tecnológica, la computación cuántica emerge como un faro de posibilidades ilimitadas. Fundamentada en los principios de la superposición de la materia y el entrelazamiento cuántico, esta rama innovadora de la informática promete transformar la capacidad de procesamiento de la información de manera radical.

Con la evolución hacia la computación cuántica, surge un desafío significativo para la criptografía tradicional. La capacidad mejorada de trabajo de las computadoras cuánticas amenaza directamente la robustez de los algoritmos criptográficos convencionales utilizados en la actualidad. La amenaza inminente se concentra en la capacidad de algoritmos cuánticos como el de Shor para factorizar números grandes rápidamente, poniendo en riesgo sistemas criptográficos fundamentales, entre ellos RSA y ECC. Ante este escenario, la criptografía post-cuántica emerge como una necesidad apremiante. Explorar algoritmos y protocolos que pueden resistir el poder computacional de las máquinas cuánticas se convierte en una prioridad esencial. Este paradigma de seguridad post-cuántica se erige como un escudo crítico para preservar la integridad de las comunicaciones y la privacidad en la inminente era cuántica.

La amenaza cuántica se traduce de manera directa en la vulnerabilidad de los sistemas de seguridad de la información actuales, comprometiendo la privacidad y la seguridad de las comunicaciones. La pérdida de privacidad en un mundo cada vez más conectado plantea desafíos significativos para individuos, empresas y gobiernos, resaltando la importancia crítica de abordar no solo los aspectos técnicos sino también los éticos y sociales de esta transición.

La computación cuántica no solo ofrece nuevas posibilidades, sino que también presenta desafíos cruciales que deben abordarse de manera proactiva. La necesidad de adaptar la criptografía a este nuevo paradigma se vuelve cada vez más evidente, y la investigación y la implementación rápidas de soluciones post-cuánticas se erigen como un pilar esencial para salvar la seguridad de la información en la era cuántica que se avecina. De esta forma se plantea ¿Cómo implementar de forma eficiente en hardware la Transformada Teórica Numérica (NTT) del mecanismo de encapsulación de clave CRYSTALS-Kyber?

1.2. ANTECEDENTES

En este apartado, se evidencian trabajos que buscan entender y mejorar los cripto procesos post- cuánticos. Haciendo uso de diferentes arquitecturas de hardware. Con el objetivo de dificultar las intrusiones de la computación cuántica. Al revisar una pequeña lista de antecedentes encontrados destacaron 3 documentos, aunque el estándar de criptografía trabajado en el anteproyecto tiene menos de dos años en el mercado, hubo dos referencias del 2009 y 2010 que resultaban importantes.

"An Efficient Implementation of KYBER" [18]: Este artículo propone una implementación eficiente en hardware del esquema de encapsulación de claves post-cuántico CRYSTALS-KYBER, basado en el problema Module-LWE. El diseño utiliza un algoritmo de reducción modular optimizado, un sumador modular modificado y una arquitectura reconfigurable que permite compartir recursos computacionales en distintas operaciones polinomiales. Se implementó en una FPGA Xilinx Artix-7, obteniendo un diseño más compacto, con menor tiempo de ejecución y menor uso de registros respecto a trabajos similares. Este trabajo demuestra cómo se pueden optimizar los recursos en hardware sin comprometer la seguridad post-cuántica.

El documento "Design of Lattice-Based Post-Quantum Cryptoprocessors"[11]. Es una tesis doctoral de la Universidad del valle que describe de forma detallada y ordenada el proceso de diseño de un cripto procesador basado en conceptos matemáticos de celosía para asegurar la información. Como consecuencia se obtuvo (mediante el lenguaje de descripción de hardware VHDL) un procesador que contiene algoritmos de criptografía post-cuantica que asegura la información incluso con la existencia de computadoras cuánticas.

"DISEÑO DE UNA ARQUITECTURA HARDWARE PARA CALCULAR LA TRANSFORMADA TEÓRICO-NUMÉRICA (NTT)" [3]. Este siguiente documento describe el proceso de integración en una arquitectura hardware de un bloque para calcular una transformada (NTT) de una cadena de datos pseudo aleatorios. En el proceso se describe el algoritmo usado para al transformada, así como conceptos fundamentales que guardan relación con: la transformada teórico-numérica (NTT), la criptografía basada en redes, el proceso de estandarización (NIST), entre otros. La implementación del proyecto se realizó bajo el estándar criptográfico NIST y se implementaron lo siguientes algoritmos: *Barrett Modulo Multiplication, Montgomery Modulo Multiplication, Basic Look Up Table, Run Length Look Up Table, Word-Level Montgomery Reduction, Modulo m addition*. Este trabajo es el que mayor relación guarda con el proyecto en curso, puesto que busca (haciendo uso del primer borrador del estándar criptográfico) dar solución al mismo esquema criptográfico, con la diferencia de que en esta referencia la aplicación corresponde a versión de la ronda 3, así pues; presenta modelos y metodologías de desarrollo importantes para aportar al proyecto en curso; también para comparar resultados en la etapa final.

1.3. OBJETIVOS

1.3.1. Objetivo general

Implementar una arquitectura de hardware para el algoritmo de la Transformada Teórico Numérica (NTT), correspondiente al mecanismo de encapsulación de clave Crystals-Kyber.

1.3.2. Objetivos específicos

- Implementar en lenguaje C una estructura genérica para la Transformada Teórico Numérica (NTT).
- Diseñar una arquitectura hardware en VHDL para la Transformada Teórico Numérica (NTT).
- Implementar en el FPGA EP4CE115F29C7N el diseño en VHDL para la Transformada Teórico Numérica (NTT).
- Comparar con trabajos previos la implementación en hardware.

1.4. CONTRIBUCIONES DE ESTA TESIS

Las principales contribuciones alcanzadas en esta tesis son:

- Crystals-Kyber, seleccionado por el NIST como un estándarer de cifrado e intercambio de claves, incorpora una NTT cuya implementación en software se aborda en este trabajo.
- Diseño de 4 arquitecturas en hardware de la Transformada Teórico-Numérica (NTT) utilizada en Crystals-Kyber.
- Comparación de las 4 arquitecturas hardware de la NTT diseñadas en este trabajo usando dos opciones de reducción modular.
- Desarrollo en hardware de la Transformada Teórico-Numérica (NTT) utilizada en Crystals-Kyber, actualmente el único algoritmo de cifrado e intercambio de claves seleccionado por el NIST para su estandarización.

1.5. ORGANIZACIÓN DE ESTA TESIS

Esta tesis está organizada en cinco capítulos:

- Capítulo 1 – Introducción: se plantea el problema de investigación, se describen los objetivos generales y específicos, las contribuciones esperadas y se presenta una visión general del contenido del documento.

- Capítulo 2 – Fundamentación teórica: se expone el marco teórico necesario para comprender el desarrollo del trabajo. Se incluyen conceptos clave de criptografía clásica y post-cuántica, fundamentos matemáticos del algoritmo Kyber, detalles sobre la transformada teórico-numérica (NTT), y el estado del arte en implementaciones previas.
- Capítulo 3 – Diseño de la NTT de Crystals-Kyber: se detalla el proceso de diseño e implementación de la NTT. Inicialmente, se presentan versiones en software utilizando C++, y posteriormente se abordan cuatro arquitecturas distintas en hardware utilizando VHDL, haciendo uso de diferentes estrategias relacionadas con el flujo de datos, como word groups y constant geometry, así como técnicas para la reducción modular, incluyendo el algoritmo de Barrett y la función mod nativa de VHDL.
- Capítulo 4 – Pruebas y resultados: se muestran los análisis de área, latencia y frecuencia máxima de las arquitecturas implementadas. Además, se realiza una comparación entre los resultados obtenidos y trabajos previos relacionados.
- Capítulo 5 – Conclusiones: se presentan las conclusiones del trabajo realizado y se sugieren posibles líneas de investigación futura.

2. FUNDAMENTACIÓN TEÓRICA

Este capítulo presenta una breve revisión de los principales desarrollos relacionados con la evolución de la criptografía, así como también avances recientes en el marco de la criptografía post-cuántica, especialmente en respuesta a los desafíos que plantea la computación cuántica. Esta contextualización tiene como objetivo permitirle al lector entender la necesidad de nuevos esquemas criptográficos y la relevancia del algoritmo CRYSTALS-Kyber.

2.1. EVOLUCIÓN DE LA CRIPTOGRAFÍA PRE-CUÁNTICA

La criptografía desde sus orígenes ha estado enfocada en garantizar la confidencialidad de la información. Inicialmente nace para ser usada en contextos militares y diplomáticos, y se valía de técnicas como la sustitución o la transposición; luego, con el desarrollo de la informática, surgieron los sistemas criptográficos modernos, entre ellos rsa y ecc, los cuales son más complejos y se fundamentan en problemas matemáticos como la factorización y el logaritmo discreto. Conocemos entonces a la criptografía pre-cuántica a la que utilizamos hoy en día y que consideramos segura en tanto no sea viable el uso de un computador cuántico.

Estos algoritmos, ampliamente adoptados en sistemas digitales, han sido fundamentales para asegurar comunicaciones, datos bancarios y transacciones electrónicas. Sin embargo, su seguridad depende de la dificultad computacional de dichos problemas, lo que ha generado preocupación ante la aparición de nuevas tecnologías capaces de resolverlos de forma eficiente.

2.1.1. Criptografía simétrica

Se le llama criptografía simétrica al sistema de encriptación que usa la misma clave para cifrar y descifrar el texto plano. Es importante resaltar que esta clave en particular fue acordada (en secreto y previamente) por el emisor y el receptor.

Uno de los principales desafíos de la criptografía simétrica es el intercambio seguro de la clave, ya que si un tercero logra interceptarla, toda la comunicación queda comprometida. A pesar de ello, este tipo de criptografía sigue siendo ampliamente utilizada debido a su rapidez y eficiencia computacional. Algoritmos como el DES, Triple DES y especialmente el AES (Advanced Encryption Standard) han sido estandarizados y aplicados en múltiples sistemas, desde dispositivos móviles hasta comunicaciones bancarias y redes privadas.

2.1.2. Criptografía asimétrica

A diferencia de la simétrica, esta no usa la misma clave para cifrar y descifrar. En este caso se trabaja con dos claves, una pública y otra privada. La clave pública puede compartirse libremente, mientras que la privada debe mantenerse en secreto.

Por ejemplo, si una persona quiere enviarte un mensaje, puede usar tu clave pública para cifrar. Luego, solo tú, con tu clave privada, podrás leerlo. Esto facilita la comunicación segura sin que ambas partes tengan que reunirse antes para acordar una clave secreta.

Uno de los algoritmos más conocidos que usa este enfoque es RSA, el cual fue uno de los primeros en usarse de forma práctica. Aunque este es más lento que los métodos simétricos, soluciona el problema de compartir claves de forma segura, algo que es clave cuando se trabaja en redes abiertas como Internet.

2.2. CRIPTOGRAFÍA POST-CUÁNTICA

La criptografía post cuántica, nace para enfrentar al creciente y potencial impacto de la computación cuántica. Algoritmos como el Shor (usado en la computación cuántica) facilitan la resolución eficiente de problemas matemáticos comunes en sistemas criptográficos en la actualidad. Esto representa un gran logro pero en la misma medida una amenaza para la ciberseguridad. Debido a esto, diversas instituciones, como el Instituto Nacional de Estándares y Tecnología (NIST), han trabajado en la creación de algoritmos que resistan a ataques realizados desde la computación cuántica. Entre los candidatos seleccionados se encuentra CRYSTALS-Kyber, un esquema de encapsulación de claves basado en el problema de aprendizaje con errores (LWE).

Esta criptográfica emplea estructuras algebraicas y métodos diferentes entre los que se encuentran el método de retículas, corrección de errores o isogénicas, en reemplazo de los clásicos problemas de factorización o logaritmos, mencionados anteriormente.

2.3. ESTÁNDAR NIST PARA CRIPTOGRAFÍA POST-CUÁNTICA

El Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés) es una agencia del gobierno de los Estados Unidos que se encarga de establecer normas y directrices para diversas áreas, incluyendo la seguridad informática. Aunque fue creado hace varias décadas, su rol en la criptografía tomó fuerza en los últimos años. Inicialmente, sus estándares no eran de uso obligatorio, sino más bien recomendaciones para sectores específicos como la industria o la academia. Sin embargo, con el tiempo, se volvieron normativos para todas las agencias gubernamentales de EE. UU., aunque en el sector privado su adopción sigue siendo opcional.

En respuesta al creciente desarrollo de la computación cuántica y su potencial para romper los algoritmos criptográficos tradicionales, el NIST lanzó una convocatoria en 2016 para seleccionar y estandarizar nuevos algoritmos resistentes a este tipo de amenazas. El proceso incluyó la evaluación de decenas de propuestas por parte de expertos de todo el mundo. La preocupación se basa en que una computadora cuántica suficientemente avanzada podría resolver, en poco tiempo, problemas matemáticos que hoy requieren miles de años con los equipos actuales, comprometiendo así la seguridad de los sistemas de clave pública

como RSA o ECC[15].

Esta iniciativa busca garantizar que la criptografía futura sea segura tanto frente a computadores clásicos como cuánticos. Entre los algoritmos destacados se encuentra CRYSTALS-Kyber, que ha sido seleccionado como uno de los nuevos estándares para la encapsulación de claves. El objetivo del NIST con este esfuerzo es que, cuando llegue el momento en que las computadoras cuánticas sean una realidad práctica, ya exista una infraestructura de seguridad lista para enfrentar ese nuevo escenario.

El marco de ciberseguridad del NIST también se representa visualmente a través de un esquema que agrupa sus funciones clave: identificar, proteger, detectar, responder y recuperar. Estas funciones reflejan el ciclo completo de gestión del riesgo en ciberseguridad. Además, el componente central de "gobernar" (govern) actúa como eje transversal, controlando y asegurando que todas las acciones estén alineadas con políticas y objetivos específicos. Este enfoque integral permite a las organizaciones anticiparse, reaccionar y adaptarse de forma más eficiente ante amenazas y vulnerabilidades digitales.

Figura 1: Funciones clave del NIST (2025) [1]

2.4. MECANISMO DE ENCAPSULACIÓN DE CLAVE CRYSTALS-KYBER

El mecanismo de encapsulación de claves CRYSTALS-Kyber es un esquema criptográfico diseñado para ser seguro ante ataques de computadoras cuánticas. Forma parte de los algoritmos seleccionados por el NIST en su proceso de estandarización de criptografía post-cuántica. Su función principal es permitir que dos partes establezcan una clave secreta compartida, incluso si se comunican por un canal inseguro. Lo interesante es que esto se logra sin enviar directamente la clave secreta, sino mediante una técnica llamada encapsulación, en la que una clave pública permite cifrar una "clave encapsulada" que solo puede recuperar

quien tenga la clave privada correspondiente[1].

CRYSTALS-Kyber se basa en el problema de aprendizaje con errores (LWE, por sus siglas en inglés), específicamente en una variante conocida como Module-LWE. Este problema es considerado resistente incluso frente a ataques cuánticos. Con el fin de optimizar la eficiencia, Kyber utiliza una versión estructurada de LWE que permite operar con polinomios, y es aquí donde entra la Transformada Teórica de Números (NTT, por sus siglas en inglés).

La NTT permite acelerar las operaciones con polinomios, así como la multiplicación de estos, que son parte fundamental de los cálculos dentro del esquema. La clave aquí es que, en lugar de realizar las multiplicaciones directamente en el dominio de los coeficientes, la NTT transforma estos polinomios a un dominio diferente, donde las multiplicaciones se realizan de manera más eficiente. Luego, al aplicar la NTT inversa, se regresa al dominio original, obteniendo el resultado de la multiplicación de manera mucho más rápida que si se hiciera de forma directa. Esta optimización es crucial en Kyber, pues permite realizar las operaciones necesarias sin comprometer la seguridad del sistema.

Gracias a la NTT, Kyber puede mantener una alta seguridad sin sacrificar rendimiento, algo clave para su implementación práctica en dispositivos reales. Además, Kyber tiene propiedades útiles como la simplicidad en la implementación y la posibilidad de usar tamaños de clave razonables para un entorno moderno. Todo esto lo convierte en una de las propuestas más serias y prácticas dentro del campo de la criptografía post-cuántica.

2.5. FUNDAMENTOS MATEMÁTICOS DE KYBER: LWE Y NTT

El esquema CRYSTALS-Kyber se basa en una variante estructurada del problema de aprendizaje con errores (Module-LWE) y en la aceleración de sus operaciones polinómicas mediante la Transformada Teórico-Numérica (NTT). A continuación se describen los elementos clave de estos fundamentos[1].

2.5.1. *Learning With Errors (LWE)* y *Module-LWE*

El problema LWE consiste en recuperar un vector secreto s, dado un conjunto de muestras (a_i, b_i) donde:

$$b_i = \langle a_i, s \rangle + e_i \mod q,$$

y e_i es un "error" pequeño elegido según una distribución discreta. En su variante *Module-LWE*, en lugar de trabajar con vectores del espacio $\mathbb{Z}_q^m$, se emplean módulos de polinomios sobre el anillo:

$$\mathbb{Z}_q[x]/(x^n + 1),$$

lo que aporta estructura algebraica y reduce los tamaños de clave. El secreto y el error se codifican como vectores de polinomios de grado menor que n, con coeficientes en $\mathbb{Z}_q$ que siguen una distribución de baja entropía. Esta versión es

resistente a ataques cuánticos, ya que resolverla implica romper la estructura de retículas en dimensiones muy altas, tarea considerada intratable incluso para ordenadores cuánticos.

2.5.2. Representación polinómica

Tanto el secreto como la matriz pública y el mensaje aleatorio se representan como polinomios:

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}, \quad a_i \in \mathbb{Z}_q,$$

en el anillo $\mathbb{Z}_q[x]/(x^n + 1)$. Las operaciones de encapsulación y decapsulación implican esencialmente multiplicaciones y sumas de estos polinomios, siempre reducidas módulo $x^n + 1$ y módulo q.

2.5.3. Transformada Teórico-Numérica (NTT)

Multiplicar dos polinomios de forma directa requiere $\mathcal{O}(n^2)$ multiplicaciones de coeficientes. La NTT reduce esta complejidad a $\mathcal{O}(n \log n)$ al aprovechar el teorema de la convolución en el contexto modular. El procedimiento es:

- Elegir un generador g de $\mathbb{Z}_q^*$ (una raíz primitiva de orden $q - 1$).
- Calcular la raíz n-ésima de la unidad:

  $$\omega = g^{(q-1)/n} \mod q,$$

  que satisface $\omega^n \equiv 1 \mod q$ y $\omega^k \not\equiv 1$ para $0 < k < n$.
- Transformar los coeficientes del polinomio mediante evaluación en $\{1, \omega, \omega^2, \ldots, \omega^{n-1}\}$.
- Multiplicar punto a punto los resultados en ese dominio.
- Aplicar la NTT inversa para regresar al dominio de coeficientes.

Así, la NTT convierte la convolución de coeficientes (multiplicación de polinomios) en una multiplicación punto a punto mucho más eficiente, y luego reconstruye el producto mediante la transformada inversa.

2.5.4. Integración de LWE y NTT en Kyber

En **CRYSTALS-Kyber**, la NTT se emplea en las tres fases principales:

- **Generación de claves**, para multiplicar el vector secreto por la matriz pública;
- **Encapsulación**, al cifrar el mensaje codificado como polinomio con la clave pública;
- **Decapsulación**, al recuperar la clave compartida usando el secreto.

Gracias a la NTT, Kyber logra un balance óptimo entre seguridad postcuántica y rendimiento en hardware, manteniendo tiempos de ejecución bajos y tamaños de clave moderados, lo que facilita su adopción en entornos empotrados y de alto rendimiento.

2.5.5. La estructura matemática de una NTT simple

La NTT es una herramienta eficiente para realizar transformaciones de Fourier en cuerpos finitos. En el contexto de ML-KEM, se utiliza para realizar operaciones polinomiales de manera eficiente. En ML-KEM, se definen dos parámetros clave: q y n. q es el número primo 3329, que es $28 \cdot 13 + 1$, y n es 256. Existen 128 raíces primitivas de la unidad de 256 en $\mathbb{Z}_q$, y $\zeta = 17 \in \mathbb{Z}_q$ es una de ellas [**?**].

La función BitRev7(i) se utiliza para invertir los bits de un entero de entrada i en el rango de 0, ...,127. El polinomio $X^{256} + 1$ se puede factorizar en 128 polinomios de grado 2 modulo q de la siguiente manera:

$$X^{256} + 1 = \prod_{k=0}^{127} (X^2 - \zeta^{2 \cdot Bitrev7(k)} + 1) \tag{1}$$

Por lo tanto, el anillo Rq: = Zq[X]/ $(X^{256} + 1)$ es isomorfo a una suma directa de 128 campos de extensión cuadrática de Zq, denotados Tq. La representación NTT de un polinomio $f \in Rq$ es el vector que consta de los residuos correspondientes de grado uno:

$$\hat{f} = fmod(X^2 - \zeta^{2 \cdot Bitrev7(k)} + 1), \ldots, (X^2 - \zeta^{2 \cdot Bitrev7(127)} + 1) \tag{2}$$

En el algoritmo, f se almacena como una matriz de 256 enteros modulo q.

$$f \text{ mód } \left(X^2 - \zeta^{2\text{BitRev7}(i)} + 1\right) \; = \; \hat{f}[2i] \; + \; \hat{f}[2i+1]\, X \tag{3}$$

2.5.6. NTT basada en la Convolución Cíclica

La NTT basada en la convolución cíclica (CC-NTT) es importante para el enfoque del proyecto. La NTT CC de n puntos tiene dos parámetros clave: la longitud (o el punto) n y el módulo q. Cuyos detalles son los siguientes [**?**]:

- n: Es una potencia de dos.
- q: Es un número primo que cumple con $q \equiv 1 \pmod{n}$.

Esto implica que existe una raíz primitiva n-ésima de la unidad, denotada como ω_n, en el conjunto $\mathbb{Z}_q$.

La transformación directa, denotada como NTT, se define como:

$$a_j = \sum_{i=0}^{n-1} a_i \left(\omega_n^{ij}\right) \text{ mód } q, \quad j = 0, 1, \ldots, n-1. \tag{4}$$

La transformación inversa, denotada como INTT, se define como:

$$a_i = \sum_{j=0}^{n-1} a_j \left(\omega_n^{-ij}\right) \text{ mód } q, \quad i = 0, 1, \ldots, n-1. \tag{5}$$

Las propiedades más relevantes de estas transformadas son las siguientes:

- Siempre se cumple que $a = \text{INTT}(\text{NTT}(a))$.
- La NTT y la DFT comparten fórmulas y propiedades similares, pero la NTT utiliza raíces primitivas enteras de la unidad, mientras que la transformada discreta de Fourier, denotada como DFT, utiliza factores de giro complejos.

2.6. TIPOS DE NTT

La Transformada Número-Teórica (NTT) es una herramienta fundamental para acelerar operaciones de convolución polinómica en aritmética modular, siendo una pieza clave en esquemas criptográficos modernos como CRYSTALS-Kyber. Existen diversos tipos de NTT que no sólo responden a variaciones en la forma de cálculo, sino también a criterios de eficiencia, arquitectura de implementación, y requisitos del dominio algebraico donde se aplican. Estas variantes emergen principalmente al considerar dos aspectos clave: la estructura del algoritmo (como la forma en que los datos se dividen y combinan en cada etapa) y la estrategia de implementación (como la disposición iterativa, recursiva o paralela). Dependiendo del contexto, una u otra combinación puede ofrecer ventajas significativas en términos de latencia, área ocupada en hardware, complejidad de control y consumo energético.

A grandes rasgos, los tipos de NTT pueden clasificarse según el radix o base utilizada para segmentar y procesar los datos —por ejemplo, radix-2, radix-4 o radix-2^k— lo que define cuántos elementos se transforman simultáneamente en cada etapa. Asimismo, desde el punto de vista arquitectónico, pueden distinguirse implementaciones iterativas que optimizan la reutilización de recursos, recursivas que siguen un enfoque divide y vencerás, o paralelas que maximizan el rendimiento al costo de una mayor ocupación de recursos. Adicionalmente, ciertos esquemas incorporan optimizaciones como la aritmética de Montgomery para eficientar las operaciones modulares, o utilizan variantes como la NTT in-place para reducir el uso de memoria.

Así pues, el término "NTT"no hace referencia a un único algoritmo, sino a una familia de métodos y estrategias que comparten un fundamento algebraico común, pero que se adaptan según distintas necesidades de implementación y el entorno en que se apliquen, ya sea en software de propósito general o en hardware priorizando su optimización.

2.6.1. CLASIFICACIÓN SEGÚN SU FORMA DE DIVIDIR LOS DATOS

En la NTT, el término "radix" define el tamaño de cada "mariposa" o unidad de computación dentro del algoritmo: es decir, cuántos coeficientes se procesan simultáneamente en cada paso. Esta división condiciona tanto la cantidad de etapas necesarias como la complejidad interna de cada una. En un esquema *radix-2*, cada mariposa toma un par de valores, multiplica uno de ellos por un único factor de raíz y realiza una suma y una resta; sin embargo, en un esquema *radix-4* esa misma unidad de cómputo eleva el número de entradas a cuatro, aplica tres factores diferentes y combina los cuatro valores para producir cuatro salidas en una sola operación.

La elección de *radix-2* suele considerarse la más sencilla de implementar, dado que cada mariposa es casi idéntica a la operación básica de suma/resta modular con un solo "twiddle factor". Esto facilita el control de flujo y el reuso de hardware, pero a costa de un mayor número de etapas — exactamente $\log_2(n)$ rondas de mariposas para un NTT de tamaño n. Por el contrario, usar *radix-4* reduce la profundidad del algoritmo pues cada etapa consume bloques de cuatro coeficientes de una sola vez. El *trade-off* (desventaja o costo de la elección) es que la lógica interna de cada mariposa es más compleja: tres multiplicaciones modulares simultáneas y un esquema de sumas/restas que mezcla los cuatro valores entre sí.

Más allá de estos dos casos, existen variantes $radix{-}2^k$ que extienden la idea: en radix-8 o radix-16, por ejemplo, cada mariposa manejará ocho o dieciséis coeficientes respectivamente. Estas aproximaciones pueden ofrecer latencias aún menores (más reducción de etapas), pero el coste en lógica y en número de multiplicaciones por mariposa crece rápidamente. En hardware, la decisión de qué radix emplear depende de la disponibilidad de recursos (cantidad de multiplicadores modulares, ancho de bus de memoria, registros) y del objetivo de diseño.

2.6.2. CLASIFICACIÓN SEGÚN SU TIPO DE IMPLEMENTACIÓN

La implementación de la NTT no se limita a la elección del radix, sino que también puede clasificarse según la estrategia arquitectónica adoptada para ejecutarla. Estas variantes se relacionan con la forma en que se organiza el flujo de datos, la reutilización de recursos, y la manera en que se aplica la aritmética modular.

Una primera distinción se puede hacer entre implementaciones iterativas y recursivas. En las versiones iterativas, cada etapa de la transformada se ejecuta secuencialmente usando bucles explícitos, lo que facilita la reutilización de hardware y el control eficiente del acceso a memoria. Las versiones recursivas, por otro lado, dividen el problema en subproblemas más pequeños y los resuelven mediante llamadas recursivas (en software), lo que puede ser ventajoso en términos de legibilidad o si se aplica en entornos con stack optimizado, aunque

en hardware es menos común.

También es común distinguir entre implementaciones in-place y out-of-place. Una NTT in-place sobrescribe los datos originales con sus resultados etapa por etapa, reduciendo el uso de memoria. Las versiones out-of-place almacenan los resultados en una ubicación separada, lo que permite mayor simplicidad en el manejo de índices, pero requiere el doble de espacio.

Otra dimensión importante es si la implementación es serial o paralela. Las implementaciones seriales procesan una mariposa a la vez, ocupando poco hardware, pero requieren muchos ciclos de reloj. Las paralelas, en cambio, ejecutan múltiples mariposas simultáneamente, reduciendo el tiempo de ejecución a costa de mayor consumo de área y recursos lógicos.

Por último, muchas implementaciones modernas incorporan optimizaciones como la reducción de Barrett o Montgomery, que permiten realizar multiplicaciones modulares de forma más eficiente evitando divisiones costosas. Estas técnicas son especialmente útiles en hardware y se encuentran en bibliotecas criptográficas de alto rendimiento.

2.6.3. CLASIFICACIÓN SEGÚN SU FLUJO DE DATOS EN LA IMPLEMENTACIÓN

En esta sección se presentan diferentes formas de organizar el flujo de datos en la implementación de la NTT, particularmente en arquitecturas basadas en radix-2. Estas estrategias no modifican la estructura matemática del algoritmo, sino que optimizan su ejecución desde el punto de vista computacional. El objetivo es reducir el número de operaciones, minimizar accesos a memoria y mejorar la eficiencia paralela del sistema. Para ello, se describen dos enfoques principales: el procesamiento mediante grupos de palabras (Word Groups) y el uso de una estructura de geometría constante (Constant Geometry). Ambos métodos permiten una implementación más eficiente del algoritmo en plataformas tanto de hardware como de software, especialmente en entornos con procesamiento paralelo como los GPUs.

FFT Processing using Word Groups **(Flujo por Grupos de Palabras)**

En la implementación radix-2 de la NTT, el enfoque de word groups organiza los datos en bloques llamados grupos de palabras, que son procesados de forma conjunta. Aunque el tamaño del grupo de palabras (por ejemplo, 2 o 4) permanece constante a lo largo de todas las etapas, en este caso, con un grupo de palabras de tamaño 2, el número de conexiones entre los datos se reduce a la mitad en cada etapa de la FFT. En lugar de operar cada dato de manera individual, se cargan varios datos del mismo grupo desde memoria, se procesan mediante mariposas y se escriben nuevamente, todo en un solo ciclo. Esta técnica reduce significativamente el número de accesos a memoria, lo cual es fundamental en arquitecturas con alta latencia como las GPUs. El tamaño del grupo de palabras determina cuántas etapas de la FFT se procesan por iteración (definido como un "epoch"). A mayor tamaño del grupo, menor será la cantidad

de sincronizaciones necesarias entre hilos. Esta forma de flujo de datos aprovecha el paralelismo natural del algoritmo y mejora el rendimiento al adaptar la organización del cómputo a las características del hardware.

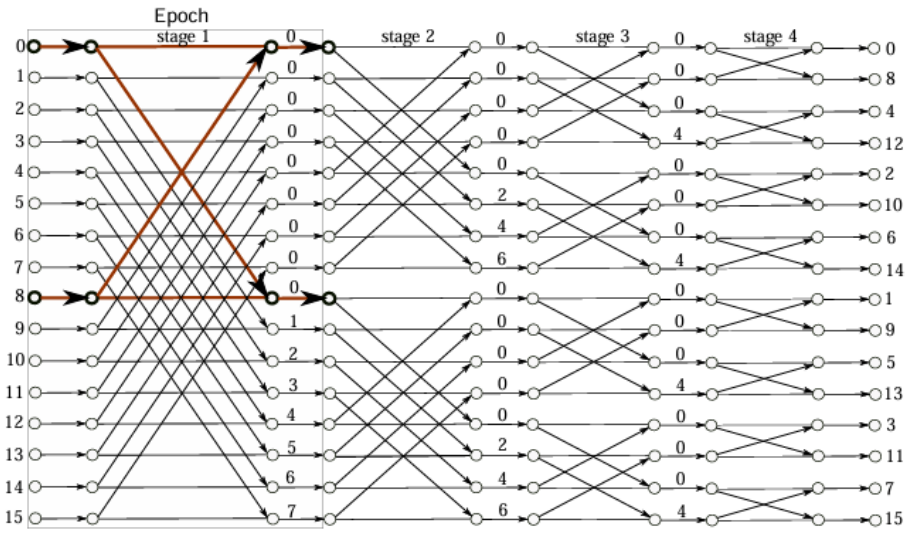


Figura 2: FFT DIF de base 2 de 16 puntos que usa grupos de 2 palabras. Extraído de *New Radix-2 and Radix-2^2 Constant Geometry Fast Fourier Transform Algorithms for GPUs* [2].

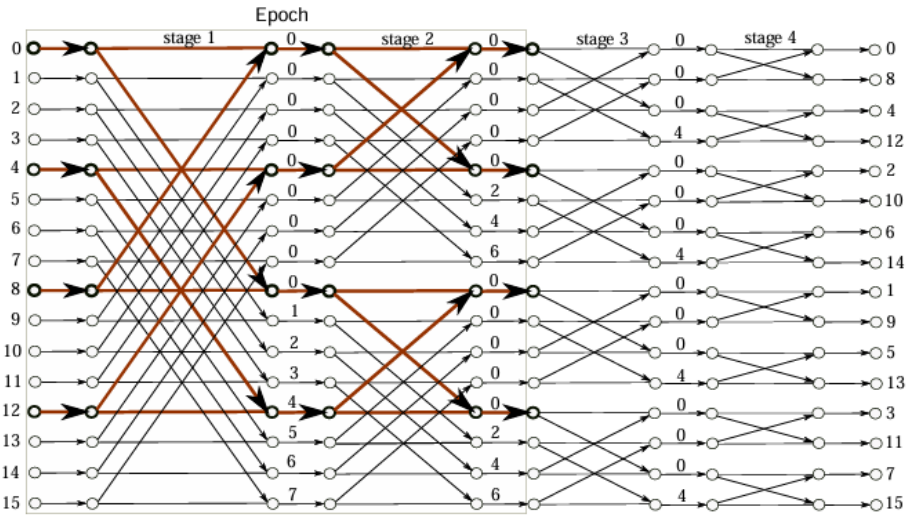


Figura 3: FFT DIF de base 2 de 16 puntos que usa grupos 4 palabras. Extraído de *New Radix-2 and Radix-2^2 Constant Geometry Fast Fourier Transform Algorithms for GPUs* [2].

Constant Geometry FFT (Flujo de Geometría Constante)

El flujo de datos basado en geometría constante busca mantener un patrón fijo en la forma en que se combinan los datos a lo largo de todas las etapas de la FFT radix-2. A diferencia de la FFT clásica, donde los pares de datos varían en cada etapa (por ejemplo, índices que difieren en N/2, N/4, etc.), en la geometría constante las combinaciones se realizan siguiendo siempre el mismo patrón estructural. Esto simplifica el cálculo de los índices de acceso a memoria, ya que estos índices se pueden calcular una sola vez al inicio del algoritmo, y no es necesario recalcularlos en cada etapa. Al mantener los mismos índices a lo largo de todo el proceso, se reduce la complejidad computacional y mejora la eficiencia del acceso a memoria, lo que es clave para implementaciones en arquitecturas paralelas como GPUs. Esta optimización facilita la paralelización y reduce la sobrecarga asociada a la gestión de índices, haciendo que este enfoque sea ideal para implementaciones de alto rendimiento tanto en hardware como en software.

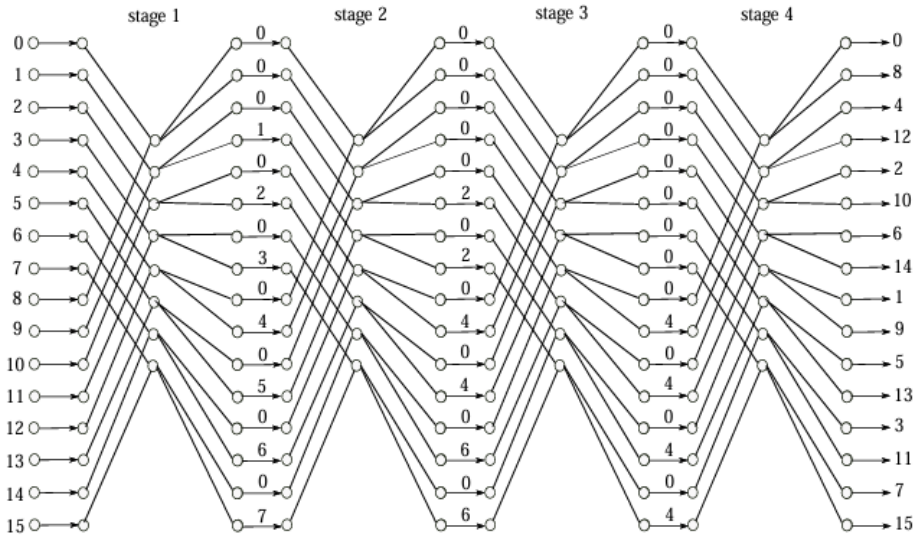


Figura 4: FFT de geometría constante DIF de base 2 de 16 puntos.[2].

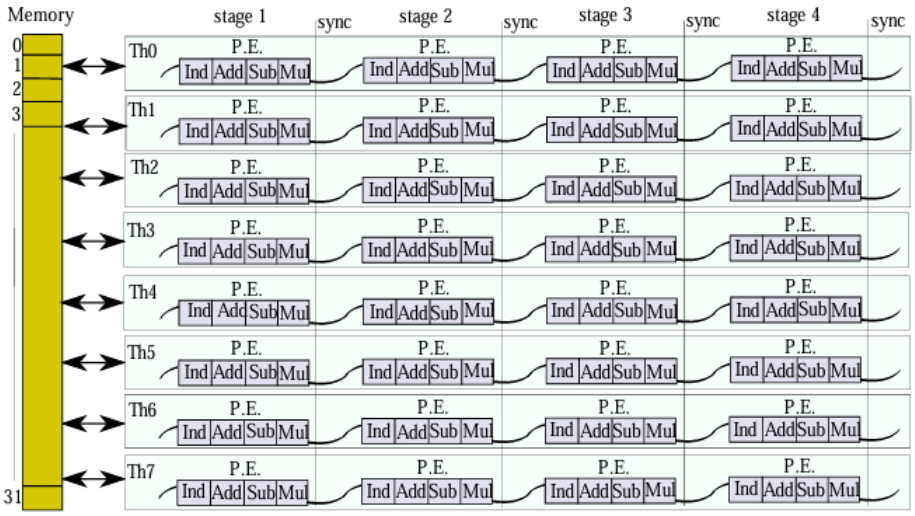


Figura 5: Implementación en paralelo de una FFT en GPUs[2].

Comparativa de Flujos en Implementación Radix-2

Aspecto	Word Groups	Constant Geometry FFT
Unidad de operación	Grupo de varios datos procesados en conjunto	Mariposas individuales con patrón estructural constante
Acceso a memoria	Menos accesos totales al procesar varios datos a la vez	Índices fijos; se calculan una sola vez
Organización del flujo	Flujo por etapas según tamaño del grupo (epoch)	Patrón uniforme en todas las etapas
Cálculo de índices	Requiere actualización entre etapas	Constantes para todas las etapas
Sincronización de hilos	Se reduce al usar grupos más grandes	Requiere por etapa, pero es predecible
Beneficio principal	Optimización de memoria y reducción de sincronización	Simplificación de control y mejor paralelismo de hilos
Orientado a	Hardware con latencia de memoria (ej. caches o DRAM)	Hardware paralelo con control flexible (ej. GPUs, SIMD)

Tabla 1: Word Groups vs Constant Geometry

2.7. ESTADO DEL ARTE

En su trabajo, Insuasty[3] diseñó e implementó una arquitectura hardware optimizada para la Transformada Teórica Numérica (NTT), empleando como base los parámetros del esquema criptográfico *Crystals-Kyber*. Este enfoque se centró en la selección de un bloque de reducción modular eficiente, esencial para operaciones en aritmética modular intensiva como las presentes en la NTT[3].

Para lograrlo, se evaluaron cinco algoritmos de reducción modular, destacándose los de *Barrett* y *Montgomery*, seleccionados finalmente por su balance entre rendimiento y consumo de recursos. Ambos algoritmos fueron implementados en descripciones comportamentales y estructurales en VHDL, y evaluados con y sin técnicas de *pipelining*.

Comparativa de Rendimiento Hardware

Los resultados de la implementación en FPGA revelaron datos relevantes:

- **Barrett** con *pipeline* logró incrementar su frecuencia de 57,59 MHz a 182,85 MHz, reduciendo su tiempo de ciclo de aproximadamente 17,37 ns a 5,47 ns.
- **Montgomery** con *pipeline* pasó de 48,66 MHz a 120,44 MHz, con una mejora similar en tiempo de ejecución.
- En términos de recursos de área, **Barrett estructural con *pipeline*** fue el que menor uso de elementos lógicos y funciones combinacionales presentó, convirtiéndolo en el más eficiente globalmente.
- En contraste, el algoritmo **Word-Level Montgomery**, aunque alcanzó la mayor velocidad (de 25,47 ns a 4 ns al incrementar su frecuencia 6,25 veces), consumió la mayor cantidad de recursos lógicos (139 LUTs).

Este análisis concluyó que el algoritmo de **Barrett** en su versión estructural con *pipeline* ofrece la mejor relación entre velocidad y uso de recursos, motivo por el cual fue seleccionado como núcleo para la implementación de la NTT de 8, 16, 128 y 256 puntos.

La arquitectura general del sistema NTT se estructuró en bloques paralelos, utilizando como componentes principales el *multiplicador Barrett*, un *sumador modular* y un *restador*, replicados de forma eficiente para escalar a transformadas de mayor tamaño.

Una contribución relevante en el campo de la criptografía post-cuántica es la tesis de Rentería-Mejía (2018), la cual se enfoca en la implementación en hardware de esquemas basados en redes, específicamente utilizando el problema Ring-LWE. Su trabajo aborda tanto cifrado de clave pública como cifrado basado en identidad, proponiendo arquitecturas eficientes orientadas a sistemas embebidos seguros[11].

En términos de enfoque, la tesis se distingue por desarrollar algoritmos optimizados para operaciones en anillos polinomiales y por su énfasis en diseños resistentes a ataques por canal lateral, haciendo uso de núcleos NTT y muestreadores gaussianos implementados en hardware.

Contribuciones técnicas clave

Entre los principales aportes técnicos del trabajo se destacan:

- El desarrollo de algoritmos NTT eficientes, incluyendo versiones radix-2 y radix-8, orientados a multiplicación polinomial de alto rendimiento.
- La implementación de muestreadores gaussianos en hardware con generación en tiempo constante, diseñados para resistir ataques temporales.
- El diseño de generadores de números aleatorios híbridos (verdaderos y determinísticos) y su integración en criptoprocesadores.
- La propuesta de arquitecturas paralelas (como arreglos sistólicos y estructuras tipo delay-commutator) para mejorar el rendimiento de los criptoprocesadores Ring-LWE en escenarios de clave pública e identidad.

Los resultados obtenidos demuestran que los criptoprocesadores diseñados alcanzan altos niveles de rendimiento, con velocidades de procesamiento en el orden de microsegundos para cifrado y descifrado, y con un uso eficiente de recursos en FPGA. Las arquitecturas propuestas —basadas en NTTs paralelas y muestreadores gaussianos de tiempo fijo— no solo superan en rendimiento a trabajos anteriores, sino que también ofrecen resistencia contra ataques temporales, validando su viabilidad para aplicaciones prácticas de criptografía post-cuántica tanto de clave pública como basada en identidad.

En su trabajo publicado en 1997, Peter W. Shor presentó algoritmos cuánticos que permiten resolver en tiempo polinomial dos problemas fundamentales en criptografía: la factorización de enteros y el cálculo del logaritmo discreto. Estos problemas constituyen la base de la seguridad de los sistemas criptográficos más

utilizados, como RSA, Diffie-Hellman y ECC. El algoritmo de Shor demuestra que una computadora cuántica suficientemente grande puede romper estos sistemas al encontrar factores primos o logaritmos discretos de manera eficiente, lo cual es impracticable para los algoritmos clásicos actuales[12].

El enfoque del algoritmo se basa en la reducción de ambos problemas a la determinación del período de una función, tarea que puede resolverse con alta eficiencia mediante una técnica cuántica conocida como la Transformada de Fourier Cuántica (Quantum Fourier Transform, QFT). Este resultado no solo marcó un hito en el campo de la computación cuántica, sino que también motivó el desarrollo activo de la criptografía post-cuántica, al evidenciar que los fundamentos de la criptografía clásica podrían no ser seguros frente a futuras amenazas tecnológicas.

En *A Decade of Lattice Cryptography (2016)*, Chris Peikert ofrece una revisión profunda de los avances más relevantes en criptografía basada en retículas durante los diez años previos. El autor se enfoca en los problemas de Short Integer Solution (SIS) y Learning With Errors (LWE), que constituyen las bases fundamentales para una amplia variedad de esquemas criptográficos modernos, incluyendo cifrado, firmas digitales, funciones de trampa, y en particular construcciones avanzadas como cifrado completamente homomórfico (FHE) y basado en la identidad. El artículo destaca que tanto SIS como LWE ofrecen garantías de seguridad basadas en la dificultad de resolver ciertos problemas en el peor caso sobre redes, algo que no es común en otras ramas de la criptografía[8].

Peikert también expone cómo las versiones estructuradas de estos problemas, como Ring-LWE y Ring-SIS, permiten lograr esquemas más eficientes sin comprometer la seguridad, facilitando implementaciones prácticas. Además, se discuten contribuciones clave como funciones hash resistentes a colisiones, esquemas de cifrado semánticamente seguros y protocolos de firma sin necesidad de oráculos aleatorios. El texto subraya que la criptografía basada en retículas no solo se ha consolidado como una candidata seria para resistir ataques cuánticos, sino que también ha ofrecido soluciones elegantes a problemas abiertos en criptografía clásica, ganando atención en procesos de estandarización como los impulsados por el NIST. Este trabajo se ha convertido en una referencia fundamental para entender la madurez, solidez teórica y aplicabilidad práctica de la criptografía basada en retículas en el contexto post-cuántico.

Una contribución relevante en la implementación hardware de CRYSTALS-Kyber es el trabajo de Sun y Bai (2024), que propone una arquitectura aceleradora de la NTT de alta velocidad basada en radix-4 para plataformas FPGA. El artículo presenta un **acelerador NTT unificado** capaz de ejecutar NTT, INTT y PWM sobre la misma microarquitectura, optimizando la unidad **butterfly** y empleando una técnica de reducción modular K2-RED para el módulo específico de Kyber[13].

Contribuciones técnicas clave

Entre los aportes técnicos del trabajo se destacan:

- Diseño de una unidad *butterfly radix-4* modificada que permite ejecutar las multiplicaciones modulares en paralelo (añadiendo un multiplicador extra) y reduce el retraso por etapa aproximando la latencia a la de una sola multiplicación modular más dos sumas.
- Empleo y adaptación de la **reducción modular K2-RED** (dos pasadas de K-RED) para el módulo $q = 3329$, con un selector de corrección que garantiza resultados en el rango $[0, 3328]$ y ahorro de recursos al preprocesar los twiddle factors como $\omega \cdot k^{-2} \pmod{q}$ en ROMs.
- Esquema de **memoria ping-pong con mapeo conflict-free** (CFMM) y 8 BRAMs para almacenamiento y 5 ROMs para *twiddles*, que permite lectura/escritura paralela libre de conflictos y mayor *throughput* en diseño radix-4.
- Arquitectura configurable en tiempo de compilación (CTC) que soporta distintos grados de paralelismo y modos (NTT/INTT/PWM) sobre la misma lógica, facilitando *trade-offs* entre área y velocidad.

Resultados y parámetros relevantes

Los resultados implementados y reportados muestran:

- Implementación en **Xilinx Virtex-7**funcionando a 296 MHz.
- Parámetros exactos de la NTT empleados: $n = 256$, $q = 3329$ (coeficientes en 12 bits). Entrada y salida son vectores de 256 coeficientes (cada coeficiente ocupa 12 bits).
- Mejora frente a trabajos previos: aceleración en latencia y throughput entre 1.02× y 2.30×, y hasta 3.30× de mejora en area-throughput (ATP) en algunos casos. El diseño sacrifica más BRAMs pero gana significativamente en rendimiento por área.
- Número de twiddle factors almacenados: 425 (distribuidos en 5 ROMs), y la arquitectura realiza pre/post-procesamiento para compensar la forma en que K2-RED devuelve $k^2 \cdot C \pmod{q}$.

Observaciones sobre los autores

El primer autor, **Junyan Sun**, es estudiante de posgrado (M.S.); el autor correspondiente, **Xuefei Bai**, es docente/investigador con Ph.D., ambos afiliados a la University of Science and Technology of China.

Otro aporte significativo en el campo de la criptografía post-cuántica es el de Nguyen, Kieu-Do-Nguyen, Pham y Hoang (2024), quienes presentan un acelerador de hardware de alta velocidad para la transformada NTT enfocado en los algoritmos *CRYSTALS-Kyber* y *CRYSTALS-Dilithium*. Los autores proponen una arquitectura optimizada en Verilog-HDL, sintetizada en FPGA Xilinx *Artix-7*, con el objetivo de reducir la latencia y mejorar el throughput en operaciones de multiplicación polinómica de gran tamaño [14].

Contribuciones técnicas clave

Entre los aportes principales del trabajo se destacan:

- El diseño de un **acelerador NTT unificado**, capaz de ejecutar NTT, INTT y operaciones punto a punto (PWM) en una misma arquitectura.
- Implementación de una unidad *butterfly* optimizada, orientada a reducir la latencia combinacional en las operaciones modulares.
- Integración de técnicas de paralelismo y pipeline, lo que permite incrementar el *throughput* significativamente.
- Compatibilidad con los parámetros de Kyber ($n = 256$, $q = 3329$) y *Dilithium* ($n = 256$, $q = 8380417$), mediante un diseño flexible y reconfigurable en *hardware*.

Resultados y parámetros relevantes

Los resultados reportados muestran:

- Implementación en una **FPGA Xilinx Artix-7 (XC7A100T)** utilizando la herramienta Vivado 2021.2.
- Operación a frecuencias de hasta 250 MHz, con mejoras significativas en latencia y eficiencia área-rendimiento frente a trabajos previos.
- Validación mediante métricas de síntesis (LUTs, FFs, DSPs, BRAMs), confirmando que el diseño logra un balance eficiente entre recursos y rendimiento.

Observaciones sobre los autores

Los autores, Trong-Hung Nguyen, Binh Kieu-Do-Nguyen, Cong-Kha Pham y Trong-Thuc Hoang, están afiliados a instituciones de investigación en Vietnam y Japón, y el trabajo corresponde a un proyecto de investigación de posgrado publicado en *IEEE Access*.

2.8. ANALISIS DE ALGORITMOS DE REDUCCIÓN MODULAR Y FUNCION `MOD` EN VHDL

En esta subsección se presenta un análisis sobre algunos de los métodos fundamentales empleados en la operación de reducción modular, una operación crítica en sistemas criptográficos y de procesamiento digital de señales. En particular, se abordarán dos enfoques: la función `mod` provista por el lenguaje de descripción de hardware VHDL y los algoritmos genéricos de reducción modular más utilizados en implementaciones hardware, como los de Barrett y Montgomery.

La operación de módulo es ampliamente utilizada en sistemas digitales debido a su aplicación directa en la aritmética modular, especialmente en algoritmos criptográficos como RSA, ECC y los esquemas post-cuánticos como Crystals-Kyber. Aunque VHDL proporciona la función `mod` de manera nativa, su uso en hardware sintetizable debe evaluarse cuidadosamente en términos de eficiencia, dado que una operación aparentemente simple puede traducirse en una arquitectura costosa en recursos y con bajo rendimiento si no se implementa adecuadamente.

Por otro lado, los algoritmos clásicos de reducción modular como Barrett y Montgomery han sido objeto de múltiples investigaciones debido a su efectividad para implementar operaciones modulares sin recurrir directamente a divisiones, que suelen ser costosas en hardware. Estos métodos permiten reemplazar divisiones por multiplicaciones y desplazamientos, optimizando así el desempeño del circuito.

El objetivo de este apartado es establecer una comparación conceptual y técnica entre la implementación directa de la función `mod` en VHDL y el uso de algoritmos optimizados de reducción modular. Se busca identificar las implicaciones que cada enfoque tiene en términos de recursos de área, tiempo de ejecución y escalabilidad en arquitecturas FPGA, con énfasis en su aplicabilidad a sistemas criptográficos y transformadas como la NTT.

2.8.1. Algoritmo de reducción Barrett

El algoritmo de Barrett realiza la operación $z = x \bmod m$ usando aritmética de multiplicación y resta en lugar de división entera. La idea central es precomputar una aproximación de la inversa del módulo, normalmente denotada μ. Por ejemplo, escogiendo una base $B = 2^k$ (con k número de bits aproximadamente del doble de m), Barrett propone calcular la constante $\mu = \left\lfloor \frac{B^{2k}}{m} \right\rfloor$[5]. Esta constante $\mu \approx 1/m$ en formato entero escalado. Con μ precomputado, la reducción se obtiene mediante los siguientes pasos generales: (1) se calcula $t = \left\lfloor \frac{x}{B^{k-1}} \right\rfloor$ (una estimación del cociente x/m obtenida con corrimientos), (2) se multiplica $q = t \times \mu$, y luego (3) se desplaza q a la derecha $k+1$ bits para obtener $\tilde{q} = \left\lfloor \frac{q}{B^{k+1}} \right\rfloor$ que sirve como aproximación del cociente real x/m. Después (4) se calcula $r = x - \tilde{q} \times m$, obteniendo un residuo provisional, y finalmente (5) si $r \geq m$, se resta m una vez (o repetidamente hasta ajustar al rango). En forma de pseudocódigo simplificado, Barrett se resume así[5]:

Entrada: x (valor a reducir), m (módulo), precomputados $B = 2^k$ y $\mu = \left\lfloor \frac{B^2}{m} \right\rfloor$ (constante de Barrett).

Cálculo:

1. $t = \left\lfloor \frac{x}{B} \right\rfloor$ (toma los bits altos de x).
2. $q = t \times \mu$ (multiplicación por la constante precomputada).
3. $q' = \left\lfloor \frac{q}{B} \right\rfloor$ (toma los bits altos del producto anterior).
4. $r = x - q' \times m$ (resta el producto del módulo).

5. Si $r \geq m$ entonces $r = r - m$ (posible ajuste final).

Salida: $r \equiv x \pmod{m}$.

En esencia, Barrett usa una multiplicación por la constante μ para aproximar la división por m arxiv.org. El resultado r obtenido tras la resta puede exceder m solo ligeramente (a lo sumo una vez m), por lo cual basta con una o dos correcciones condicionales para obtener el residuo exacto. Matemáticamente, este método aprovecha que $\mu \approx \frac{B^k}{m}$, de modo que $x \cdot \mu / B^k \approx x/m$.

Flujo de datos en hardware Una implementación típica en FPGA del algoritmo de Barrett incluye: (a) un multiplicador de precisión múltiple para calcular $t \times \mu$ (puede ser un multiplicador DSP si la FPGA lo soporta), (b) registros o circuitos de corrimiento para las divisiones por potencias de $B = 2^k$ (que se realizan por desplazamiento de bits, operación muy eficiente en hardware), y (c) un bloque sumador/restador para computar $r = x - q' \times m$ y realizar la posible resta final de m. En hardware, B suele elegirse como una potencia de 2 (p. ej. $B = 2^{32}$ o 2^w para algún tamaño de palabra) facilitando los desplazamientos. La figura a continuación ilustra esquemáticamente los pasos principales de la reducción de Barrett, donde se observa la ruta de datos principal: multiplicación por μ, obtención del producto más significativo (desplazamiento) y resta del módulo[6]:

Pseudocódigo del algoritmo de Barrett, que muestra la multiplicación por la constante precomputada $m = \lfloor 2^k/m \rfloor$ y la resta subsecuente del módulo q para obtener el residuo[5].

2.8.2. Algoritmo de reducción Montgomery

El algoritmo de Montgomery, propuesto por Peter L. Montgomery (1985), es otra técnica clásica para evitar divisiones costosas al efectuar multiplicaciones modulares. A diferencia de Barrett, Montgomery opera en un sistema de residuos transformados: dado un módulo m, se elige un factor $R = 2^n$ (con n típicamente igual al número de bits de m) que sea coprimo con m (lo que implica usualmente que m es impar)[7]. Los números se convierten a la forma Montgomery multiplicándolos por R mód m. Entonces, la multiplicación de Montgomery permite calcular $A \times B \times R^{-1}$ mód m sin realizar divisiones explícitas por m. En términos prácticos, la reducción de Montgomery toma un producto convencional $T = A \times B$ y lo reduce a $T \cdot R^{-1}$ mód m de manera eficiente, eliminando el factor R. La lógica básica iterativa del algoritmo de Montgomery (en su forma radix-2) se puede describir así:

Entrada: A, B en representación Montgomery (es decir, $A' = A \times R$ mód m, $B' = B \times R \pmod{m}$), módulo m (impar), n = número de bits.

Inicialización: $R = 0$.

Bucle para $i = 0$ hasta $n - 1$:

1. $R = R + a_i \times B$ (se añade a R el multiplicando B si el bit i-ésimo de A es 1).

2. Si R es impar ($r_0 = 1$) entonces $R = R + m$ (si el bit menos significativo r_0 es 1, se agrega el módulo).

3. $R = R \div 2$ (desplazar R a la derecha un bit, efectivamente dividiendo entre 2).

Fin bucle (tras n iteraciones).
Salida: R (que resulta ser $A \times B \times R^{-1} \mod m$). Si A' y B' estaban en forma Montgomery, entonces R corresponde a $(A \times B) \mod m$ en representación normal.

Este proceso realiza la reducción modular "en línea"durante la suma de productos, evitando una división por m al final. La corrección condicional $R = R + m$ cuando R es impar garantiza que el bit menos significativo de R (después de sumar m si era necesario) sea 0, de modo que el corrimiento a la derecha equivale a dividir por 2 sin pérdida de información mod m. Un requisito importante es que m sea impar, para que 2 tenga inverso multiplicativo mod m y el algoritmo sea válido (si m fuera par, el factor $R = 2^n$ no sería coprimo con m).

2.8.3. Implementaciones en FPGA y Comparación de Rendimiento

En años recientes, numerosos trabajos han implementado y comparado los algoritmos de Barrett y Montgomery en FPGAs, incluyendo dispositivos Intel/Altera, con diferentes arquitecturas optimizadas[9]. A continuación, se resumen hallazgos clave de implementaciones prácticas en cuanto a recursos utilizados, velocidades alcanzadas y eficiencia global:

- **Uso de recursos y latencia:** En una implementación iterativa clásica de Montgomery sobre FPGA Altera Cyclone (placa DE2-70), se reporta un consumo de 17,540 elementos lógicos (equivalente a 15,480 LUTs, 25.6 % de la FPGA) para operar con operandos de 2048 bits, con una latencia de 2048 ciclos de reloj por multiplicación modular. Este diseño en VHDL/Verilog sintetizado con Altera Quartus II mostró mejor desempeño en la métrica área-tiempo-cuadrado ($\text{AT}\hat{2}$) frente a diseños previos, gracias a su simplicidad y paralelismo moderado. En contraste, una implementación de Barrett para tamaños similares requeriría al menos dos multiplicadores grandes de 2048×2048 bits o su equivalente en lógica/DSP, lo cual típicamente ocupa bastante más LUTs. De hecho, un estudio de 2018 en Spartan-6 (Xilinx) encontró que un módulo Barrett de 1024 bits ocupaba alrededor de 2770 LUTs frente a solo 489 LUTs para un módulo de reducción Montgomery de similar tamaño, ilustrando la diferencia de complejidad combinacional. No obstante, el módulo Barrett en ese caso logró tiempos de operación ligeramente menores gracias a su cálculo más combinado (menos ciclos).

- **Rendimiento en tiempo:** Para medir la velocidad, es útil ver casos de estudio con tamaños de operandos típicos de RSA/ECC. En 2019, Gürdan et al. diseñaron un multiplicador modular Montgomery optimizado por

dígitos utilizando DSPs en FPGA, y lo compararon con un multiplicador Barrett "full-word"también usando DSPs. En un dispositivo FPGA moderno (Xilinx Virtex-7) ambos diseños alcanzaron frecuencias altas, logrando tiempos por operación muy pequeños incluso para anchos de hasta 1056 bits. En sus resultados, el multiplicador Montgomery fue más rápido: por ejemplo, una multiplicación modular de 528 bits tomó 0.43 μs con Montgomery, frente a 0.49 μs con Barrett; y para 1056 bits, Montgomery tardó 1.09 μs contra 1.88 μs de Barrett. Esto indica que, cuando ambos algoritmos se llevan al extremo de utilizar recursos de forma agresiva (DSPs, pipeline), Montgomery puede escalar mejor en desempeño para operandos muy grandes, posiblemente debido a que su arquitectura digit-serial paralelizada aprovecha mejor los bloques aritméticos internos[5]. Barrett, aunque también paralelo, tuvo un costo temporal ligeramente mayor, especialmente notable a 1056 bits (un 72 % más lento que Montgomery en ese caso). Los autores también implementaron versiones usando herramientas de alto nivel (HLS) y observaron que el Montgomery generado por HLS superaba al Barrett generado por HLS en rendimiento, lo que sugiere que Montgomery es más amigable a optimizaciones automáticas.

- **Eficiencia área/velocidad:** Si se considera la eficiencia como la razón entre el throughput obtenido y los recursos empleados, ambos algoritmos tienen distintos puntos fuertes. Montgomery tiende a requerir menos área para una solución serial y con datos de 16 bits en adelante y más area para tamaños menores a 16 bits[6], pero si se desea alta velocidad mediante paralelismo, su área crece linealmente con el factor de paralelismo (por ejemplo, procesar 32 bits por ciclo en lugar de 1 bit por ciclo multiplica el uso de sumadores y registros)[9]. Barrett inicia con más área combinacional (por los multiplicadores de gran tamaño), pero una vez que se usan los DSPs disponibles, puede lograr throughput muy alto con un número fijo de etapas. En la práctica, estudios han demostrado que para módulos de tamaño pequeño a mediano, Barrett puede lograr latencias (en nanosegundos) inferiores a Montgomery para tamaños menores a 16 bits con arquitecturas equivalentes, y ademas consumiendo menos slices/LUTs[6]. Por ejemplo, Walter et al. mostraron que para anchos menores a 16 bits Barrett era preferible en FPGAs, mientras que superado ese tamaño Montgomery ofrecía menor latencia total. Esto se atribuye a que las FPGAs modernas incluyen multiplicadores hardware que favorecen algoritmos basados en multiplicación (como Barrett) cuando el ancho es lo suficientemente grande para utilizar esos DSP eficientemente. Por otro lado, en sistemas de muy gran tamaño (512–2048 bits), la estrategia iterativa de Montgomery puede mantener un uso de área moderado y tolerar frecuencias de reloj altas al estar basada en operaciones simples (sumas y desplazamientos con carry predictible), lo que le permite alcanzar una eficiencia global notable (multiplicaciones por segundo por LUT) en diseños optimizados.

3. DISEÑO DE LA NTT DE CRYSTALS-KYBER

Para el desarrollo de este proyecto, se implementaron versiones sucesivas de la Transformada Teórico Numérica (NTT) en software y hardware, siguiendo un enfoque incremental. Inicialmente, se realizaron implementaciones para tamaños reducidos de entrada (16 y 32 puntos) utilizando flujos de datos de tipo Word Groups y radix-2, permitiendo una interiorización del algoritmo y la validación de las operaciones fundamentales. Posteriormente, se migró hacia el flujo Constant Geometry, aplicándolo a tamaños de 16, 32 y 256 puntos, optimizando así el uso de recursos hardware. Finalmente, para el caso de 256 puntos, se diseñaron distintas arquitecturas orientadas a la eficiencia temporal y espacial; una versión paralela, dos versiones tipo pipeline y una con reutilizacion de componentes. Cada implementación fue simulada, sintetizada y probada en la FPGA EP4CE115F29C7N, conforme a los objetivos planteados en este trabajo de grado.

3.1. IMPLEMENTACIONES INICIALES EN SOFTWARE

En las implementaciones iniciales en software se utilizó como base el algoritmo descrito en el estándar CRYSTALS-KYBER[1], el cual define la Transformada Teórico Numérica (NTT) para operaciones sobre cuerpos finitos con parámetros fijos (n = 256, q = 3329). No obstante, para facilitar la comprensión y validar el comportamiento del algoritmo, se realizaron versiones adaptadas a tamaños reducidos de entrada (n = 16, n = 32 y n=128), ajustando el módulo y las raíces primitivas de acuerdo con los requisitos matemáticos de la NTT en cada caso, finalmente se realizó la implementación final de 256 puntos teniendo en cuenta sus parametros asociados. El siguiente pseudocódigo corresponde al Algoritmo 8 del estándar NIST FIPS 203 [1], que sirve como base para las implementaciones desarrolladas:

Algorithm 1: NTT(f) – Transformada Teórico Numérica

Input: Arreglo $f \in \mathbb{Z}_q^{256}$ `// Coeficientes del polinomio de entrada`

Output: Arreglo $\hat{f} \in \mathbb{Z}_q^{256}$ `// NTT del polinomio de entrada`

$\hat{f} \leftarrow f$ `// Copia inicial para operar in-place`

$k \leftarrow 1$

for $len \leftarrow 128; len \geq 2; len \leftarrow len/2$ **do**

  for $start \leftarrow 0; start < 256; start \leftarrow start + 2 \cdot len$ **do**

    $\zeta \leftarrow \zeta^{\mathrm{BitRev}_7(k)} \mod q$

    $k \leftarrow k + 1$

    for $j \leftarrow start; j < start + len; j++$ **do**

      $t \leftarrow \zeta \cdot \hat{f}[j + \mathrm{len}] \mod q$

      $\hat{f}[j + \mathrm{len}] \leftarrow (\hat{f}[j] - t) \mod q$

      $\hat{f}[j] \leftarrow (\hat{f}[j] + t) \mod q$

return $\hat{f}$

3.1.1. Implementaciones en C++ de la NTT para diferentes tamaños de entrada

Como parte del proceso de validación y comprensión del comportamiento de la Transformada Teórico Numérica (NTT), se desarrollaron diversas implementaciones en lenguaje C++ que abarcan una gama progresiva de tamaños de entrada. Este enfoque permitió observar tanto el desempeño como la correcta ejecución del algoritmo en distintas configuraciones, desde entornos reducidos y controlados hasta configuraciones compatibles con estándares criptográficos actuales.

Inicialmente, se implementó una versión del algoritmo para $n = 16$ puntos, empleando el cuerpo finito $\mathbb{Z}_{17}$. Esta configuración, sencilla y controlable, sirvió como punto de partida para verificar la estructura general de la transformada, así como para validar las operaciones modulares y el procedimiento de ordenamiento mediante bit-reversal sobre índices de 4 bits. La raíz primitiva utilizada fue $\zeta = 6$, cumpliendo las propiedades algebraicas requeridas para garantizar una transformada válida en este dominio.

Posteriormente, se escaló la implementación a $n = 32$ puntos utilizando el cuerpo finito $\mathbb{Z}_{97}$, con raíz primitiva $\zeta = 19$. Esta versión mantuvo la estrategia de decimación en el tiempo (DIT) y realizó las operaciones in-place, aplicando bit-reversal sobre 5 bits. La validación de los resultados se realizó utilizando un programa auxiliar para el cálculo de los factores de multiplicación (twiddle factors), cuyos valores fueron comparados contra simulaciones manuales realizadas en hojas de cálculo, lo cual permitió confirmar la correcta ejecución del flujo de datos.

La siguiente ampliación llevó la implementación a $n = 128$ puntos, empleando parámetros consistentes con el estándar NIST FIPS 203: un módulo primo

de $q = 3329$ y una raíz primitiva $\zeta = 289$. Esta raíz cumple con la propiedad $\zeta^{64} \equiv -1 \mod 3329$, lo cual asegura su idoneidad como raíz de orden 128. Nuevamente, el algoritmo se ejecutó en modalidad in-place y con bit-reversal ajustado a 7 bits, replicando las mismas metodologías de validación manual descritas anteriormente, y asegurando así una transición fluida hacia estructuras más complejas.

Finalmente, se implementó la versión correspondiente a $n = 256$ puntos, utilizando los parámetros exactos definidos por el estándar CRYSTALS-KYBER: módulo $q = 3329$ y raíz primitiva $\zeta = 17$. Esta versión no solo implicó un aumento significativo en el número de operaciones, sino que además se estructuró cuidadosamente para reflejar el comportamiento esperado según el estándar, incluyendo el orden específico de las mariposas (butterflies) y la disposición geométrica de los datos. La validación se realizó por comparación con resultados generados manualmente, consolidando así la fidelidad de la implementación.

Estas versiones progresivas en C++ no solo permitieron afinar el entendimiento de los detalles matemáticos y computacionales de la NTT, sino que también proporcionaron una base sólida y confiable para futuros desarrollos y migraciones hacia plataformas hardware.

Tabla 2: Parámetros utilizados en las implementaciones de la NTT en software

n	**Módulo q**	**Raíz primitiva ζ**	**Bits de bit-reversal**	**Estándar / Uso**
16	17	6	3	Prueba inicial (manual)
32	97	19	4	Validación extendida
128	3329	289	6	Compatible con NIST FIPS 203
256	3329	17	7	CRYSTALS-KYBER

Las distintas implementaciones en software desarrolladas para tamaños $n = 16, 32, 128$ y 256 permitieron construir una comprensión progresiva del comportamiento de la Transformada Teórico Numérica (NTT) y de sus particularidades en cuerpos finitos. Al ajustar los parámetros de módulo y raíz de la unidad en cada caso, fue posible validar paso a paso la lógica de funcionamiento del algoritmo.

Este enfoque incremental resultó fundamental para detectar errores, optimizar estructuras y confirmar resultados, incluso antes de abordar implementaciones más complejas o específicas del estándar. Además, la validación de resultados mediante herramientas auxiliares como hojas de cálculo fortaleció la confianza en la consistencia de las salidas obtenidas.

Nota: el código y resultados correspondientes a estas implementaciones se presentan en los Anexos 1, 2 y 3.

3.2. IMPLEMENTACIONES EN HARDWARE USANDO EL FLUJO WORD GROUPS

La implementación en hardware de la NTT requiere un diseño cuidadoso del flujo de datos y el acceso a memoria. Una estrategia ampliamente utilizada en

arquitecturas FFT es el procesamiento por *Word Groups* (grupos de palabras), que consiste en dividir los datos en bloques que son procesados de forma conjunta durante cada etapa de la transformada.

En lugar de operar sobre cada dato de manera individual, el enfoque de Word Groups permite cargar múltiples datos desde memoria que pertenecen al mismo grupo, procesarlos simultáneamente mediante mariposas, y luego escribirlos nuevamente en memoria, todo dentro de un solo ciclo de control. Este esquema se repite por etapas, reorganizando el flujo de datos internamente.

Una característica clave de este método es que el tamaño del grupo de pares de datos inicial se reduce a la mitad en cada etapa. Por ejemplo, si el grupo inicial contiene 8 datos, en la siguiente etapa se dividirá en dos grupos de 4, luego en cuatro grupos de 2, y así sucesivamente. Esta división implica que, aunque el número total de accesos de lectura y escritura se mantiene constante, la posición de esos accesos cambia dinámicamente entre etapas.

Como consecuencia, el diseño de control debe adaptarse etapa por etapa para calcular correctamente las direcciones de memoria y coordinar las operaciones. Esto también significa que no existe un patrón único de conexión reutilizable entre etapas, lo que puede aumentar la complejidad de implementación, especialmente en arquitecturas rígidas.

La Figura 3 ilustra este comportamiento para el caso de una FFT DIF radix-2 de 16 puntos (comportamiento usado para la NTT), mostrando cómo evoluciona el flujo de datos cuando se procesan en grupos de 2 y 4 palabras. Cada etapa modifica las agrupaciones y las conexiones internas, lo cual exige una lógica de control variable en función del tamaño del grupo.

Este enfoque, aunque más complejo de controlar, permite adaptar la organización del cómputo al comportamiento natural de la transformada, aprovechando eficientemente las capacidades del hardware disponible.

3.2.1. Implementación en VHDL para 16 puntos

La primera implementación en hardware de la NTT fue realizada para una longitud de entrada de $n = 16$ elementos, utilizando el cuerpo finito $\mathbb{Z}_{17}$ y la raíz primitiva $\zeta = 6$. Esta versión fue desarrollada en VHDL como un módulo monolítico, es decir, todo el procesamiento de la transformada se codificó dentro de un único bloque secuencial.

El diseño replica paso a paso las tres etapas de la NTT en flujo Word Groups, realizando en cada una de ellas las mariposas correspondientes, las multiplicaciones por los factores ω^k (precalculados según la etapa), y las sumas y restas modulares necesarias. El código fue estructurado de forma directa, codificando explícitamente los índices, los factores de multiplicación y la reorganización de los datos entre etapas.

Cada iteración procesa los datos según la estructura de mariposas radix-2 en decimación en el tiempo, organizados como grupos de datos cuyo tamaño se reduce a la mitad en cada etapa. De este modo, el diseño refleja fielmente el comportamiento del flujo Word Groups ya descrito, con acceso a memoria y combinaciones de datos que cambian etapa tras etapa.

Para la validación funcional del sistema, no se utilizó una transformada inversa. En su lugar, los resultados obtenidos en esta implementación fueron comparados directamente con los generados por la versión en software (C++) etapa por etapa. En dicha validación se emplearon hojas de cálculo que replican el flujo de datos de la NTT, permitiendo visualizar las combinaciones intermedias y verificar que los resultados del hardware coincidan con los esperados en cada punto. El RTL asociado a esta arquitectura se presenta en el Anexo 9.

Nota: Los resultados de la síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 10, 11 y 12.

3.2.2. Implementación de Operaciones Aritméticas en Hardware

Las funciones descritas en esta sección son utilizadas de forma común en las implementaciones de la NTT con flujo Word Groups, tanto para 16 como para 32 puntos, ajustando los parámetros correspondientes a cada caso.

Para la implementación en hardware de la NTT de 16 puntos utilizando flujo Word Groups, se desarrollaron funciones específicas en VHDL para realizar las operaciones aritméticas requeridas en el dominio modular, correspondientes a la suma, resta y multiplicación en el cuerpo finito $\mathbb{Z}_{17}$. Estas funciones fueron empaquetadas en la librería `pckg_OPMOD`, facilitando su integración y reutilización en los módulos de mayor jerarquía.

Función de Suma Modular (`suma_m`) Esta función realiza la operación de suma modular considerando la reducción inmediata del resultado para asegurar que permanezca dentro del rango válido del módulo 17. Si la suma de los operandos excede dicho valor, se realiza la operación de reducción correspondiente:

$$\text{suma\_m}(A, B) = \begin{cases} A + B - 17, & \text{si } A + B \geq 17 \\ A + B, & \text{en caso contrario} \end{cases}$$

Función de Resta Modular (`resta_m`) La resta modular implementa un mecanismo de corrección para garantizar que el resultado no sea negativo. Cuando el minuendo es menor que el sustraendo, se ajusta el resultado sumando el módulo:

$$\text{resta\_m}(A, B) = \begin{cases} A - B + 17, & \text{si } A < B \\ A - B, & \text{en caso contrario} \end{cases}$$

Función de Multiplicación Modular (`mult_m`) La operación de multiplicación modular calcula el producto de los operandos y aplica la reducción módulo 17 utilizando la función `mod`, la cual asegura que el resultado final se mantenga en el rango válido del cuerpo finito $\mathbb{Z}_{17}$. Esta función es de vital importancia para la etapa de aplicación de los factores ω^k en la mariposa de la NTT, donde $\omega = 6$ es la raíz primitiva de la unidad en $\mathbb{Z}_{17}$:

$$\text{mult\_m}(A, B) = (A \times B) \mod 17$$

Estas funciones fueron diseñadas para operar con vectores de 5 bits, lo cual permite representar de forma eficiente los valores en el rango de $[0, 16]$. La correcta implementación y verificación de estas operaciones es fundamental para asegurar la exactitud de los resultados en cada etapa de la NTT, especialmente al realizar las combinaciones intermedias de los datos en las mariposas radix-2.

3.2.3. Implementación en VHDL para 32 puntos

La segunda implementación en hardware se desarrolló para una NTT de $n = 32$ puntos, empleando el cuerpo finito $\mathbb{Z}_{97}$ y utilizando como raíz primitiva de la unidad el valor $\zeta = 19$. Al igual que en el caso de 16 puntos, esta versión fue implementada como un módulo monolítico en VHDL, replicando explícitamente cada etapa de procesamiento dentro de un único bloque.

El flujo de datos se organizó nuevamente bajo la metodología Word Groups, en la cual los datos se agrupan y se procesan de forma conjunta por etapas. Con cada nueva etapa, el tamaño de los grupos se reduce a la mitad, lo que conlleva un aumento en la cantidad de grupos y un cambio en el patrón de lectura y escritura en memoria. Este comportamiento se refleja directamente en el código, que fue escrito considerando los factores ω^k específicos para cada mariposa, y las operaciones modulares correspondientes.

La implementación realiza internamente todas las operaciones de suma, resta y multiplicación modular, controlando de forma secuencial el avance de las tres etapas de la transformada. Al tratarse de una estructura codificada directamente, el diseño no emplea controladores dinámicos de índice, sino que las posiciones y operaciones están fijadas en el flujo lógico del módulo.

Para verificar el correcto funcionamiento de esta versión, se utilizó el mismo procedimiento que en el caso anterior: se compararon los resultados intermedios y finales del hardware con los generados por la implementación en software. El análisis por etapas se apoyó en hojas de cálculo que modelan el flujo de la NTT con los factores y combinaciones correctas, lo que permitió corroborar la coincidencia total de los resultados.

3.3. IMPLEMENTACIONES EN HARDWARE USANDO EL FLUJO CONSTANT GEOMETRY

Tras la experiencia con el enfoque Word Groups, se exploró una segunda metodología para la implementación en hardware de la NTT: el flujo de *Constant Geometry*. Esta alternativa se basa en una estructura regular donde todas las etapas de la transformada comparten exactamente el mismo patrón de conexión entre datos. En este enfoque, la organización de las mariposas y los accesos a memoria permanecen constantes a lo largo de todas las etapas.

A diferencia del enfoque Word Groups —en el que el tamaño del grupo se reduce progresivamente y cambia el patrón de acceso a memoria en cada etapa—, el flujo Constant Geometry mantiene:

- El mismo número de agrupaciones de datos en todas las etapas.

- Las mismas posiciones de lectura y escritura de memoria.
- Una lógica de direccionamiento fija y predecible.

Esto simplifica el diseño del control de hardware y reduce el riesgo de errores asociados a la manipulación dinámica de índices. Además, al conservar una estructura repetitiva y uniforme, permite una mayor reutilización de módulos de hardware y una mejor escalabilidad hacia tamaños mayores.

La estructura de este tipo de flujo puede observarse en la Figura 5, donde se muestra el grafo de una FFT radix-2 DIF de 16 puntos con geometría constante. Aunque está representada como una FFT, la misma topología fue utilizada para implementar la NTT en este trabajo, debido a la equivalencia estructural entre ambas transformadas.

Nota: La Figura 5 fue presentada anteriormente en la Sección 2.3.2.

3.3.1. Optimización de la Reducción Modular mediante el Algoritmo de Barrett

Con el fin de optimizar la utilización de recursos en hardware y reducir la latencia de las operaciones aritméticas modulares durante la implementación de la NTT con flujo *Constant Geometry*, se integró el algoritmo de Barrett para la reducción modular. Este método reemplaza la tradicional operación de reducción basada en la función `mod`, permitiendo realizar la operación con un menor consumo de recursos lógicos y un tiempo de cómputo reducido, especialmente en plataformas FPGA.

Principio del Algoritmo de Barrett El algoritmo de Barrett permite calcular de forma eficiente la operación de reducción modular, evitando divisiones costosas en hardware. Dado un número entero x que se desea reducir módulo m, la aproximación se realiza utilizando las ecuaciones (6), (7), (8), (9) y (10):

$$x \mod m \approx x - \lfloor q \rfloor \cdot m \tag{6}$$

$$2^k \geq m \tag{7}$$

$$\mu = \left\lfloor \frac{2^k}{N} \right\rfloor \tag{8}$$

$$q = \left\lfloor \frac{\mu \cdot x}{2^k} \right\rfloor \tag{9}$$

$$r = x - q \cdot N \tag{10}$$

En hardware, esta operación se implementa mediante multiplicaciones y desplazamientos a la derecha, eliminando la necesidad de operaciones de división explícitas.

Implementación en VHDL La función `barrett_reduce` fue implementada en VHDL dentro del paquete `pckg_OPMOD`, y se utiliza directamente en la función `mult_m` para realizar la reducción modular de forma más eficiente. Esta función opera sobre operandos de hasta 24 bits y permite realizar la reducción módulo 3329, optimizando las operaciones de multiplicación en el dominio de $\mathbb{Z}_{3329}$, que es el módulo utilizado en las implementaciones de NTT para tamaños de 128 y 256 puntos.

Parámetros de la Implementación para 128 y 256 puntos

- $k = 24$: Número de bits utilizados en la operación intermedia.
- $\mu = \lfloor \frac{2^k}{m} \rfloor = 5039$: Constante precalculada para simplificar la multiplicación.
- $m = 3329$: Módulo utilizado en las implementaciones de NTT con tamaños mayores.

Ventajas de la Optimización El uso del algoritmo de Barrett permite reducir la cantidad de recursos lógicos consumidos, especialmente registros y unidades de división, además de minimizar la latencia de la operación de reducción modular. Esto se traduce en implementaciones más compactas y rápidas, lo que es fundamental en aplicaciones de criptografía post-cuántica donde la eficiencia es crítica.

3.3.2. Implementación en VHDL para 16 puntos

La primera implementación bajo el enfoque de geometría constante se desarrolló para una NTT de $n = 16$ puntos, utilizando el cuerpo finito $\mathbb{Z}_{17}$ y la raíz primitiva $\zeta = 6$. A diferencia de la implementación con Word Groups, esta versión presenta una estructura más uniforme, donde todas las etapas del procesamiento comparten exactamente el mismo patrón de conexiones y operaciones.

El diseño fue implementado en VHDL utilizando un módulo monolítico, donde se replicaron de forma explícita las etapas de la transformada, manteniendo fijo el patrón de combinaciones de datos. Esto fue posible gracias a que, en el flujo Constant Geometry, las mismas posiciones de memoria se leen y escriben en cada etapa, y las mariposas operan sobre los mismos pares de índices.

Esta regularidad permitió simplificar considerablemente el diseño de control, ya que no fue necesario recalcular direcciones de lectura y escritura entre etapas. Además, al no variar la cantidad ni la ubicación de los grupos de datos, se redujo el riesgo de errores de conexión o solapamiento de datos.

Cada etapa aplica las mariposas correspondientes con sus respectivos factores de multiplicación ω^k, manteniendo la estructura típica de una transformada DIF radix-2. Como en las implementaciones anteriores, los factores fueron precalculados y codificados directamente en el diseño.

La validación funcional de esta versión se realizó mediante la comparación de resultados con la implementación en software, utilizando hojas de cálculo que

replicaban el flujo Constant Geometry. Esta comparación se efectuó etapa por etapa, confirmando la correcta operación del hardware.

Si bien los valores numéricos obtenidos en ambas implementaciones fueron equivalentes, se observó una diferencia en el orden de salida de los datos. Por ejemplo, el valor correspondiente a la posición 2 en la salida del software aparece en la posición 8 en la salida del hardware; el valor de la posición 3 en software corresponde a la posición 2 en hardware, y el valor de la posición 4 en software se encuentra en la posición 9 en hardware, y así sucesivamente. Esta diferencia es atribuible a la forma en que se distribuyen y reorganizan los datos internamente en la arquitectura de geometría constante.

Nota: Los resultados de síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 10, 11 y 12.

3.3.3. Implementación en VHDL para 32 puntos

La implementación para $n = 32$ puntos bajo el enfoque de geometría constante se diseñó con base en el cuerpo finito $\mathbb{Z}_{97}$ y utilizando como raíz primitiva el valor $\zeta = 19$. Al igual que en el caso de 16 puntos, se adoptó esta estructura completamente regular, en la que todas las etapas de la NTT comparten un patrón idéntico de conexiones y combinaciones de datos.

El diseño fue codificado en VHDL como un único módulo secuencial. Gracias a la estructura de geometría constante, las direcciones de lectura y escritura en memoria permanecen fijas a lo largo de toda la ejecución de la transformada, lo cual simplificó considerablemente el control de flujo y minimizó las posibilidades de error por direccionamiento.

Las mariposas se aplican de forma repetitiva sobre los mismos pares de índices en cada etapa, utilizando los factores de multiplicación ω^k correspondientes, los cuales fueron precalculados e incluidos directamente en el diseño. Esta repetitividad no solo facilita la implementación, sino que también hace más predecible y verificable el comportamiento del sistema.

Para validar esta versión, se realizó una comparación directa con los resultados de la implementación en software. Dado que se conoce la equivalencia estructural entre la NTT y la FFT en términos de flujo de datos, se utilizó una hoja de cálculo que modela el comportamiento de la transformada bajo geometría constante, permitiendo visualizar y verificar los resultados intermedios y finales.

Al igual que en la implementación para 16 puntos, se observó una diferencia en el orden de salida respecto a la versión en software. Aunque los valores numéricos obtenidos fueron idénticos, sus posiciones en la salida variaron. Por ejemplo, algunos datos que en el software aparecen en posiciones consecutivas, en el hardware bajo geometría constante aparecen distribuidos en posiciones no adyacentes. Esta diferencia no afecta la validez de los resultados, pero debe tenerse en cuenta al momento de comparar salidas o integrarlas en etapas posteriores del sistema.

Nota: Los resultados de síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 13, 14 y 15

3.3.4. Implementación en VHDL para 128 puntos (un sólo ciclo)

La implementación de la NTT para $n = 128$ puntos se diseñó siguiendo un enfoque completamente paralelo, en el cual toda la transformada se ejecuta en un único ciclo de reloj. Esta arquitectura mantiene el esquema de geometría constante, facilitando la organización regular de las etapas y la reutilización de patrones de procesamiento.

El diseño se estructuró en tres componentes principales. En primer lugar, se desarrolló un *package* que contiene las funciones modulares necesarias para realizar las operaciones de suma, resta y multiplicación, todas parametrizadas con el módulo $q = 3329$. Estas operaciones se ejecutan de manera concurrente y optimizada para el entorno hardware.

En segundo lugar, se implementó una memoria tipo ROM, denominada `ROM_memoria`, encargada de almacenar los 128 valores de entrada. Esta memoria provee los datos a la entidad principal de forma secuencial.

Por último, la entidad principal controla el flujo general del sistema. A medida que los datos son cargados desde la ROM, se almacenan internamente hasta completar los 128 valores. Una vez alcanzada esta condición, se activa una señal denominada `lleno`, la cual habilita el proceso de transformada. En ese instante, todas las operaciones necesarias para calcular la NTT se ejecutan simultáneamente, y el sistema entrega los 128 resultados procesados de forma inmediata.

Este enfoque maximiza la velocidad de ejecución, ya que toda la transformada se realiza en un solo ciclo de reloj. No obstante, dicha ventaja implica también un mayor consumo de recursos lógicos, ya que requiere instanciar todas las operaciones necesarias en paralelo. La evaluación de esta arquitectura en términos de área y latencia se detalla en las secciones comparativas posteriores.

Nota: Los resultados de síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 16, 17 y 18.

3.3.5. Implementación pipeline en VHDL con 64 mariposas

Esta versión de la NTT para $n = 128$ puntos fue implementada con una arquitectura tipo *pipeline*, conservando el enfoque de geometría constante y operando 64 mariposas por vez. A diferencia de la implementación completamente paralela, en esta variante se incorporaron registros intermedios que permiten realizar el procesamiento por etapas distribuidas en el tiempo, facilitando la escalabilidad y el flujo continuo de datos.

El diseño se basa en un conjunto de registros capaces de almacenar 128 datos de 12 bits cada uno. Estos registros actúan como bancos intermedios que conservan los resultados generados por cada etapa de la transformada, permitiendo que los datos sean utilizados inmediatamente en la siguiente etapa. La transformada completa requiere 6 etapas, y dado que cada etapa implica una operación

de lectura y posterior escritura, el proceso total se completa en 6 ciclos de reloj debido a que la ultima etapa no requiere registro para almacenar la salida.

Un aspecto fundamental de esta arquitectura es que, aunque cada registro tiene una única entrada, permite sobrescribir los datos justo después de ser leídos. Es decir, los valores que se leen en un ciclo pueden ser reemplazados inmediatamente en el siguiente, sin conflictos, lo que posibilita una alimentación continua del sistema.

Gracias a este mecanismo, es posible ingresar una nueva entrada al sistema en cada ciclo. Tras un retardo inicial de 6 ciclos, los resultados de las entradas comienzan a emerger en cada ciclo sucesivo. Por ejemplo, la salida correspondiente a la primera entrada cargada estará disponible en el ciclo 7, mientras que la de la segunda entrada aparecerá en el ciclo 8, y así sucesivamente.

Esta arquitectura balancea eficiencia temporal y uso de recursos, permitiendo un procesamiento continuo de datos sin recurrir a una duplicación completa de lógica como en la versión paralela. Además, la naturaleza regular del flujo Constant Geometry facilita la organización de las conexiones y el control secuencial. Con el objetivo de facilitar al lector la comprensión del flujo de información, se presenta el diagrama de la figura 6.

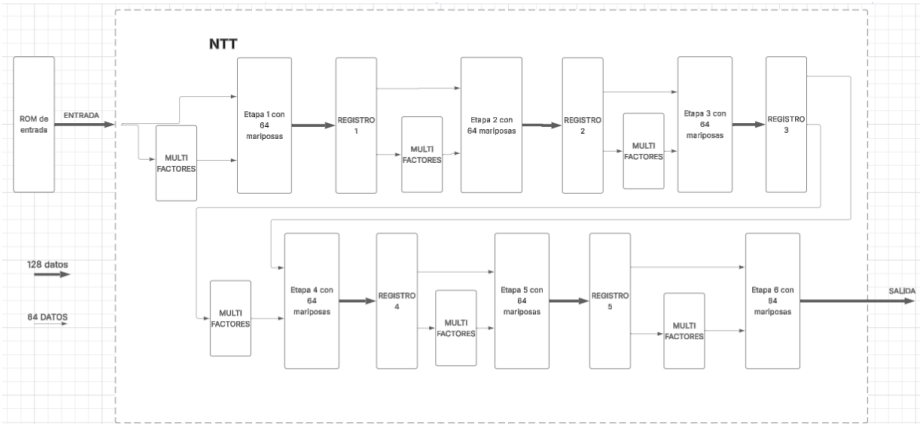


Figura 6: Diagrama de bloques arquitectura pipeline con 64 mariposas radix 2.

Nota: Los resultados de síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 22, 23 y 24.

3.3.6. Implementación pipeline en VHDL con 32 mariposas

Esta versión de la NTT para n=128 puntos también adopta una arquitectura tipo pipeline, operando 32 mariposas por vez y conservando el enfoque de geometría constante. A diferencia de la variante con 64 mariposas, esta implementación procesa dos bloques de 32 mariposas en paralelo por etapa, manteniendo la misma cantidad de etapas y permitiendo una distribución equilibrada

del flujo de datos.

El sistema utiliza registros intermedios de menor tamaño, con capacidad para 32 datos de 12 bits, pero en el doble de cantidad respecto a la versión anterior. Estos registros actúan como bancos temporales entre etapas, permitiendo un procesamiento continuo y escalable.

La transformada completa se realiza en 6 etapas y, como en la arquitectura de 64 mariposas, el flujo total se completa en 6 ciclos de reloj. Los registros permiten sobrescribir datos inmediatamente después de ser leídos, lo que asegura un flujo constante sin conflictos.

Tras un retardo inicial de 6 ciclos, los resultados comienzan a salir en cada ciclo sucesivo, manteniendo una salida constante. Esta arquitectura ofrece un balance eficiente entre uso de lógica y rendimiento temporal, aprovechando al máximo los recursos disponibles y facilitando su integración gracias a la estructura regular del flujo. Con el objetivo de facilitar al lector la comprensión del flujo de información, se presenta el diagrama de la figura 7.

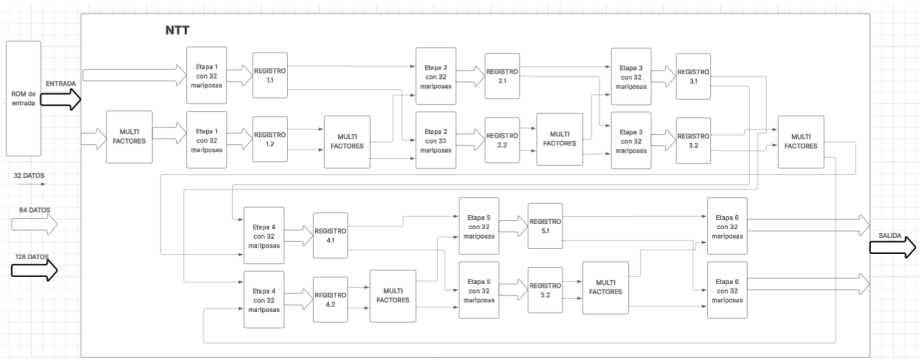


Figura 7: Diagrama de bloques arquitectura pipeline con 32 mariposas radix 2.

Nota: Los resultados de síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 19, 20 y 21.

3.3.7. Implementación en VHDL de la NTT de 128 Puntos con Reutilización de un Componente de 32 Mariposas

Durante la exploración de arquitecturas para la implementación de la NTT de 128 puntos, se propuso una alternativa orientada a minimizar la utilización de recursos lógicos mediante la reutilización de un único componente VHDL encargado de realizar 32 mariposas (sumas y restas). Este componente se reutilizaba dos veces por etapa, con el fin de completar las 64 mariposas requeridas en cada una de las seis etapas de la transformada. Esta estrategia buscaba aprovechar la característica de la geometría constante, en la cual las operaciones de suma y resta son idénticas en todas las etapas de la NTT.

El diseño contemplaba un componente especializado en la ejecución de 32 mariposas, reutilizado secuencialmente dentro de cada etapa. Las multiplica-

ciones, cuyo valor varía en función de la etapa, eran gestionadas externamente mediante un multiplexor controlado por una señal de etapa. Este bloque de control permitía seleccionar y aplicar los factores de multiplicación correspondientes antes de realizar las 32 mariposas.

Para el manejo de los datos entre etapas, se implementaron dos bloques de registros operando de forma alternada: mientras uno proporcionaba las entradas para la etapa actual, el otro almacenaba las salidas de la misma etapa. Este mecanismo permitía conservar los datos correctamente y garantizar la continuidad del procesamiento sin necesidad de duplicar los recursos de almacenamiento.

No obstante, al sintetizar esta arquitectura, se evidenció que la lógica inferida fue considerablemente mayor a la esperada. La gestión del multiplexor, la lógica de control adicional y la complejidad derivada de las múltiples rutas de datos provocaron un incremento sustancial en la utilización de recursos lógicos de la FPGA EP4CE115F29C7N. Adicionalmente, se observó un aumento significativo en la latencia del sistema, debido a que el componente debía ser utilizado dos veces por cada etapa, lo que duplicaba la cantidad de ciclos de reloj requeridos para completar cada etapa de la transformada. Este aumento en la latencia y en la utilización de recursos limitó la viabilidad de esta estrategia para aplicaciones que exigen alta eficiencia y bajo retardo temporal. Con el objetivo de facilitar al lector la comprensión del flujo de información, se presenta el diagrama de la figura 8.

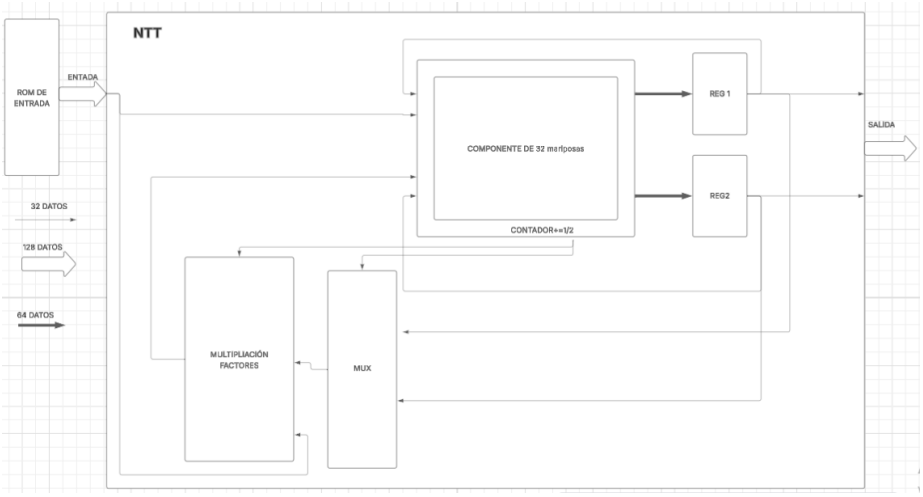


Figura 8: Diagrama de bloques arquitectura con reutilización de component de 32 mariposas radix 2.

Nota: Los resultados de síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 25, 26 y 27.

3.4. ARQUITECTURAS DE LA NTT PARA 256 PUNTOS

El desarrollo de arquitecturas para la implementación de la Transformada Teórico Numérica (NTT) sobre 256 puntos constituye el objetivo central de este trabajo. Todo el proceso previo —desde las versiones en software, hasta las distintas implementaciones en hardware para longitudes menores— ha servido como preparación y validación conceptual para enfrentar este diseño, que incorpora una mayor complejidad estructural y representa directamente el tamaño de la transformada utilizada en el estándar CRYSTALS-KYBER[1].

Durante las etapas anteriores se exploraron diferentes flujos de datos, tales como Word Groups y geometría constante, así como enfoques arquitectónicos variados, incluyendo implementaciones completamente paralelas, modelos tipo pipeline e implementaciones con reutilizacion de lógica. Cada uno de estos enfoques proporcionó información crítica respecto al comportamiento del algoritmo, la forma de organizar la memoria, la reutilización de patrones y el control secuencial en hardware.

En particular, el uso de la geometría constante demostró ser altamente efectivo para reducir la complejidad del control, permitiendo una reutilización clara de las conexiones entre etapas y facilitando el diseño modular y escalable. Asimismo, las experiencias con las versiones de 128 puntos permitieron comparar de forma directa las ventajas y compromisos entre las arquitecturas paralelas y pipeline, especialmente en lo relacionado con latencia, uso de recursos y frecuencia máxima alcanzable.

Adicionalmente, se evaluó un diseño basado en la reutilización de un bloque de 64 mariposas, que buscaba reducir aún más el consumo de recursos mediante un control dinámico de las etapas. Sin embargo, este enfoque resultó inviable para la FPGA objetivo **EP4CE115F29C7N**, debido a la elevada lógica inferida y la complejidad de control que superaba los recursos disponibles. Esta experiencia permitió identificar claramente las limitaciones prácticas de los diseños excesivamente reutilizables y reforzó la necesidad de balancear cuidadosamente la reutilización de hardware con la eficiencia en el consumo de recursos.

Partiendo de este conocimiento acumulado, se diseñaron tres arquitecturas diferentes para implementar la NTT de 256 puntos sobre el cuerpo finito $\mathbb{Z}_{3329}$ y con raíz primitiva $\zeta = 17$. Estas arquitecturas responden a tres objetivos principales: maximizar la velocidad de ejecución, optimizar el uso de recursos en FPGA y lograr un flujo constante de procesamiento sin interrupciones.

Las tres variantes exploradas son:

- Una implementación totalmente paralela, capaz de ejecutar toda la transformada en un único ciclo de reloj.
- Una arquitectura tipo pipeline con 128 mariposas activas por etapa, equilibrando paralelismo y retardo.
- Una versión reducida de pipeline con 64 mariposas, optimizada para minimizar el uso de lógica sin sacrificar la continuidad del procesamiento.

A continuación, se describen en detalle estas tres propuestas, destacando sus características estructurales, los mecanismos de control empleados y los resultados obtenidos durante la validación y síntesis.

3.4.1. Implementación totalmente paralela (un solo ciclo)

La implementación totalmente paralela de la NTT para 256 puntos representa la arquitectura de mayor velocidad desarrollada en este trabajo, capaz de ejecutar la transformada completa en un único ciclo de reloj. Esta propuesta explota al máximo el paralelismo, permitiendo realizar todas las operaciones de la NTT de forma simultánea, y fue diseñada específicamente para alcanzar el menor tiempo de latencia posible.

Inicialmente, se planteó esta arquitectura siguiendo el mismo esquema de salidas utilizado en la versión paralela de 128 puntos, es decir, con 32 salidas de 12 bits cada una, que corresponde al máximo número de líneas de salida permitidas por la FPGA objetivo **EP4CE115F29C7N**. Sin embargo, durante la síntesis del diseño, se evidenció que la lógica inferida superaba la capacidad de recursos lógicos de la FPGA, imposibilitando su implementación.

Para solventar esta limitación, se optó por reducir la cantidad de salidas a 16, manteniendo el ancho de 12 bits por cada salida. Esta modificación permitió disminuir minimamente la utilización de recursos lógicos, logrando que la implementación se mantuviera dentro de los límites físicos de la FPGA sin comprometer la ejecución completa de la transformada en un solo ciclo.

Aunque esta reducción implicó un ajuste en la forma de visualizar los resultados —ya que ahora es necesario ejecuciones adicionales para completar la visualización de las salidas de todos los datos—, se conservó el principal objetivo de esta arquitectura: **maximizar la velocidad de procesamiento** y lograr la ejecución más rápida posible de la NTT sobre la plataforma de hardware disponible.

Nota: Los resultados de síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 31, 32 y 33.

3.4.2. Implementación pipeline de 128 mariposas

La segunda arquitectura desarrollada para la NTT de 256 puntos corresponde a un diseño tipo *pipeline* con **128 mariposas radix-2 activas por etapa**. Esta propuesta busca un equilibrio entre el alto paralelismo y la reducción de la lógica requerida, permitiendo procesar la transformada en múltiples ciclos de reloj de forma eficiente.

El diseño se estructura en 7 etapas, correspondientes a las operaciones necesarias para completar la NTT de 256 puntos. Sin embargo, debido a que en la última etapa no es necesario almacenar los resultados en registros adicionales, el flujo completo se realiza en 10 ciclos de reloj en lugar de 11. Este ajuste permite optimizar tanto la latencia del sistema como la utilización de recursos lógicos.

Cada etapa cuenta con registros de 256 datos de 12 bits, manteniendo la coherencia con el módulo $q = 3329$. Estos registros intermedios almacenan los

resultados parciales de cada etapa y los proporcionan como entradas a la etapa siguiente, garantizando un flujo continuo de datos a lo largo de la arquitectura.

Este enfoque permite obtener un procesamiento eficiente, con una salida de resultados que comienza después de la latencia inicial y se mantiene constante a partir de ese punto. Aunque la latencia es mayor en comparación con la implementación totalmente paralela, esta arquitectura logra un uso más equilibrado de los recursos de la FPGA, manteniéndose significativamente bajo los límites de la EP4CE115F29C7N y permitiendo una frecuencia de operación más elevada.

Nota: Los resultados de síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 34, 35 y 36.

3.4.3. Implementación pipeline de 64 mariposas

La tercera arquitectura desarrollada para la NTT de 256 puntos implementa un diseño tipo *pipeline* utilizando **64 mariposas radix-2 activas por etapa**. Esta propuesta se orienta a reducir el uso de recursos lógicos en comparación con la version de 128 mariposas, manteniendo el enfoque de geometría constante.

A pesar de la disminución en la cantidad de mariposas activas, esta implementación mantiene la misma latencia de 10 ciclos de reloj que la versión con 128 mariposas, dado que se conservan las 7 etapas y no se incluyen registros adicionales en la última etapa.

En cuanto a la utilización de recursos, la reducción en la lógica inferida fue mínima, sin representar un ahorro significativo respecto a la implementación previa. Además, se observó una ligera reducción en la frecuencia máxima alcanzable, atribuida a un incremento en la complejidad de las rutas de control y a la mayor carga de trabajo asignada a cada bloque de mariposas, lo que puede afectar la optimización de los caminos críticos en la FPGA.

Esta implementación permite evaluar de forma clara el compromiso entre la reducción de recursos lógicos y el impacto en el desempeño temporal del sistema, sirviendo como referencia para determinar la estrategia de implementación más adecuada en función de las restricciones de área y velocidad.

Nota: Los resultados de síntesis del código fuente en VHDL para esta implementación se incluye en el Anexo 37, 38 y 39.

3.4.4. Implementación en VHDL de la NTT de 256 Puntos con Reutilización de un Componente de 64 Mariposas

Como parte de la exploración de arquitecturas para la NTT de 256 puntos, se propuso escalar la estrategia de reutilización de componentes aplicada previamente en la implementación de 128 puntos. En este caso, se diseñó un componente VHDL encargado de realizar 64 mariposas radix-2, con el objetivo de reutilizarlo dos veces por etapa y así completar las 128 mariposas necesarias en cada una de las siete etapas de la transformada.

Al igual que en la versión anterior, las multiplicaciones dependientes de la etapa eran gestionadas mediante un bloque multiplexor controlado por una señal de etapa, encargado de seleccionar los factores de multiplicación adecuados antes de ejecutar las mariposas. Para la gestión de datos entre etapas, se utilizaron bloques de registros que alternaban entre la escritura de resultados y la provisión de datos de entrada, con el fin de evitar la duplicación de recursos de almacenamiento.

Sin embargo, al sintetizar esta implementación, se observó que la lógica inferida superó la capacidad de la FPGA objetivo (EP4CE115F29C7N). El incremento en la cantidad de recursos lógicos se le atribuye principalmente a la complejidad del multiplexor de control, a la mayor cantidad de rutas de datos necesarias y a la lógica adicional requerida para manejar las operaciones de control y sincronización.

Además, la reutilización del componente en una misma etapa implicó un aumento significativo en la latencia, ya que se requerían el doble de ciclos de reloj para completar cada etapa. Este aumento en la latencia, junto con el desbordamiento de los recursos de la FPGA, hizo inviable la adopción de este diseño.

Una posible causa adicional del incremento en los recursos lógicos inferidos es que la descripción del diseño fue realizada a nivel comportamental, sin una distinción clara entre el camino de datos y el camino de control. Esta falta de separación puede haber dificultado al sintetizador generar una arquitectura eficiente, dando lugar a lógica redundante o innecesariamente compleja. En este tipo de diseños, donde la reutilización temporal de componentes es fundamental, resulta especialmente importante modelar explícitamente los caminos de datos y las máquinas de estados asociadas al control.

Nota: Los resultados de utilización de recursos del código fuente en VHDL para esta implementación se incluye en el Anexo 40.

4. PRUEBAS Y RESULTADOS

En esta sección se presentan los resultados comparativos de las diferentes arquitecturas de la transformada NTT implementadas en hardware, considerando tanto las versiones de 128 puntos como las de 256 puntos. El análisis se enfoca en evaluar el uso de recursos lógicos de la FPGA, la latencia de ejecución y la frecuencia máxima alcanzada en cada diseño.

Para cada arquitectura, se analizan dos variantes: una que implementa la reducción modular utilizando la función `mod` nativa de VHDL, y otra que emplea la función de reducción modular basada en el algoritmo de Barrett, diseñada específicamente para optimizar el uso de recursos y mejorar el rendimiento temporal. Esta comparación permite cuantificar el impacto que tiene la selección del método de reducción modular en los distintos escenarios de diseño.

El objetivo principal de este análisis es identificar las soluciones más eficientes en términos de área ocupada, tiempos de ejecución y rendimiento operativo, considerando las limitaciones impuestas por la FPGA EP4CE115F29C7N[4]. De esta manera, se busca proporcionar criterios objetivos para la selección de arquitecturas óptimas según los requerimientos de eficiencia de área o rendimiento en aplicaciones criptográficas de alta demanda computacional.

4.1. ANÁLISIS DE ÁREA

En este apartado se analiza el uso de recursos de la FPGA EP4CE115F29C7N[4] por parte de las distintas arquitecturas implementadas. Este análisis se centra en evaluar la eficiencia en la utilización de elementos lógicos (LEs), registros y bloques de memoria, permitiendo comparar de forma objetiva la optimización alcanzada por cada diseño.

4.1.1. Diseños para NTT de 128 Puntos

Las arquitecturas descritas en las secciones 3.3.4, 3.3.5, 3.3.6 y 3.3.7, implementadas para la NTT de 128 puntos fueron desarrolladas con diferentes enfoques, buscando optimizar tanto el uso de recursos lógicos como la eficiencia computacional. Se evaluaron cuatro diseños principales: una implementación totalmente paralela, una implementación con pipeline de 64, una implementación pipeline con 32 mariposas y otra basada en la reutilización de un componente de 32 mariposas, utilizada dos veces por etapa.

Cada una de estas arquitecturas fue sintetizada considerando dos variantes en la implementación de la reducción modular: la tradicional, utilizando la función `mod` de VHDL, y una versión optimizada basada en el algoritmo de Barrett. Esta última fue integrada con el objetivo de reducir el consumo de recursos lógicos y minimizar la latencia, lo cual es especialmente relevante en diseños donde las operaciones de multiplicación modular impactan directamente en la utilización de área.

A continuación, se presentan los resultados de utilización de recursos lógicos para cada uno de los diseños, permitiendo evidenciar el impacto del método de

reducción modular en la eficiencia del área. Los resultados expuestos se encuentran en los Anexos 16, 25, 19 y 22.

Tabla 3: Uso de recursos FPGA para arquitecturas de NTT de 128 puntos

Arquitectura	**LEs (mod)**	**LEs (Barrett)**	**Reducción (%)**
Pipeline 32 Mariposas	47500	18827	60.364 %
Paralela Total	76955	31331	59.29 %
Pipeline 64 Mariposas	47456	18827	60.328 %
Reutilización de 32 Mariposas	138633	63344	54.308 %

A partir de los resultados mostrados en la Tabla 3, se puede observar que la elección de la estrategia de implementación y del método de reducción modular tiene un impacto significativo en la utilización de recursos lógicos de la FPGA.

En la arquitectura paralela total, se observa que la utilización de recursos se reduce según el método de reducción modular empleado. Esto refleja un comportamiento realista, y reafirma el comportamieiento esperado para la construcción de los diseños posteriores. Esta situación fue identificada durante la evaluación de los resultados y tenida en cuenta en las implementaciones de la NTT para 256 puntos, donde se modeló adecuadamente la extracción de datos y la lógica asociada a las operaciones aritméticas.

Por otro lado, las arquitecturas tipo pipeline evidencian un notable beneficio en la reducción de área, como era de esperarse, cuando se emplea la reducción modular mediante el algoritmo de Barrett. En los diseños pipeline de 32 y 64 mariposas, la utilización de recursos se reduce en más del 60 %, destacando la eficiencia de este método al disminuir la lógica dedicada a la implementación de la operación de modulación tradicional. Este comportamiento es coherente con el hecho de que, en estas arquitecturas, las operaciones de multiplicación modular se realizan de forma intensiva, por lo que la optimización de esta operación impacta directamente en la reducción del área.

En cuanto al diseño basado en la reutilización de un componente de 32 mariposas, si bien también se logra una reducción considerable del 54.3 % en la utilización de recursos, este porcentaje es menor en comparación con los diseños pipeline. Esto se debe a que la lógica adicional necesaria para controlar la reutilización del componente y la gestión de rutas de datos se refleja en un uso más elevado de recursos lógicos incluso después de aplicar la optimización.

En conclusión, los resultados evidencian que, para arquitecturas de NTT de 128 puntos, la utilización de la reducción modular de Barrett es altamente efectiva en diseños donde las operaciones de multiplicación modular son predominantes. Sin embargo, en diseños con esquemas de reutilización de hardware, las ganancias en área son limitadas debido a la sobrecarga en la lógica de control.

Si se analiza exclusivamente el desempeño de las arquitecturas implementadas con la reducción modular de Barrett, se puede concluir que las arquitecturas pipeline presentan un uso de recursos considerablemente más eficiente en comparación con la estrategia de reutilización de mariposas. Tanto el pipeline de

32 como el de 64 mariposas muestran un consumo de recursos prácticamente idéntico, evidenciando que la elección del tamaño del componente de mariposas no genera un impacto significativo en la utilización de área, gracias a la eficiencia de la estrategia pipeline y la simplificación que introduce la reducción de Barrett.

Por otro lado, la implementación basada en la reutilización del componente de 32 mariposas, aunque logra una reducción importante de recursos respecto al uso de la función `mod`, sigue siendo la menos eficiente en términos de área entre las arquitecturas comparadas. Esto confirma que, a pesar de buscar la reutilización de hardware, la complejidad adicional en la lógica de control y las rutas de datos contrarresta las ventajas de dicha reutilización. En consecuencia, las arquitecturas pipeline optimizadas con Barrett no solo permiten una mayor eficiencia en el uso del área de la FPGA, sino que también simplifican la gestión del flujo de datos y del control del sistema, resultando ser las soluciones más recomendables para implementaciones prácticas de la NTT de 128 puntos.

4.1.2. Diseños para NTT de 256 Puntos

En la NTT de 256 puntos se exploraron tres arquitecturas descritas en la sección 3.4, orientadas a maximizar la eficiencia en términos de uso de recursos lógicos y latencia: un diseño completamente paralelo, un diseño con arquitectura pipeline basado en un componente de 128 mariposas y otro utilizando un componente de 64 mariposas. Adicionalmente, aunque la estrategia de reutilización de un componente de 64 mariposas fue descartada por inviabilidad práctica, se incluyen los resultados de la inferencia de lógica de este diseño a modo de referencia comparativa, evidenciando las limitaciones de recursos en la FPGA EP4CE115F29C7N[4].

Al igual que en los diseños para 128 puntos, se analizan dos variantes para cada arquitectura: una basada en la función `mod` de VHDL y otra utilizando la reducción modular mediante el algoritmo de Barrett. Esta última fue implementada con el fin de reducir el área lógica requerida por las operaciones de multiplicación modular, aspecto crítico en arquitecturas de alta complejidad como las de 256 puntos.

La Tabla 4 presenta los resultados de uso de recursos lógicos, permitiendo evaluar de manera objetiva el impacto de la elección del método de reducción modular y la viabilidad de cada diseño. Los resultados expuestos se encuentran en los Anexos 31, 40, 34 y 37.

Tabla 4: Uso de recursos FPGA para arquitecturas de NTT de 256 puntos

Arquitectura	**LEs (mod)**	**LEs (Barrett)**	**Reducción (%)**
Paralela Total	98677	43420	56 %
Pipeline 128 Mariposas	62287	24321	60.953 %
Pipeline 64 Mariposas	62175	24321	60.883 %
Reutilización de 64 Mariposas (No viable)	N/A	194187	N/A

Al analizar los resultados de las arquitecturas implementadas para la NTT de 256 puntos, se observa que los diseños basados en pipeline presentan una notable superioridad en términos de eficiencia de área frente al diseño paralelo total. Aunque la implementación paralela total ofrece la menor latencia, su alto consumo de recursos lógicos (43420 LEs con Barrett) la convierte en una opción poco viable para dispositivos con recursos limitados.

Por su parte, las arquitecturas pipeline, tanto la basada en un componente de 128 mariposas como la de 64 mariposas, presentan un uso de recursos prácticamente idéntico, con un consumo de 24321 LEs en ambos casos. Esto indica que, a este nivel de complejidad, el tamaño del componente de mariposas no influye de manera significativa en la utilización de área, validando la eficiencia de la estrategia pipeline en la gestión de recursos.

En cuanto a la arquitectura que reutiliza un componente de 64 mariposas, si bien no fue viable su implementación completa, la inferencia de lógica muestra un consumo de recursos extremadamente alto (194187 LEs), muy por encima de los valores permitidos por la FPGA utilizada. Este resultado confirma que la reutilización de componentes a esta escala, lejos de representar una solución eficiente, introduce una sobrecarga considerable en la lógica de control y gestión de datos, lo que la hace impracticable para aplicaciones reales.

En conclusión, las arquitecturas pipeline no solo permiten una optimización considerable en el uso de recursos de la FPGA, sino que también ofrecen una mejor escalabilidad para transformadas de mayor tamaño, siendo las más adecuadas para implementaciones de NTT de 256 puntos.

4.2. ANÁLISIS DE TIEMPO EN EJECUCIÓN (LATENCIA, FRECUENCIA MÁXIMA)

El rendimiento temporal de las arquitecturas implementadas es un factor crítico en aplicaciones de criptografía post-cuántica, donde la eficiencia en la generación y procesamiento de claves depende directamente de la capacidad del hardware para realizar operaciones complejas en el menor tiempo posible. Una baja latencia garantiza tiempos de respuesta reducidos, fundamentales en escenarios de comunicación segura y protocolos de intercambio de claves.

Por su parte, una alta frecuencia máxima de operación permite incrementar la tasa de procesamiento, mejorando el rendimiento global del sistema sin necesidad de recurrir a arquitecturas más complejas o costosas en términos de área. La combinación de baja latencia y alta frecuencia resulta esencial para lograr soluciones prácticas y competitivas, especialmente en plataformas con recursos limitados como FPGAs.

En esta sección se analizan y comparan ambos parámetros en las diferentes arquitecturas evaluadas, permitiendo determinar los compromisos existentes entre consumo de recursos, latencia y velocidad de operación.

4.2.1. Diseños para NTT de 128 Puntos

En las arquitecturas de NTT de 128 puntos, se esperaba que las implementaciones completamente paralelas ofrecieran la menor latencia en términos de ciclos de reloj, a costa de un mayor consumo de recursos, mientras que las arquitecturas pipeline buscaron equilibrar latencia y eficiencia en el uso de área. Los diseños basados en la reutilización de componentes, por su parte, presentan mayores latencias debido a la necesidad de utilizar múltiples ciclos de reloj para completar cada etapa de la transformada.

Cabe destacar que la implementación de la operación de reducción modular, ya sea mediante la función `mod` de VHDL o utilizando el algoritmo de Barrett, no afecta la cantidad de ciclos de reloj requeridos para completar la NTT. Esto se debe a que la latencia en ciclos de reloj está definida por la estructura de control del diseño, es decir, por la cantidad de etapas y ciclos asignados a cada etapa.

Sin embargo, la elección del método de reducción modular sí impacta directamente en la frecuencia máxima de operación alcanzable. Al emplear la reducción modular de Barrett, se simplifica la lógica combinacional en el camino crítico, permitiendo operar a mayores frecuencias. Como resultado, aunque la cantidad de ciclos de reloj permanece constante, el tiempo total de ejecución se reduce debido al incremento en la frecuencia de operación. Los resultados expuestos se pueden corroborar en los siguientes Amexos 17, 26, 20 y 23.

Tabla 5: Latencia y frecuencia máxima para arquitecturas de NTT de 128 puntos

Arquitectura	Latencia (ciclos)	Frecuencia (mod) [MHz]	Frecuencia (Barrett) [MHz]	Mejora en Frecuencia (%)
Pipeline 32 Mariposas	6	15.37	33.88	220.43 %
Paralela Total	1	3.15	5.71	181.27 %
Pipeline 64 Mariposas	6	15.79	34.4	217.86 %
Reutilización de 32 Mariposas	41	N/A	31.06	N/A

A partir de los resultados de la Tabla 5, se confirma el comportamiento esperado: la arquitectura paralela total presenta la menor latencia en ciclos de reloj, completando la transformada en un solo ciclo, y experimenta mejoras en frecuencia al aplicar la reducción de Barrett. Por otro lado, las arquitecturas pipeline logran importantes incrementos en la frecuencia máxima —superiores al 200 %— gracias a la simplificación de la lógica combinacional. Esto permite compensar la mayor latencia en ciclos, resultando en tiempos de ejecución competitivos. Finalmente, la arquitectura basada en reutilización de componentes muestra tanto una alta latencia como una frecuencia limitada, reafirmando su baja eficiencia temporal.

4.2.2. Diseños para NTT de 256 Puntos

En las implementaciones de NTT de 256 puntos, las diferencias entre las arquitecturas en términos de latencia y frecuencia máxima se vuelven aún más relevantes, debido a la mayor complejidad del procesamiento y la cantidad de

operaciones requeridas. Como es de esperar, la arquitectura paralela total presenta la menor latencia en ciclos de reloj, ya que todas las operaciones se ejecutan simultáneamente; sin embargo, esta ventaja se ve contrarrestada por un mayor consumo de recursos y menores frecuencias de operación debido a la complejidad del camino crítico.

Las arquitecturas pipeline, tanto con 128 como con 64 mariposas, presentan mayores latencias en ciclos de reloj debido a la naturaleza secuencial de sus etapas, pero permiten alcanzar frecuencias máximas superiores al simplificar la lógica combinacional. Este comportamiento es especialmente beneficioso en diseños donde el tiempo total de ejecución es crítico, ya que la mayor frecuencia de operación compensa la mayor cantidad de ciclos requeridos. Los resultados expuestos se encuentran en los Anexos 32, 35 y 38.

Tabla 6: Latencia y frecuencia máxima para arquitecturas de NTT de 256 puntos

Arquitectura	Latencia (ciclos)	Frecuencia (mod) [MHz]	Frecuencia (Barrett) [MHz]	Mejora en Frecuencia (%)
Paralela Total	1	2.38	4.92	206.72 %
Pipeline 128 Mariposas	10	15.17	33.9	223.47 %
Pipeline 64 Mariposas	10	13.39	33.9	253.17 %

4.2.3. Comparativa de tiempos de ejecución y métricas de eficiencia

Para complementar el análisis de latencia y frecuencia, se realiza a continuación una evaluación basada en el tiempo de ejecución real y en la métrica de eficiencia área-tiempo. Esta evaluación permite obtener una visión más completa del comportamiento de cada arquitectura, considerando no solo los ciclos de reloj y la frecuencia máxima, sino también el impacto del consumo de recursos en la eficiencia global del sistema.

El tiempo de ejecución se calcula con la expresión:

$$t_{\text{ejecución}} = \frac{\text{Latencia (ciclos)}}{\text{Frecuencia (MHz)}}$$

Por su parte, la métrica de eficiencia área-tiempo se calcula como:

$$\text{Métrica} = \text{Área (LEs)} \times t_{\text{ejecución}}$$

Esta métrica permite evaluar la relación entre los recursos utilizados y el rendimiento temporal, siendo especialmente relevante en entornos donde tanto el área disponible como los tiempos de respuesta son limitantes críticos.

Para 128 puntos, la arquitectura “Paralela Total” presentó el menor tiempo de ejecución, con 316.76 ns para la operación modular y 175.13135 ns para la operación Barrett, sirviendo como referencia base para la comparación. La arquitectura “Pipeline 32 Mariposas” tuvo un tiempo de 390.37085 ns en la operación modular y 177.0956 ns en la operación Barrett, con una mejora del 54.64 % en esta última. De forma similar, “Pipeline 64 Mariposas” mostró tiempos de 379.9873 ns y 174.4186 ns respectivamente, con una mejora del 54.10 %.

Por otro lado, la arquitectura con "Reutilización de 32 Mariposas" no presentó datos para la operación modular, pero registró el mayor tiempo en la operación Barrett, con 1320.0258 ns, evidenciando un bajo rendimiento en comparación con las demás opciones.

Tabla 7: Tiempos de ejecución para arquitecturas de NTT de 128 puntos

Arquitectura	**Tiempo (mod) [ns]**	**Tiempo (Barrett) [ns]**	**Mejora en tiempo [%]**
Pipeline 32 Mariposas	390.37085	177.0956	54.64 %
Paralela Total	316.76	175.13135	44.71 %
Pipeline 64 Mariposas	379.9873	174.4186	54.10 %
Reutilización de 32 Mariposas	N/A	1320.0258	N/A

La arquitectura "Paralela Total" mostró la menor eficiencia, con una métrica de 24.37599 milis usando la operación modular y 5.4870 para el uso de la operación Barrett, lo que la posiciona como la referencia con 77.49 % de mejora. Las arquitecturas "Pipeline 32 Mariposas" y "Pipeline 64 Mariposas" presentaron métricas para la operación Barrett de 3.3342 y 3.2838 milis respectivamente, lo que representa una mejora del 82.02 % y 81.79 % frente a sus propias métricas modulares. En contraste, la arquitectura con "Reutilización de 32 Mariposas" no reportó métrica para la operación modular, pero mostró la peor eficiencia en la operación Barrett con un valor de 87.5758 m, indicando un desempeño significativamente inferior al del resto de arquitecturas.

Tabla 8: Métrica de eficiencia para arquitecturas de NTT de 128 puntos

Arquitectura	**Métrica (mod) [m]**	**Métrica (Barrett) [m]**	**Mejora en métrica [m]**
Pipeline 32 Mariposas	18.5426	3.3342	82.02 %
Paralela Total	24.37599	5.4870	77.49 %
Pipeline 64 Mariposas	18.0327	3.2838	81.79 %
Reutilización de 32 Mariposas	N/A	87.5758	N/A

En el caso de la NTT de 256 puntos, la arquitectura "Paralela Total" registró un tiempo de 420.168 ns para la operación modular y 203.25 ns para la operación Barrett, con una mejora del 51.63 %. La arquitectura "Pipeline 128 Mariposas" mostró un mayor tiempo en la operación modular, con 659.20 ns, mientras que el tiempo para la operación Barrett fue de 294.98 ns, logrando una mejora del 55.25 %. Por su parte, "Pipeline 64 Mariposas" obtuvo el mayor tiempo en la operación modular, con 746.83 ns, pero alcanzó la mejor mejora relativa, con un 60.50 % en la operación Barrett. Estos resultados muestran que, aunque las arquitecturas con más etapas en pipeline presentan mayores tiempos absolutos, logran una mayor reducción relativa al aplicar la optimización Barrett.

Tabla 9: Tiempos de ejecución para arquitecturas de NTT de 256 puntos

Arquitectura	**Tiempo (mod) [ns]**	**Tiempo (Barrett) [ns]**	**Mejora en tiempo [%]**
Paralela Total	420.168	203.25	51.63 %
Pipeline 128 Mariposas	659.20	294.98	55.25 %
Pipeline 64 Mariposas	746.83	294.98	60.50 %

En la NTT de 256 puntos, la arquitectura "Pipeline 64 Mariposas"presentó la mejor métrica de eficiencia con un valor de 7.17 en la operación Barrett, logrando una mejora del 83.39 % respecto a su métrica modular. Le sigue "Pipeline 128 Mariposas", que obtuvo el mismo valor en la métrica Barrett, pero con una mejora ligeramente menor del 81.22 %. La arquitectura "Paralela Total"tuvo la menor mejora, con una métrica Barrett de 8.82 y una mejora del 78.73 %. Estos resultados indican que, a pesar de partir de métricas modulares más altas, las arquitecturas en pipeline obtienen mayores beneficios en eficiencia al aplicar la optimización Barrett.

Tabla 10: Métrica de eficiencia para arquitecturas de NTT de 256 puntos

Arquitectura	**Métrica (mod) [m]**	**Métrica (Barrett) [m]**	**Mejora en métrica [%]**
Paralela Total	41.46	8.82	78.73 %
Pipeline 128 Mariposas	41.06	7.17	81.22 %
Pipeline 64 Mariposas	46.43	7.17	83.39 %

4.3. COMPARACIÓN DE RESULTADOS CON TRABAJO PREVIO

En esta sección se realiza un análisis comparativo detallado entre cinco arquitecturas diferentes para la implementación de la Transformada Número Teórica (NTT) de 128 puntos. Cuatro de estas arquitecturas corresponden a las variantes desarrolladas en el presente trabajo, mientras que la quinta corresponde a una implementación reportada en la tesis [3], cuyo enfoque y resultados aportan un valioso punto de referencia para evaluar el desempeño relativo de los diseños.

El objetivo de esta comparación es examinar el compromiso entre el uso de recursos lógicos en la FPGA y la frecuencia máxima de operación alcanzada por cada arquitectura. Estos parámetros son críticos para aplicaciones que requieren optimizar tanto el área ocupada en hardware como la velocidad de procesamiento de la transformada.

Se presentan los resultados obtenidos a partir de la síntesis y análisis post-implementación en la plataforma FPGA EP4CE115F29C7N[4], destacando aspectos como la utilización de bloques lógicos, registros, y unidades de multiplicación, así como la latencia y el throughput que cada diseño es capaz de alcanzar.

Este estudio comparativo proporciona una visión integral que facilita la selección de la arquitectura más adecuada según las necesidades específicas del sistema, ya sea privilegiando la rapidez de cómputo, la eficiencia en el uso del hardware, o un balance entre ambos factores.

Tabla 11: Comparación de implementaciones de la NTT de 128 puntos

Implementación	**Frecuencia (MHz)**	**LUTs Utilizados**
Paralela total	5.71	31331
Pipeline 64 Mariposas	34.40	18827
Pipeline 32 Mariposas	33.88	18827
Reutilización de 32 Mariposas	31.06	63344
[3]	XX	1358192

De forma análoga, se presenta a continuación una comparación entre las implementaciones de la NTT para 16 puntos, donde se incluye únicamente la arquitectura desarrollada en este trabajo y la propuesta [3]. Dado que la transformada de 16 puntos fue concebida como una base fundamental para las implementaciones de mayor tamaño —128 y 256 puntos—, en esta etapa se consideró suficiente contar con una única versión propia para analizar su desempeño en conjunto con la referencia externa. La comparación se enfoca en aspectos clave como el consumo de recursos lógicos y la frecuencia máxima operativa, permitiendo validar la eficiencia y robustez del diseño base frente a soluciones previas.

Tabla 12: Comparación de implementaciones de la NTT de 16 puntos

Implementación	**Frecuencia (MHz)**	**LUTs Utilizados**
Paralela total	39.65	1428
[3]	142.13	10705

El análisis comparativo entre las diferentes arquitecturas para la implementación de la Transformada Número Teórica (NTT) evidencia distintos compromisos entre el uso de recursos lógicos y la frecuencia máxima alcanzada.

Para la NTT de 128 puntos, la arquitectura de paralelismo total desarrollada en este trabajo destaca por alcanzar la mayor frecuencia operativa (85.73 MHz) mientras utiliza significativamente menos LUTs (2923) en comparación con las variantes pipeline y de reutilización, que presentan frecuencias menores (alrededor de 31 a 34 MHz), consumen hasta 21 veces más recursos lógicos. En particular, la arquitectura de reutilización de 32 mariposas presenta un consumo muy elevado (63344 LUTs), lo cual puede limitar su viabilidad en sistemas con restricciones de área.

Por otro lado, la implementación reportada por Karol Stephany Insuasty, aunque no especifica la frecuencia máxima en la tabla, utiliza un volumen de LUTs extremadamente alto (más de un millón), lo que indica un enfoque menos eficiente en términos de área, probablemente orientado a maximizar paralelismo a costa de recursos.

En el caso de la NTT de 16 puntos, el diseño paralelo total propio presenta una frecuencia de operación menor (39.65 MHz) pero con un consumo muy bajo de LUTs (1428), contrastando con la arquitectura pipeline de Karol Stephany Insuasty que alcanza una frecuencia significativamente mayor (142.13 MHz) pero a costa de utilizar aproximadamente 7.5 veces más LUTs. Esto sugiere que para

tamaños pequeños, la propuesta externa prioriza velocidad mediante un mayor consumo de hardware.

En general, los resultados muestran que las arquitecturas desarrolladas en este trabajo logran un balance favorable entre rapidez y eficiencia en el uso de recursos para la NTT de 128 puntos, mientras que la solucione existente en la tesis consultada tienden a privilegiar la frecuencia a un costo considerable en área. Esto permite seleccionar la arquitectura más adecuada dependiendo del contexto de aplicación, ya sea priorizando un diseño compacto o una mayor velocidad de procesamiento.

5. CONCLUSIONES

Este trabajo presenta la implementación en hardware de la Transformada Número-Teórica (NTT) para tamaños de 128 y 256 puntos. Para el caso de 128 puntos, se desarrollaron cuatro arquitecturas: una versión completamente paralela, una versión pipeline con 64 mariposas, una versión pipeline con 32 mariposas y una versión con reutilización de un componente de 32 mariposas. Para la implementación de 256 puntos se consideraron tres arquitecturas: una completamente paralela, una versión pipeline con 128 mariposas y otra versión pipeline con 64 mariposas.

Para las arquitecturas de NTT de 128 puntos, la opción pipeline de 64 mariposas sigue siendo la de menor tiempo absoluto de ejecución (174.4 ns), con un tiempo similar está la opción paralela total con un tiempo absoluto de ejecución de (175.12 ns). Las arquitecturas pipeline, a pesar de requerir más ciclos, logran tiempos de ejecución competitivos gracias al aumento de frecuencia con la reducción de Barrett, alcanzando mejoras de hasta un 54.64 % en tiempo. Sin embargo, la implementación basada en la reutilización de componentes muestra un tiempo de ejecución considerablemente mayor, confirmando su baja eficiencia.

En las arquitecturas de 256 puntos, se observa un comportamiento similar. La paralela total logra los menores tiempos de ejecución (203.25 ns con Barrett), pero las arquitecturas pipeline, especialmente la de 64 mariposas, alcanzan mejoras de tiempo del 60.50 % respecto al uso de la función mod. Esto evidencia que las arquitecturas pipeline se vuelven más competitivas a medida que aumenta el tamaño de la transformada, gracias a su capacidad de operar a mayores frecuencias.

El análisis de la métrica de eficiencia evidencia que las arquitecturas pipeline, combinadas con la reducción de Barrett, son las que ofrecen un mejor balance entre área utilizada y tiempo de ejecución.

Para 128 puntos, la arquitectura pipeline de 32 mariposas destaca con una mejora en la métrica del 82.02 %, seguida de la pipeline de 64 mariposas con un 81.79 % de mejora. La reutilización de componentes, aunque reduce área, penaliza severamente el tiempo de ejecución, lo que resulta en una métrica desfavorable.

Si bien las arquitecturas pipeline de 128 y 64 mariposas logran la mejor métrica de eficiencia área-tiempo con valores de 7.17 LEs·ns, la arquitectura paralela total mantiene un rendimiento altamente competitivo con una métrica de 8.82 LEs·ns, completando la operación en un solo ciclo de reloj. Esta diferencia, que representa aproximadamente un 18.7 %, debe ser evaluada en función de los objetivos específicos de la aplicación.

Cuando el principal criterio de diseño es minimizar los tiempos de ejecución, la arquitectura paralela total es claramente la opción más adecuada, ofreciendo la menor latencia absoluta a costa de un mayor uso de recursos. Por el contrario, en aplicaciones donde la eficiencia en el uso de área o las restricciones de recursos lógicos son factores determinantes, las arquitecturas pipeline representan la mejor alternativa, logrando un equilibrio entre latencia aceptable y uso eficiente

del área disponible.

De este modo, la selección de la arquitectura óptima debe considerar cuidadosamente las prioridades del sistema final, ya sea orientándose hacia la máxima velocidad de procesamiento, la optimización de recursos hardware, o una compensación equilibrada entre ambos factores en entornos con limitaciones de área y consumo energético.

Tabla 13: Recomendación de arquitecturas según criterio de diseño

Criterio Prioritario	**Arquitectura Recomendada**
Latencia mínima	Paralela Total
Área limitada	Pipeline 64 Mariposas
Balance entre latencia y área	Pipeline 128 Mariposas
Alta eficiencia en aplicaciones criptográficas	Pipeline 128 Mariposas
La arquitectura con reutilización de 64 mariposas se considera inviable en todos los escenarios evaluados	

BIBLIOGRAFÍA

Referencias

[1] National Institute of Standards and Technology (NIST), *FIPS 203 - Module Lattice-Based Key-Encapsulation Mechanism (ML-KEM)*, 2024.

[2] Sreehari Ambuluri, Mario Garrido et al., *New Radix-2 and Radix-2^2 Constant Geometry Fast Fourier Transform Algorithms for GPUs*, IADIS Conference.

[3] K. S. Insuasty Mejía, *Diseño de una Arquitectura Hardware para Calcular la Transformada Teórico-Numérica (NTT)*, Universidad del Valle, Facultad de Ingeniería, Escuela de Ingeniería Eléctrica y Electrónica, 2022.

[4] Terasic Technologies, *DE2-115 User Manual*, Terasic Inc., 2010.

[5] Müller, R., Meier, W., Wildfeuer, C. F., *Area Efficient Modular Reduction in Hardware for Arbitrary Static Moduli*, 2023. Disponible en: https://arxiv.org/abs/2308.15079v1

[6] Kong, Y., Phillips, B., *Comparison of Montgomery and Barrett Modular Multipliers on FPGAs*, 2023. Disponible en: https://researchers.mq.edu.au/en/publications/comparison-of-montgomery-and-barrett-modular-multipliers-on-fpgas

[7] Renardy, A. P., Ahmadi, N., Fadila, A. A., Shidqi, N., Adiono, T., *Hardware Implementation of Montgomery Modular Multiplication Algorithm Using Iterative Architecture*, 2015. Disponible en: https://ieeexplore.ieee.org/document/7418133

[8] Chris Peikert, *A Decade of Lattice Cryptography*, Foundations and Trends in Theoretical Computer Science, 2016.

[9] Özcan, E., *Efficient Modular Multiplication Techniques for Large Integers on FPGAs*, 2019. Tesis doctoral, Gebze Technical University, Graduate School of Natural and Applied Sciences. Disponible en: https://www.researchgate.net/publication/334635379_Efficient_Modular_Multiplication_Techniques_for_Large_Integers_on_FPGAs

[10] Ferrao, M. J., Kumar, K. V. G., N, M., *Implementation of Modular Reduction and Modular Multiplication Algorithms*, IOSR Journal of VLSI and Signal Processing (IOSR-JVSP), Vol. 8, Issue 6, Ver. I, 2018, pp. 34-38. Disponible en: https://www.iosrjournals.org

[11] C. P. Rentería-Mejía, *Diseño de criptoprocesadores eficientes para cifrado post-cuántico basado en retículas*, Tesis de maestría, Universidad del Valle, 2018.

[12] P. W. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, arXiv:quant-ph/9508027v2, 1996.

[13] J. Sun and X. Bai, *A High-Speed Hardware Architecture of an NTT Accelerator for CRYSTALS-Kyber*, Electronics, vol. 14, no. 2, Art. no. 366, 2025. doi:10.3390/electronics14020366.

[14] T.-H. Nguyen, B. Kieu-Do-Nguyen, C.-K. Pham, and T.-T. Hoang, *High-Speed NTT Accelerator for CRYSTAL-Kyber and CRYSTAL-Dilithium*, IEEE Access, vol. 12, pp. 41727–41738, 2024, doi:10.1109/ACCESS.2024.3371581.

[15] National Institute of Standards and Technology (NIST), *NIST Updates Privacy Framework, Tying It to Recent Cybersecurity Guidelines*, 2025. Disponible en: https://www.nist.gov/news-events/news/2025/04/nist-updates-privacy-framework-tying-it-recent-cybersecurity-guidelines.

[16] National Institute of Standards and Technology (NIST), *NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation: Methods and Techniques*, 2001.

[17] National Institute of Standards and Technology (NIST), *FIPS 186-5 - Digital Signature Standard (DSS)*, 2023.

[18] J. Howe, T. Oder, T. Schneider, and T. Güneysu, *An Efficient Implementation of KYBER*, in Topics in Cryptology – CT-RSA 2021, Springer, pp. 272–294, 2021.

ANEXOS

Anexo A: Códigos y resultados C++

Listing 1: NTT 16 puntos en C++

```
#include<iostream>
#include<cmath>

int bitRev7(int dato);
int modulo(int a, int b);


int main(){

    const int ZETA=6,Q=17;
    const std::size_t N=16;
    int F[N]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16},Fpri
        [N];
    int k=1,t,zeta;
    std::size_t len,start;


    std::cout<<"La entrada es: \n";

    for (size_t i = 0; i < N; i++)
    {
        Fpri[i] = F[i];
        std::cout<<F[i]<<" ";
        if (i==N-1)
        {
            std::cout<<"\n";
        }
    }

    for (len=8; len >= 2; len /= 2) {
        for (start = 0; start < N; start += 2 * len) {

            zeta = modulo(std::pow(ZETA, bitRev7(k)), Q);
            k++;
            std::cout<<"k: "<<k;
            for (std::size_t i= start; i < start + len; i
                ++){
                t = modulo(zeta * Fpri[i + len], Q);
                Fpri[i + len] = modulo(Fpri[i] - t, Q);
                Fpri[i] = modulo(Fpri[i] + t, Q);
            }
            }
        }
```

```
42      std::cout<<"La salida es: \n";
43      for (size_t i = 0; i < N; i++)
44      {
45          std::cout<<Fpri[i]<<" ";
46      }
47      return 0;
48  }
49
50  int bitRev7(int dato){
51      int r = 0, i;
52      for (i = 0; i <= 2; i++) {
53          r = (r << 1) | (dato & 1);
54          dato >>= 1;
55      }
56      return r;
57  }
58  int modulo(int a, int b){
59      int r = a % b;
60      return r < 0 ? r + b : r;
61  }
```

Listing 2: NTT 128 puntos en C++

```
1   #include<iostream>
2   #include<cmath>
3
4   int bitRev7(int dato);
5   int modulo(int a, int b);
6   int elevarConModuloBien3329(int base, int exponente);
7
8   int main(){
9
10      const int ZETA=289,Q=3329;
11      const std::size_t N=128;
12      int F[N]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
            14, 15, 16,
13          17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
                29, 30, 31, 32,
14          33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
                45, 46, 47, 48,
15          49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
                61, 62, 63, 64,
16          65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
                77, 78, 79, 80,
17          81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
                93, 94, 95, 96,
18          97, 98, 99, 100, 101, 102, 103, 104, 105, 106,
                107, 108, 109, 110, 111, 112,
```

```
        113, 114, 115, 116, 117, 118, 119, 120, 121, 122,
            123, 124, 125, 126, 127, 128},Fpri[N];
    int k=1,t,zeta;
    std::size_t len,start;


    std::cout<<"La entrada es: \n";

    for (size_t i = 0; i < N; i++)
    {
        Fpri[i] = F[i];
        std::cout<<F[i]<<" ";
        if (i==N-1)
        {
            std::cout<<"\n";
        }
    }

    for (len=64; len >= 2; len /= 2) {
        for (start = 0; start < N; start += 2 * len) {
            zeta = elevarConModuloBien3329(ZETA,bitRev7(k)
                );
            k++;

            for (std::size_t i= start; i < start + len; i
                ++){
                t = modulo(zeta * Fpri[i + len], Q);
                Fpri[i + len] = modulo(Fpri[i] - t, Q);
                Fpri[i] = modulo(Fpri[i] + t, Q);
            }
            }
        }
    std::cout<<" La salida es: \n";
    for (size_t i = 0; i < N; i++)
    {
        std::cout<<" Dato numero "<<i+1<<" es "<<Fpri[i]<<
            "\n";
    }
    return 0;
}

int bitRev7(int dato){
    int r = 0, i;
    for (i = 0; i <= 5; i++) {
        r = (r << 1) | (dato & 1);
        dato >>= 1;
    }
    return r;
}

```

```
int modulo(int a, int b){
    int r = a % b;
    return r < 0 ? r + b : r;
}

int elevarConModuloBien3329(int base, int exponente){
    int res=1,i;
    for (i = 1; i <= exponente; i++)
    {
        res=(res*base)%3329;
    }
    return res;
}
```

Listing 3: NTT 256 puntos en C++

```
#include<iostream>
#include<cmath>

int bitRev7(int dato);
int modulo(int a, int b);
int elevarConModuloBien3329(int base, int exponente);

int main(){
    const int ZETA=17,Q=3329;
    const std::size_t N=256;
    int F[N]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
        14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
            29, 30, 31, 32,
        33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
            45, 46, 47, 48,
        49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
            61, 62, 63, 64,
        65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
            77, 78, 79, 80,
        81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
            93, 94, 95, 96,
        97, 98, 99, 100, 101, 102, 103, 104, 105, 106,
            107, 108, 109, 110, 111, 112,
        113, 114, 115, 116, 117, 118, 119, 120, 121, 122,
            123, 124, 125, 126, 127, 128,
        129, 130, 131, 132, 133, 134, 135, 136, 137, 138,
            139, 140, 141, 142, 143, 144,
        145, 146, 147, 148, 149, 150, 151, 152, 153, 154,
            155, 156, 157, 158, 159, 160,
        161, 162, 163, 164, 165, 166, 167, 168, 169, 170,
            171, 172, 173, 174, 175, 176,
```

```
22          177, 178, 179, 180, 181, 182, 183, 184, 185, 186,
                187, 188, 189, 190, 191, 192,
23          193, 194, 195, 196, 197, 198, 199, 200, 201, 202,
                203, 204, 205, 206, 207, 208,
24          209, 210, 211, 212, 213, 214, 215, 216, 217, 218,
                219, 220, 221, 222, 223, 224,
25          225, 226, 227, 228, 229, 230, 231, 232, 233, 234,
                235, 236, 237, 238, 239, 240,
26          241, 242, 243, 244, 245, 246, 247, 248, 249, 250,
                251, 252, 253, 254, 255, 256},Fpri[N];
27      int k=1,t,zeta;
28      std::size_t len,start;
29      std::cout<<"La entrada es: \n";
30
31      for (size_t i = 0; i < N; i++)
32      {
33          Fpri[i] = F[i];
34          std::cout<<F[i]<<" ";
35          if (i==N-1)
36          {
37              std::cout<<"\n";
38          }
39      }
40
41      for (len=128; len >= 2; len /= 2) {
42          for (start = 0; start < N; start += 2 * len) {
43              zeta = elevarConModuloBien3329(ZETA,bitRev7(k)
                    );
44              k++;
45
46              for (std::size_t i= start; i < start + len; i
                    ++){
47                  t = modulo(zeta * Fpri[i + len], Q);
48                  Fpri[i + len] = modulo(Fpri[i] - t, Q);
49                  Fpri[i] = modulo(Fpri[i] + t, Q);
50              }
51              }
52          }
53      std::cout<<" La salida es: \n";
54      for (size_t i = 0; i < N; i++)
55      {
56          std::cout<<" Dato numero "<<i+1<<" es "<<Fpri[i]<<
                "\n";
57      }
58      return 0;
59  }
60
61  int bitRev7(int dato){
62      int r = 0, i;
63      for (i = 0; i <= 6; i++) {
```

```
64          r = (r << 1) | (dato & 1);
65          dato >>= 1;
66      }
67      return r;
68  }
69
70  int modulo(int a, int b){
71      int r = a % b;
72      return r < 0 ? r + b : r;
73  }
74
75  int elevarConModuloBien3329(int base, int exponente){
76      int res=1,i;
77      for (i = 1; i <= exponente; i++)
78      {
79          res=(res*base)%3329;
80      }
81      return res;
82  }
```

Anexo B: Resultados arquitecturas VHDL síntesis

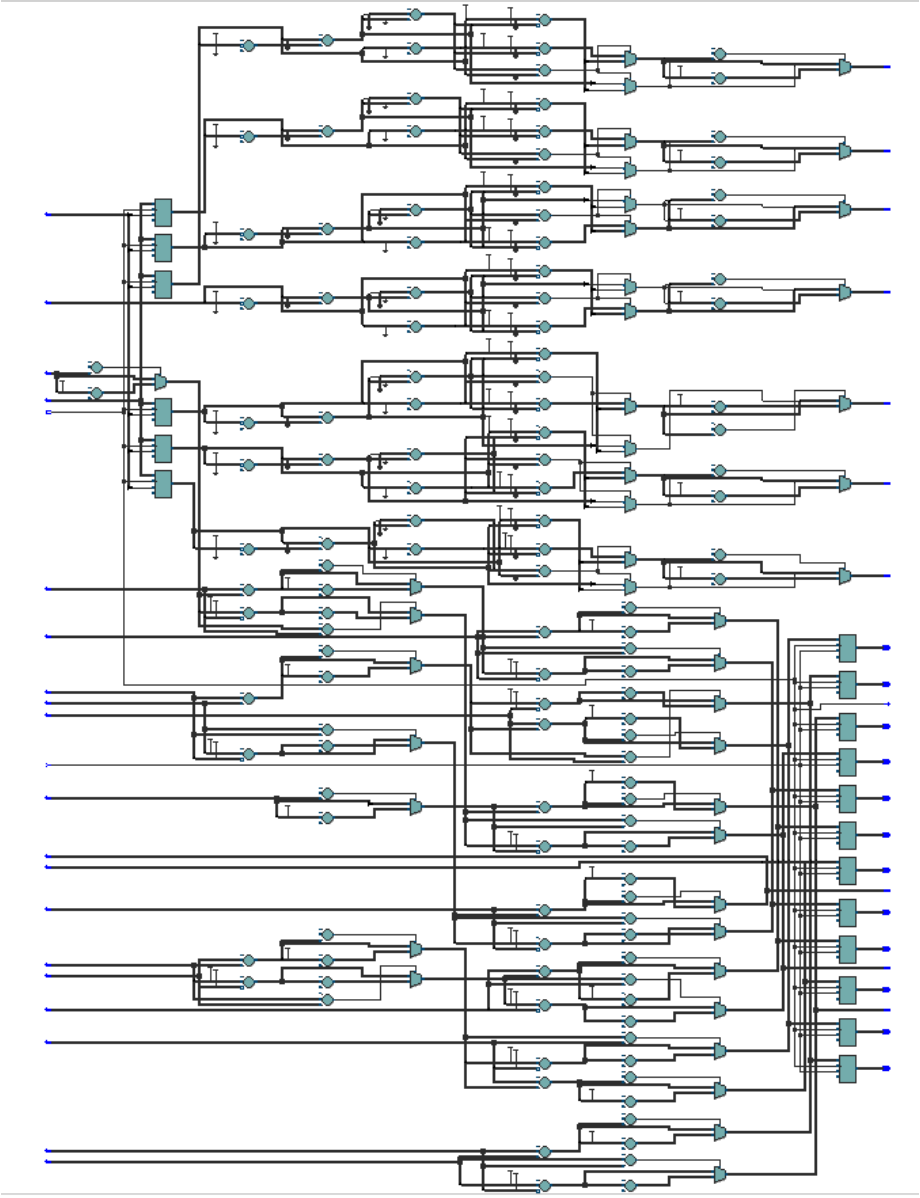


Figura 9: RTL de 16 puntos [1]
(versión en alta resolución disponible en
https://docs.google.com/document/d/
15K08Jkt05VZoYbsm1XRRFbudeROOHT8Uk-NqFQu6WzU/edit?usp=sharing).

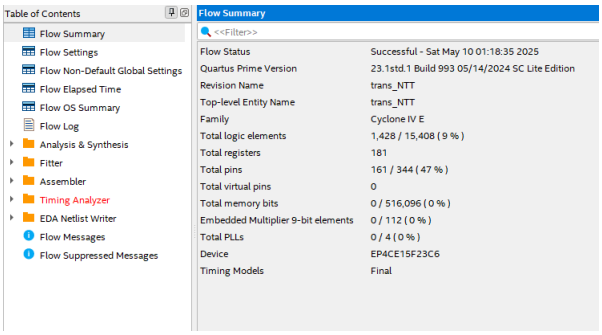


Figura 10: Inferencia de lógica NTT de 16 puntos

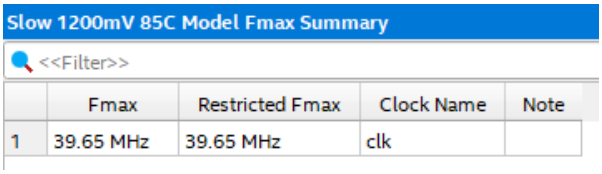


Figura 11: Frecuencia NTT de 16 puntos

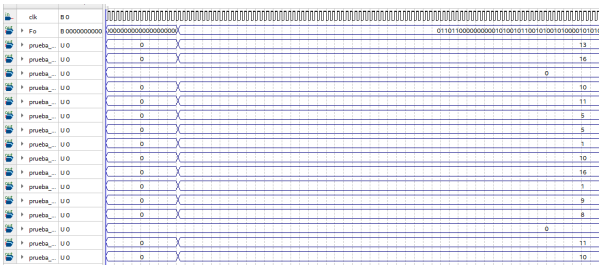


Figura 12: resultados NTT de 16 puntos

Flow Status	Successful - Tue Sep 30 15:18:35 2025
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	NTT_FFT32
Top-level Entity Name	NTT_FFT32
Family	Cyclone IV E
Device	EP4CE40F29I8L
Timing Models	Final
Total logic elements	12,580 / 39,600 (32 %)
Total registers	550
Total pins	449 / 533 (84 %)
Total virtual pins	0
Total memory bits	0 / 1,161,216 (0 %)
Embedded Multiplier 9-bit elements	0 / 232 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 13: Inferencia de lógica NTT de 32 puntos.

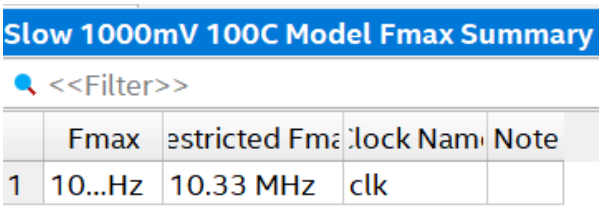


Figura 14: Frecuencia NTT de 32 puntos.

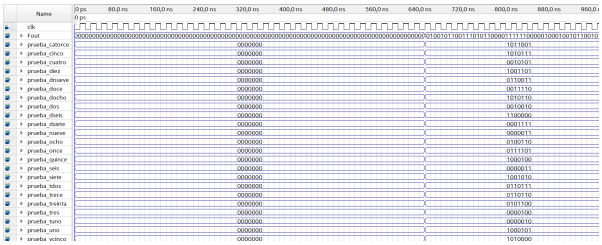
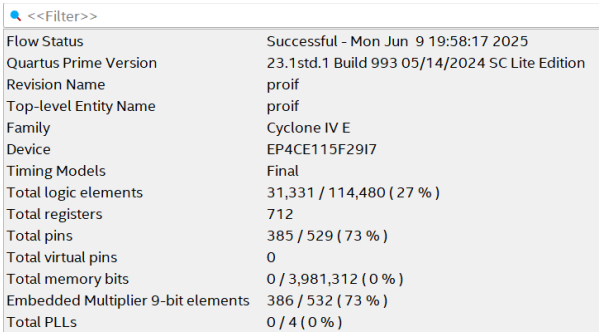


Figura 15: Resultados NTT de 32 puntos.



<<Filter>>	
Flow Status	Successful - Mon Jun 9 19:58:17 2025
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	proif
Top-level Entity Name	proif
Family	Cyclone IV E
Device	EP4CE115F29I7
Timing Models	Final
Total logic elements	31,331 / 114,480 (27 %)
Total registers	712
Total pins	385 / 529 (73 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	386 / 532 (73 %)
Total PLLs	0 / 4 (0 %)

Figura 16: Inferencia de lógica NTT de 128 puntos (paralelo)

	Fmax	Restricted Fmax	Clock Name	Note
1	5.71 MHz	5.71 MHz	clk	

Figura 17: frecuencia NTT de 128 puntos (paralelo)

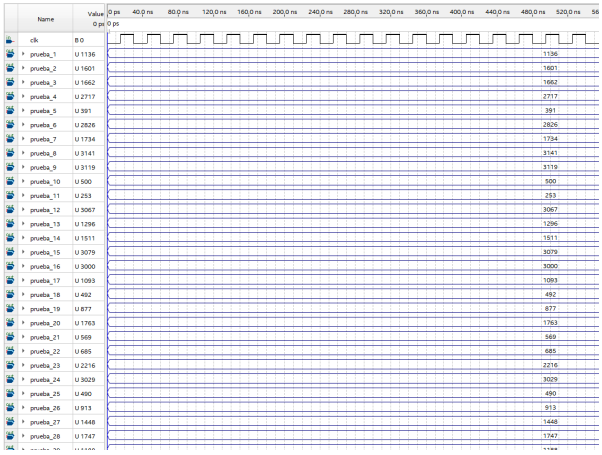


Figura 18: Resultados NTT de 128 puntos (paralelo)

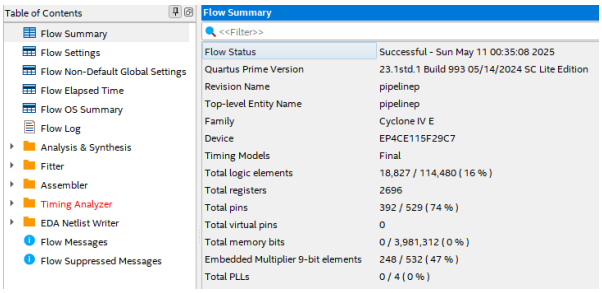


Figura 19: LUTs NTT de 128 puntos pipeline 32 BF

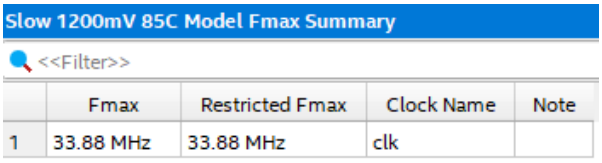


Figura 20: Frecuencia NTT de 128 puntos pipeline 32 BF

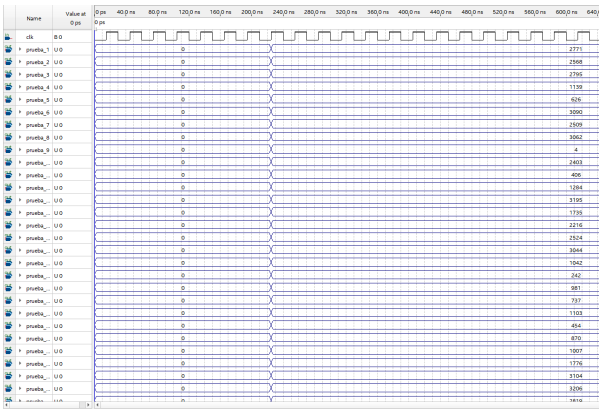


Figura 21: Resultados NTT de 128 puntos pipeline 32 BF

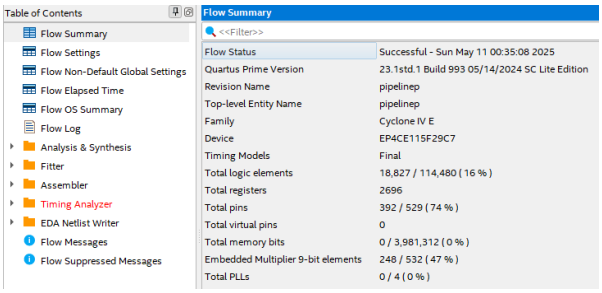


Figura 22: LUTs NTT de 128 puntos pipeline 64 BF

	Fmax	Restricted Fmax	Clock Name	Note
1	34.4 MHz	34.4 MHz	clk	

Figura 23: Frecuencia NTT de 128 puntos pipeline 64 BF

Figura 24: Resultados NTT de 128 puntos pipeline 64 BF

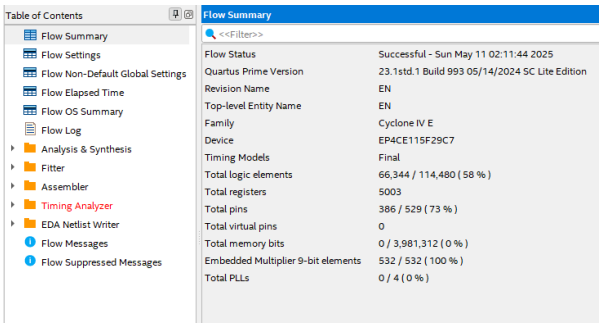


Figura 25: LUTs NTT de 128 puntos reutilización componente 32 BF

	Fmax	Restricted Fmax	Clock Name	Note
1	31.06 MHz	31.06 MHz	clk	

Figura 26: Frecuencia NTT de 128 puntos reutilización componente 32 BF

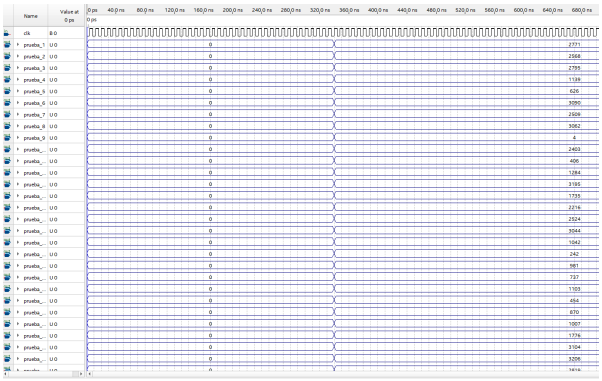


Figura 27: Resultados NTT de 128 puntos reutilización componente 32 BF

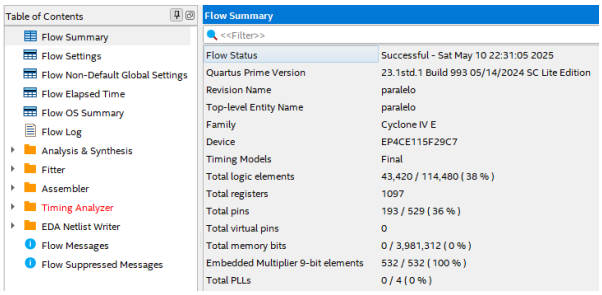


Figura 28: LUTs NTT de 256 puntos arquitectura paralela

	Fmax	Restricted Fmax	Clock Name	Note
1	4.92 MHz	4.92 MHz	clk	

Figura 29: Frecuencia NTT de 256 puntos arquitectura paralela

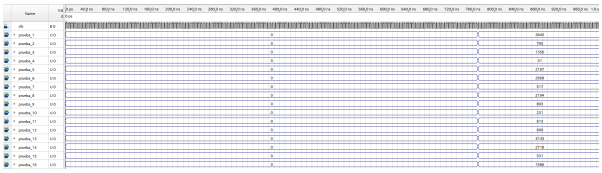


Figura 30: Resultados NTT de 256 puntos arquitectura paralela

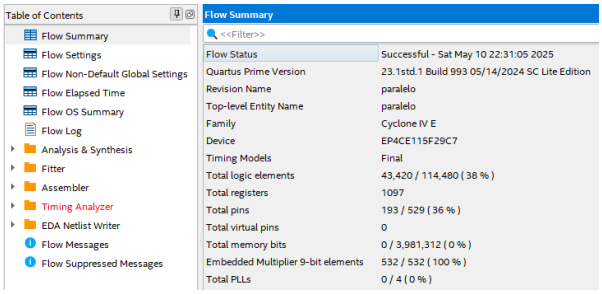


Figura 31: LUTs NTT de 256 puntos arquitectura paralela

	Fmax	Restricted Fmax	Clock Name	Note
1	4.92 MHz	4.92 MHz	clk	

Figura 32: Frecuencia NTT de 256 puntos arquitectura paralela

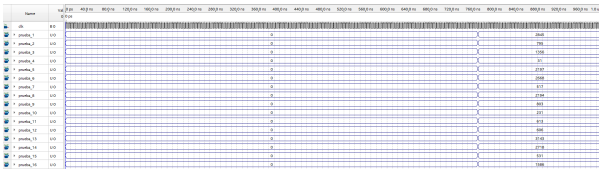


Figura 33: Resultados NTT de 256 puntos arquitectura paralela

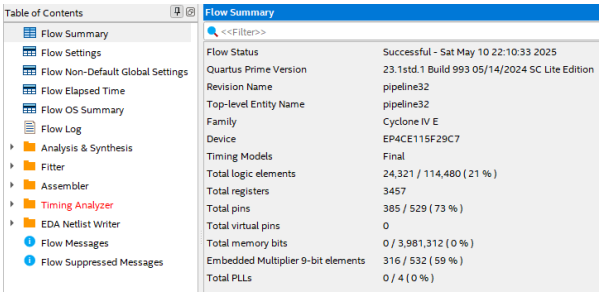


Figura 34: LUTs NTT de 256 puntos arquitectura pipeline de 128 BF

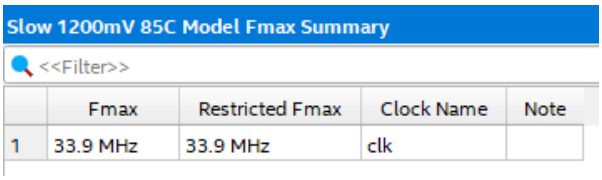


Figura 35: Frecuencia NTT de 256 puntos arquitectura pipeline de 128 BF

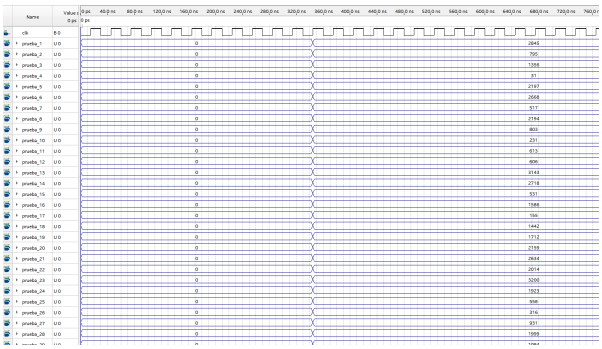


Figura 36: Resultados NTT de 256 puntos arquitectura pipeline de 128 BF

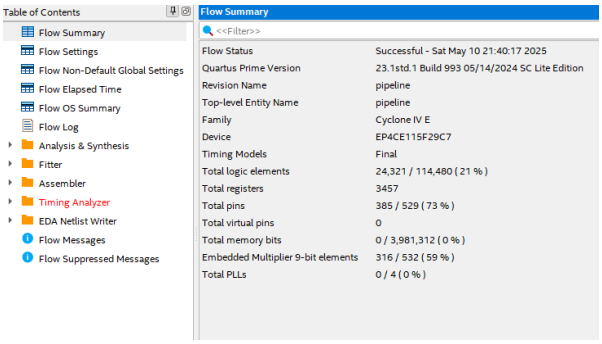


Figura 37: LUTs NTT de 256 puntos arquitectura pipeline de 64 BF

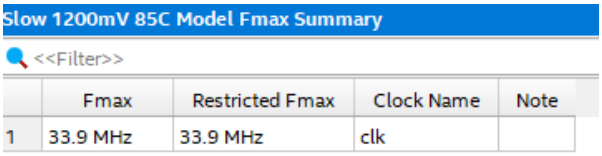


Figura 38: Frecuencia NTT de 256 puntos arquitectura pipeline de 64 BF

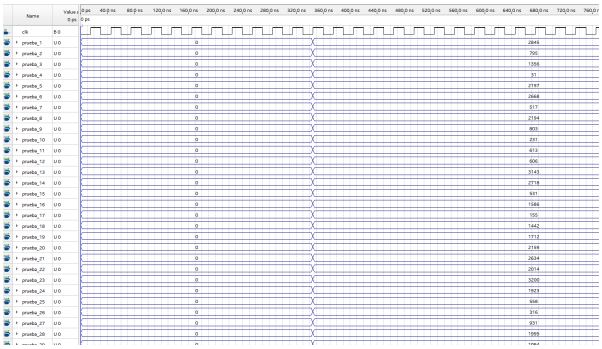


Figura 39: Resultados NTT de 256 puntos arquitectura pipeline de 64 BF

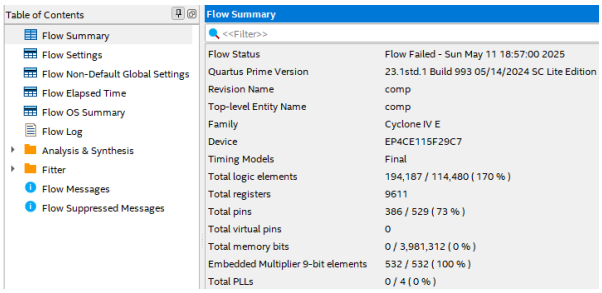


Figura 40: LUTs NTT de 256 puntos arquitectura reutilizando component 64 BF

Figura 41: Ejecución en hardware usando Signal Tap