UNIVERSIDAD COMPLUTENSE DE MADRID

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS FÍSICAS

Máster en Nuevas Tecnologías Electrónicas y Fotónicas

TRABAJO DE FIN DE MÁSTER

Implementación Hardware de Funciones Hash para Criptografía Post-cuántica

HARDWARE IMPLEMENTATION OF HASH FUNCTIONS FOR POST-QUANTUM CRYPTOGRAPHY

Jon Gómez Manchola

DIRECTOR:
JOSÉ LUIS IMAÑA PASCUAL

Departamento de Arquitectura de Computadores y Automática

Curso académico 2024-2025
Convocatoria Junio
Calificación: 9,8

Resumen:

La llegada de ordenadores cuánticos de gran escala comprometerá los esquemas clásicos de clave pública como RSA, DH y ECC. Las propuestas post-cuánticas, por ejemplo, Kyber y Dilithium, dependen de la familia SHA3/SHAKE para generar aleatoriedad interna, convirtiendo al núcleo hash en el cuello de botella de sistemas empotrados y de alto rendimiento. Este trabajo presenta la implementación hardware de las cuatro variantes exigidas por dichos estándares (SHA3-256/512 y SHAKE128/256) orientada a FPGAs Xilinx Artix-7 de gama media. Se han desarrollado nueve arquitecturas de la permutación KECCAK-f[1600] de 24 rondas, desde un núcleo totalmente combinacional hasta un pipeline de cinco etapas que separa los step-mappings $\theta, \rho, \pi, \chi, \iota$ en 1-1-1-1-1. Bajo idénticas restricciones de síntesis se comparan área, retardo crítico y eficiencia (Gb s^{-1}·$Slice^{-1}$). La segmentación 3–2 se perfila como la opción con mejor equilibrio entre prestaciones y coste, al mejorar sensiblemente la frecuencia de operación con un impacto mínimo en el área. El análisis temporal corrobora que las fases θ y χ fijan el camino crítico. A partir de esta observación, se formulan recomendaciones para balancear latencia, área y consumo en futuras migraciones a ASIC.

Abstract:

The inevitable appearance of large-scale quantum computers will render classical public-key schemes such as RSA, DH and ECC insecure. Post-quantum proposals (e.g. Kyber and Dilithium) rely heavily on the SHA3/SHAKE family to generate internal randomness, making the hash core a performance bottleneck in resource-constrained and high-throughput platforms. This thesis presents a systematic hardware implementation of the four variants required by those standards (SHA3-256/512 and SHAKE128/256) targeting mid-range Xilinx Artix-7 FPGAs. Nine architectures of the underlying 24-round KECCAK-f[1600] permutation were developed, ranging from a fully combinational core to a five-stage pipeline that splits the $\theta, \rho, \pi, \chi, \iota$ step-mappings as 1-1-1-1-1. Using identical synthesis constraints, we compare area, critical path and efficiency (Gb s^{-1}·$Slice^{-1}$). The 3–2 segmentation emerges as the option with the best trade-off between performance and cost, significantly boosting the operating frequency while incurring only a minimal area overhead. Timing analysis confirms that the θ and χ steps dictate the critical path. Building on this insight, we outline guidelines for balancing latency, area, and power in future ASIC migrations.

Palabras clave:

Criptografía post-cuántica, SHA3, SHAKE, KECCAK, FPGA, acelerador hardware, segmentación, rendimiento, Artix-7, VHDL,

Keywords:

Post-quantum cryptography, SHA3, SHAKE, KECCAK, FPGA, hardware accelerator, segmentation, throughput, Artix-7, VHDL.

Índice

1. Introducción y Objetivos **1**

1.1. Objetivos . . . 2

2. Fundamento Teórico **4**

2.1. Criptografía post-cuántica . . . 4

2.2. Funciones hash criptográficas . . . 5

2.3. SHA3: Secure Hash Algorithm 3 . . . 8

2.3.1. Construcción sponge . . . 9

2.3.2. KECCAK . . . 10

2.3.3. Funciones SHA3 y SHAKE . . . 16

2.3.4. Ventajas prácticas de SHA3 en hardware . . . 17

3. Diseño e implementación **19**

3.1. Requisitos de diseño . . . 19

3.2. Descripción general de la arquitectura . . . 20

3.3. Estructura y desarrollo del código VHDL . . . 21

3.4. Validación funcional mediante testbenches . . . 22

3.5. Variantes arquitectónicas implementadas . . . 22

3.6. Implementación en FPGA . . . 24

4. Resultados y Análisis **27**

4.1. Métricas de Evaluación . . . 27

4.2. Resultados Obtenidos . . . 29

4.2.1. SHA3-256 . . . 29

4.2.2. SHA3-512 . . . 30

4.2.3. SHAKE128 . . . 30

4.2.4. SHAKE256 . . . 31

4.3. Análisis de los Resultados . . . 32

4.3.1. Análisis de SHA3 . . . 32

4.3.2. Análisis de SHAKE . . . 34

4.3.3. Elección de arquitectura según los requisitos del sistema . . . 37

5. Conclusiones **39**

5.1. Limitaciones del trabajo . . . 40

5.2. Líneas de investigación futura . . . 41

1. Introducción y Objetivos

El avance de la computación cuántica amenaza los cimientos de la criptografía clásica. Cuando existan máquinas capaces de ejecutar el algoritmo de Shor a gran escala, esquemas como RSA, Diffie–Hellman o ECC dejarán de ser seguros. Para adelantarse a ese escenario, el NIST inició en 2016 la estandarización de algoritmos post-cuánticos [9], entre los que Kyber [14] y Dilithium [12] destacan por su eficiencia. Ambos dependen de la familia de funciones SHA3/SHAKE para generar la aleatoriedad interna que garantice su seguridad.

En sistemas empotrados y en infraestructuras de alta velocidad, el cálculo de SHA3/SHAKE se convierte en el cuello de botella principal. Las implementaciones en software no alcanzan el rendimiento ni la eficiencia requeridas. De ahí el interés en diseñar aceleradores hardware que maximicen el throughput y minimicen el consumo. El trabajo se sitúa en ese contexto, centrándose en cómo la organización interna del núcleo KECCAK-f[1600] afecta al área y a la velocidad cuando se implementa en FPGA, y sentando las bases para una futura migración a ASIC (Application-Specific Integrated Circuits), es decir, a un circuito integrado diseñado para realizar una tarea específica de manera eficiente.

Las FPGAs (Field-Programmable Gate Arrays) son dispositivos programables que permiten implementar circuitos digitales personalizados. Están formadas por una red de bloques lógicos configurables, interconexiones programables y elementos auxiliares como memorias o multiplicadores. Cada uno de estos bloques lógicos incluye principalmente dos recursos fundamentales: las Look-Up Tables (LUTs) y los flip-flops.

Las LUTs son pequeñas memorias que permiten implementar cualquier función lógica combinacional. Actúan como tablas de verdad que almacenan el resultado de una función lógica para cada posible combinación de entradas. Por ejemplo, una LUT de 4 entradas puede representar cualquier función booleana de cuatro variables. Los flip-flops, por su parte, permiten almacenar un bit de información y son esenciales para describir circuitos secuenciales. Su salida se actualiza en sincronía con una señal de reloj, lo que permite registrar estados intermedios y controlar el flujo de datos. Ambos elementos se agrupan en estructuras denominadas Slices, que constituyen la unidad básica de lógica en una FPGA. El número de LUTs, flip-flops y Slices utilizados es una métrica habitual para evaluar el área ocupada por un diseño y su complejidad.

La familia ARTIX-7 de Xilinx, empleada en este trabajo, pertenece a la gama media de las FPGAs y destaca por su buena relación entre coste, consumo y prestaciones, lo que la hace especialmente adecuada para aplicaciones empotradas. El diseño y la síntesis de los módulos se ha realizado con el entorno Vivado 2023.2, que automatiza tareas como la colocación de bloques, el enrutado de señales y el análisis temporal.

Para describir los módulos hardware se ha empleado VHDL (VHSIC Hardware Description Language), un lenguaje de descripción hardware estandarizado que permite modelar el comportamiento y la estructura de circuitos digitales.

En la literatura se describen múltiples aceleradores de SHA3/SHAKE, pero la mayoría [7, 11, 4] se centran en un único nivel de segmentación o en dispositivos de gama alta cuyos recursos y redes de interconexión distan mucho de los disponibles en FPGAs de coste medio como la Artix-7 empleada masivamente en sistemas empotrados. Pocos trabajos comparan de forma sistemática cómo varía el rendimiento cuando se pasa de una arquitectura totalmente combinacional a una segmentada en varias etapas.

1.1. Objetivos

El propósito general de este trabajo fin de máster es caracterizar de forma sistemática el impacto de la segmentación por etapas del núcleo KECCAK-f[1600] sobre el rendimiento y el consumo de área cuando se implementa SHA3/SHAKE en hardware. En primer lugar, se desarrollan en VHDL las cuatro variantes utilizadas en KYBER, especificadas en FIPS 202 [13] (SHA3-256, SHA3-512, SHAKE128 y SHAKE256). En segundo lugar, se implementan nueve arquitecturas, desde la versión totalmente combinacional hasta segmentaciones de cinco etapas (3-2, 2-3, 3-1-1, 2-2-1, 2-1-1-1, etc.), con el fin de medir, bajo idénticas condiciones de compilación, cómo evolucionan el retardo crítico, el área (LUTs, FFs, Slices) y la eficiencia (Gb/s·Slice^{-1}). El estudio no aborda la fase física completa de un flujo ASIC (aunque se discute como trabajo futuro). Se centra en el núcleo SHA3/SHAKE aislado y en su impacto sobre área y frecuencia dentro de la familia Artix-7.

El trabajo está dividido en cinco secciones. La Sección 2 comienza situando la criptografía post-cuántica, resume la amenaza que supone el algoritmo de Shor, el proceso de estandarización del NIST y las cuatro grandes familias de esquemas resistentes. A continuación se introducen las funciones hash criptográficas, se repasan sus propiedades de seguridad y las construcciones Merkle–Damgård y sponge. Por último se describe en detalle el estándar SHA3, se presenta la construcción sponge, la permutación KECCAK-f[1600] con sus cinco step-mappings ($\theta, \rho, \pi, \chi, \iota$), el padding `pad101`, y las seis variantes de SHA3-n y SHAKE.

En la Sección 3 se expone la arquitectura VHDL propuesta para las cuatro variantes SHA3/SHAKE y se describe cómo se generan de forma parametrizable distintas segmentaciones. También se definen las métricas (área, frecuencia y eficiencia) que se utilizarán más adelante para la comparación de resultados.

En la Sección 4 se presentan las cifras de área y rendimiento obtenidas para cada variante y cada segmentación, se compara su eficiencia sobre la FPGA de referencia y se analiza cómo la segmentación interna afecta al compromiso entre frecuencia máxima y ocupación de recursos. También se analizan las diferencias y semejanzas entre las variantes SHA3-256 y SHA3-512, así como entre SHAKE128 y SHAKE256.

Por último, en la Sección 5 se sintetizan las aportaciones clave, se señalan las limitaciones

detectadas y se enumeran las líneas de investigación que se abren a partir del trabajo.

2. Fundamento Teórico

En esta sección se presentan los fundamentos teóricos necesarios para contextualizar el trabajo realizado. Se comienza introduciendo la criptografía post-cuántica, analizando las amenazas que plantea la computación cuántica sobre los algoritmos criptográficos clásicos y describiendo las distintas familias de algoritmos resistentes a ataques cuánticos.

A continuación, se abordan las funciones hash criptográficas, destacando sus propiedades de seguridad, estructuras internas y relevancia en esquemas post-cuánticos. Se comparan las familias SHA2 y SHA3, poniendo en evidencia las ventajas estructurales de esta última, que la convierten en una candidata especialmente adecuada para su implementación en hardware.

Posteriormente, se analiza en profundidad el estándar SHA3. Se comienza por la descripción de la construcción sponge, seguida del funcionamiento interno del algoritmo KECCAK, incluyendo la permutación KECCAK-p y las cinco funciones de paso que la componen. También se detalla el esquema de padding `pad10*1`, y se define formalmente la función KECCAK[c] como base de todas las variantes de SHA3. Finalmente, se presentan las funciones hash SHA3-n y SHAKE, con sus respectivos parámetros y características.

Esta base teórica proporciona el conocimiento necesario para comprender las decisiones de diseño adoptadas en el capítulo siguiente, donde se describe la implementación hardware de SHA3-256 y sus variantes arquitectónicas.

2.1. Criptografía post-cuántica

El desarrollo de ordenadores cuánticos plantea una amenaza crítica para los algoritmos criptográficos actuales de clave pública. Muchos de estos algoritmos, como RSA, Diffie-Hellman o la criptografía de curva elíptica (ECC), se basan en problemas matemáticos que son considerados computacionalmente difíciles de resolver con ordenadores clásicos, pero sí pueden resolverse eficientemente con ordenadores cuánticos.

En particular, en 1994, Peter Shor introdujo un algoritmo cuántico que permite factorizar un número entero n de forma eficiente. Este algoritmo realiza aproximadamente $(\log n)^{2+o(1)}$ operaciones sobre un ordenador cuántico de tamaño $(\log n)^{1+o(1)}$ [1]. Como consecuencia, toda la criptografía basada en la dificultad de la factorización de enteros o del logaritmo discreto (RSA, ECC, Diffie-Hellman...) quedaría comprometida ante ordenadores cuánticos suficientemente potentes.

Ante esta situación, surge la necesidad de desarrollar algoritmos que sean seguros frente a adversarios con capacidad cuántica. Este nuevo paradigma se conoce como criptografía post-cuántica. A diferencia de la criptografía cuántica (que requiere tecnología especializada), la criptografía post-cuántica está diseñada para ejecutarse en ordenadores clásicos y ser segura frente a ataques cuánticos.

El NIST (National Institute of Standards and Technology) inició en 2016 un proceso de

estandarización de algoritmos post-cuánticos [9], con el objetivo de definir nuevos estándares criptográficos resistentes a este tipo de amenazas. Las propuestas finalistas se agrupan en distintas familias, cada una basada en un tipo de problema matemático diferente [1]:

- **Code-based cryptography:** basada en problemas relacionados con códigos de corrección de errores, como el problema de descodificación de errores aleatorios. El esquema más representativo es McEliece.
- **Lattice-based cryptography:** basada en problemas de retículos, como el Shortest Vector Problem (SVP) o Learning With Errors (LWE). Ofrece buena eficiencia y versatilidad. Es la base de esquemas como Kyber y Dilithium, ya estandarizados por NIST.
- **Multivariate quadratic equations:** utiliza sistemas de ecuaciones cuadráticas multivariables sobre cuerpos finitos. Es segura, pero suele tener firmas grandes.
- **Hash-based cryptography:** usa únicamente funciones hash seguras, como SHA2 o SHA3, para construir firmas digitales seguras, incluso frente a ataques cuánticos. Un ejemplo es SPHINCS+ [5], candidato aprobado por el NIST.

Por otro lado, a pesar de no ser una de las propuestas finalistas, cabe destacar también:

- **Symmetric-key-based cryptography:** los algoritmos simétricos como AES o SHA2 se ven menos afectados por la computación cuántica. El algoritmo de Grover puede acelerar la búsqueda de claves, pero no romper el sistema completamente. Por eso, duplicar el tamaño de clave suele ser suficiente.

Cabe destacar que los algoritmos post-cuánticos no dependen de suposiciones algebraicas vulnerables, y muchos de ellos están bien fundamentados en teoría de la información o combinatoria. No existe, a día de hoy, un algoritmo cuántico que rompa estos esquemas de forma eficiente, lo que los convierte en sólidos candidatos para asegurar la información en la era cuántica.

2.2. Funciones hash criptográficas

Las funciones hash criptográficas son funciones deterministas que transforman una entrada de longitud arbitraria en una salida de tamaño fijo. Su objetivo principal es ofrecer una huella digital única de los datos, y se utilizan ampliamente en la integridad de datos, autenticación, generación de claves y firmas digitales.

Para que una función hash sea considerada segura en el ámbito criptográfico, debe cumplir las siguientes propiedades fundamentales [8]:

- **Resistencia a la preimagen:** dada una salida h, debe ser computacionalmente inviable encontrar una entrada m tal que $H(m) = h$, donde H es la función hash.

- **Resistencia a la segunda preimagen:** dado un mensaje m_1, debe ser difícil encontrar otro mensaje $m_2 \neq m_1$ tal que $H(m_1) = H(m_2)$.
- **Resistencia a colisiones:** debe ser difícil encontrar dos entradas distintas $m_1 \neq m_2$ tales que $H(m_1) = H(m_2)$.
- **Efecto avalancha:** un pequeño cambio en la entrada debe provocar un cambio significativo en la salida.

Las funciones hash criptográficas suelen construirse a partir de una función de compresión básica, que se aplica de forma iterativa sobre bloques del mensaje de entrada. Existen varias formas de organizar esta estructura interna. Las dos más utilizadas son:

1. **Construcción de Merkle–Damgård:** es utilizada en funciones clásicas como MD5, SHA1 y SHA2. El mensaje se divide en bloques de tamaño fijo, y cada uno se procesa junto con un valor intermedio (estado) usando una función de compresión. El resultado final tras procesar todos los bloques es el digest. Este enfoque es sencillo y eficiente, pero presenta algunas debilidades frente a ciertos ataques (como los de extensión de longitud).
2. **Construcción esponja (sponge):** es utilizada en funciones modernas como SHA3 (KECCAK). El mensaje se absorbe en un estado interno dividido en dos partes: bitrate (parte visible) y capacida (parte oculta). Una vez absorbido el mensaje, se extrae la salida (digest) desde el mismo estado. Esta estructura es más flexible y segura frente a varias clases de ataques estructurales.

En las Figuras 1 y 2 se muestra un esquema simplificado de ambas construcciones.

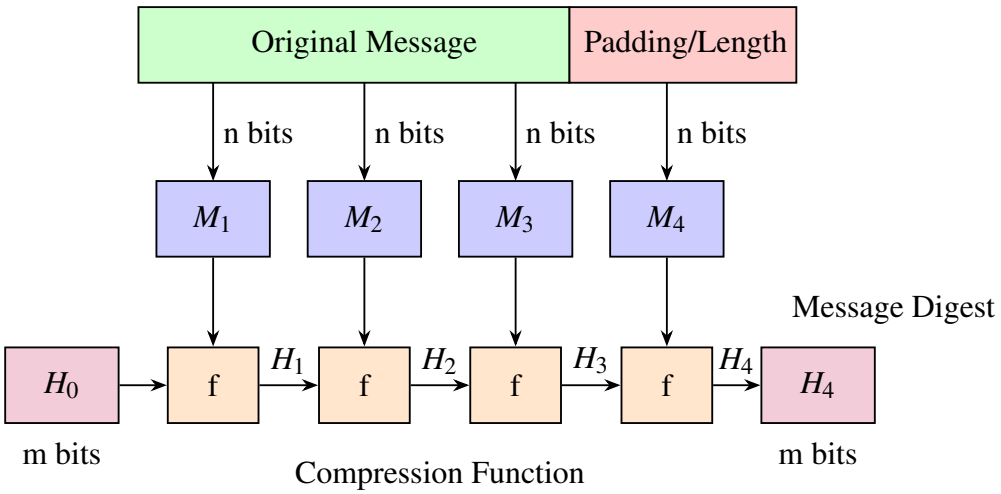

Figura 1: Construcción Merkle–Damgård: el mensaje original se divide en bloques de n bits, y cada uno se procesa con una función de compresión junto con el hash anterior, generando una cadena de hash intermedios hasta obtener el resumen final.

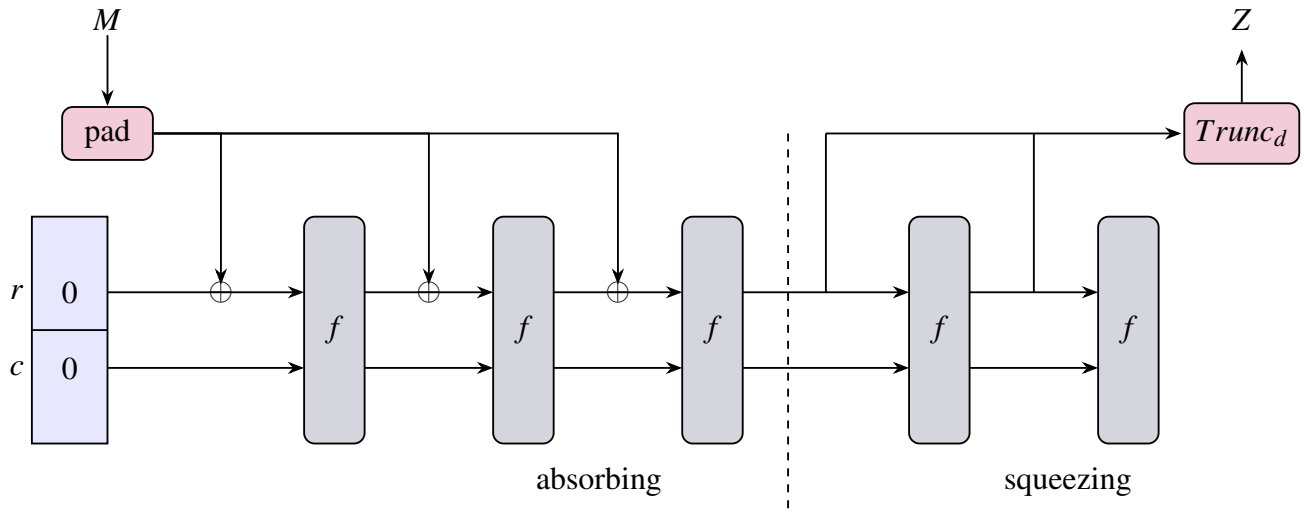

Figura 2: Diagrama de la construcción sponge, destacando las fases de absorción y extracción.

En el contexto de la criptografía post-cuántica, las funciones hash presentan una ventaja significativa respecto a los algoritmos de clave pública tradicionales, se ven menos afectadas por la computación cuántica. Aunque el algoritmo de Shor puede romper esquemas como RSA o ECC al resolver la factorización o el logaritmo discreto en tiempo polinomial, las funciones hash resisten este tipo de ataque directo.

El principal riesgo cuántico sobre funciones hash proviene del algoritmo de Grover, que permite realizar una búsqueda no estructurada en un espacio de tamaño N con una complejidad de $\mathcal{O}(\sqrt{N})$ [3]. Aplicado a la búsqueda de preimágenes, esto implica que una función hash de n bits podría ser atacada con un coste de $\mathcal{O}(2^{n/2})$, en lugar de $\mathcal{O}(2^n)$ como en el caso clásico.

Como consecuencia, se recomienda duplicar el tamaño del digest para mantener un nivel equivalente de seguridad frente a adversarios cuánticos. Por ejemplo, SHA3-256, que produce un digest de 256 bits, ofrece aproximadamente 128 bits de seguridad frente a ataques cuánticos, lo que la hace adecuada para aplicaciones en entornos post-cuánticos [1].

Las funciones hash se han convertido en un pilar fundamental de la criptografía post-cuántica, utilizándose como base en esquemas de firma, generación de claves y construcción de identificadores seguros. Su simplicidad, eficiencia y resistencia frente a ataques cuánticos las hacen especialmente atractivas para su implementación en hardware.

Dentro de las funciones hash estandarizadas, se destacan dos familias principales: SHA2 y SHA3. Ambas ofrecen niveles adecuados de seguridad, pero presentan diferencias clave tanto en su estructura como en sus propiedades criptográficas. La familia SHA2 se basa en la construcción clásica de Merkle–Damgård, mientras que SHA3 emplea la construcción sponge, lo que le confiere propiedades adicionales como resistencia a ataques de extensión de longitud y mayor flexibilidad en la salida.

En la Tabla 1 se resumen las diferencias más relevantes:

Tabla 1: Comparativa entre SHA2 y SHA3.

Característica	**SHA2**	**SHA3 (KECCAK)**
Construcción interna	Merkle–Damgård	Sponge
Ataques conocidos	Extensión de longitud	No aplica
Paralelización	Limitada	Compatible con estructura en árbol (Sakura)
Longitud del digest	Fija	Variable (SHAKE)
Padding	Basado en longitud	`pad10*1`

Además de sus ventajas teóricas, SHA3 resulta especialmente eficiente para su implementación en dispositivos hardware como FPGAs. Sus operaciones internas (rotaciones, XORs y funciones booleanas) son simples de paralelizar y consumirán menos recursos lógicos comparados con estructuras más complejas.

Por estas razones, y por su adopción en esquemas post-cuánticos como SPHINCS+, se ha optado en este trabajo por implementar la función hash SHA3, que ofrece un buen compromiso entre seguridad, eficiencia y facilidad de integración en arquitecturas digitales.

Además de su importancia en esquemas de firma hash-based como SPHINCS+, es relevante señalar que las funciones SHAKE (SHA3 Extendable Output Function), incluidas en la familia de algoritmos SHA3, han adquirido un papel central en la criptografía post-cuántica por su capacidad de generar salidas de longitud variable, adaptándose a múltiples necesidades como pseudoaleatoriedad y derivación de claves. De hecho, algoritmos post-cuánticos estandarizados recientemente por el NIST, como Kyber [14] y Dilithium [12], utilizan de forma intensiva tanto SHA3 como SHAKE para la generación de semillas, valores pseudoaleatorios y digests, así como en diversos procesos internos de encapsulación y firma. Por lo tanto, la eficiencia y seguridad global de estos algoritmos depende en gran medida del rendimiento y la robustez de las implementaciones hardware de SHA3 y SHAKE, lo que refuerza la motivación de este trabajo.

A continuación, se estudia con más detalle la estructura y funcionamiento del estándar SHA3.

2.3. SHA3: Secure Hash Algorithm 3

La familia SHA3 fue estandarizada por el NIST en 2015 mediante la publicación FIPS 202, tras un proceso público de evaluación iniciado en 2007. El objetivo de este proceso fue definir una nueva familia de funciones hash que ofreciera un diseño independiente de SHA2, con propiedades de seguridad adicionales y un enfoque más robusto ante ataques estructurales. El algoritmo seleccionado fue KECCAK, desarrollado por Guido Bertoni, Joan Daemen, Michaël Peeters y Gilles Van Assche [13].

SHA3 no sustituye a SHA2, sino que lo complementa como alternativa segura, introduciendo un nuevo paradigma de construcción interna, la estructura sponge. Esta construcción permite absorber entradas de longitud arbitraria y generar salidas de longitud también variable, mediante

un proceso basado en una permutación iterativa y un esquema de padding flexible.

Además de aportar diversidad criptográfica en términos de diseño, SHA3 destaca por su versatilidad, ya que permite definir tanto funciones hash convencionales como funciones de salida extensible (XOFs), adaptándose a múltiples aplicaciones. Esta capacidad de generar salidas de longitud arbitraria resulta especialmente útil en protocolos criptográficos modernos, como generadores de claves o firmas digitales con parámetros ajustables. En las siguientes subsecciones se detalla su funcionamiento interno y los componentes fundamentales que forman la base del algoritmo KECCAK.

2.3.1. Construcción sponge

La construcción sponge es el principio criptográfico fundamental sobre el que se basa el algoritmo KECCAK y, por tanto, toda la familia SHA3. Este esquema fue definido formalmente por Bertoni et al. en 2011 [2], y posteriormente estandarizado por el NIST en la publicación FIPS 202 [13].

El objetivo de esta construcción es permitir el diseño de funciones sobre datos binarios de entrada arbitraria y salida también arbitraria (en el caso de funciones XOF), mediante un único esquema estructural altamente flexible. Para ello, se emplean tres componentes clave:

- Una función sobre cadenas binarias de longitud fija, denotada como f.
- Un parámetro llamado rate r, que controla cuántos bits se absorben o extraen por iteración.
- Una regla de relleno o padding, denotada por pad.

La función resultante se denota por $\texttt{SPONGE}[f, \texttt{pad}, r]$ y opera sobre dos entradas, una cadena binaria N y una longitud de salida d.

El estado interno tiene una longitud total $b = r + c$, donde c es la capacidad. El valor de c determina el nivel de seguridad contra ataques como colisiones y preimagen.

El procedimiento se divide en dos fases:

1. **Absorción (absorbing)**: se aplica el padding al mensaje N para que su longitud sea múltiplo de r. Luego se divide en bloques de r bits: $P_0, P_1, \ldots, P_{n-1}$. Se inicializa el estado interno S como 0^b y, para cada bloque, se actualiza el estado como $S \leftarrow f(S \oplus (P_i || 0^c))$.
2. **Extracción (squeezing)**: se extraen los primeros r bits de S, y si la salida requerida d es mayor, se vuelve a aplicar f sobre el estado y se extraen más bits hasta completar la longitud deseada.

Este procedimiento se ilustra en la Figura 2, y su formulación detallada se muestra en el Algoritmo 1 [13].

Algoritmo 1: SPONGE[f, pad, r](N, d)

Input: Cadena binaria N, entero no negativo d

Output: Cadena Z tal que $|Z| = d$

$P \leftarrow N \,\|\, \texttt{pad}(r, |N|)$;

$n \leftarrow |P|/r$;

$c \leftarrow b - r$;

Dividir P en bloques $P_0, P_1, \ldots, P_{n-1}$ de longitud r ;

$S \leftarrow 0^b$;

for $i \leftarrow 0$ **to** $n-1$ **do**

  $S \leftarrow f(S \oplus (P_i \,\|\, 0^c))$;

$Z \leftarrow \varepsilon$ `// Cadena vacía`

while $|Z| < d$ **do**

  $Z \leftarrow Z \,\|\, \texttt{Trunc}_r(S)$;

  if $|Z| \geq d$ **then**

    return $\mathit{Trunc}_d(Z)$;

  $S \leftarrow f(S)$;

Recalcar que el parámetro de entrada d solo afecta a la cantidad de bits de salida del algoritmo y no a la obtención de estos, como se puede observar en el Algoritmo 1.

2.3.2. KECCAK

KECCAK es una familia de funciones hash que se basa en la construcción sponge y en una permutación interna iterativa denominada KECCAK-p. Para comprender adecuadamente el funcionamiento de KECCAK, y por tanto de SHA3, es necesario describir primero esta permutación, que constituye el núcleo criptográfico del algoritmo.

A continuación, se presenta la permutación KECCAK-p, la regla de relleno utilizada y la forma en que se define KECCAK como función completa dentro del esquema sponge.

Permutación KECCAK-p La permutación KECCAK-p[b, nr] es el núcleo transformador del algoritmo KECCAK. Actúa sobre un estado de b bits y aplica una secuencia de nr rondas de transformación, cada una compuesta por cinco funciones de paso (step mappings). Estas cinco funciones son θ, ρ, π, χ y ι y su funcionamiento se explica a continuación. Esta permutación no depende de ninguna clave ni parámetro externo, y puede considerarse como una transformación determinista sobre cadenas de longitud fija.

El estado interno se organiza como una matriz tridimensional A de dimensiones $5 \times 5 \times w$, donde $w = b/25$. Cada posición (x, y) de la matriz contiene una "lane"de w bits, y el conjunto completo representa el estado de trabajo del algoritmo. Esto es, si los bits de entrada se definen

como:

$$S = S[0] \,\|\, S[1] \,\|\, \ldots \,\|\, S[b-2] \,\|\, S[b-1],$$

se define la matriz A como:

$$\mathbf{A}[x,y,z] = S[w(5y+x)+z], \quad \text{para } 0 \leq x,y < 5,\ 0 \leq z < w.$$

Una ronda de la permutación se expresa como:

$$\texttt{Rnd}(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r),$$

donde A es el estado de entrada e i_r es el índice de ronda. La permutación completa se obtiene aplicando nr de estas rondas de forma secuencial.

En el contexto del estándar SHA3, se utiliza una instancia concreta de esta familia, denominada KECCAK-f[b], la cual se define como la permutación KECCAK-p[b, 12 + 2l], donde $l = \log_2(w)$ y $w = b/25$. Para $b = 1600$ (valor usado en SHA3), se tiene $l = 6$, por lo tanto:

$$\texttt{KECCAK-f}[1600] = \texttt{KECCAK-p}[1600, 24].$$

Es decir, se aplica una permutación sobre un estado de 1600 bits mediante 24 rondas consecutivas. Esta permutación se emplea como función interna f en la construcción sponge, proporcionando una excelente difusión, no linealidad y resistencia frente a ataques criptográficos estructurales. Su diseño, basado únicamente en operaciones lógicas simples, la hace especialmente adecuada para su implementación en hardware.

Función θ La función θ es la primera operación que se aplica en cada ronda de la permutación KECCAK-p. Su objetivo es introducir difusión entre los distintos bits del estado a través de la paridad de las columnas.

Para ello, primero se calcula, para cada par de índices (x,z), la paridad de las cinco posiciones a lo largo del eje y, generando así un array auxiliar $C[x,z]$. Luego, se calcula un valor de corrección $D[x,z]$ para cada posición (x,z), utilizando posiciones desplazadas de C. Finalmente, se actualiza cada bit del estado añadiendo (mediante XOR) la corrección $D[x,z]$.

El algoritmo se describe formalmente a continuación:

Algoritmo 2: $\theta(A)$

Input: Array de estado A

Output: Nuevo array de estado A'

for *todos los pares* (x,z) *con* $0 \le x < 5,\ 0 \le z < w$ **do**
  $C[x,z] \leftarrow A[x,0,z] \oplus A[x,1,z] \oplus A[x,2,z] \oplus A[x,3,z] \oplus A[x,4,z]$

for *todos los pares* (x,z) *con* $0 \le x < 5,\ 0 \le z < w$ **do**
  $D[x,z] \leftarrow C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z-1) \bmod w]$

for *todos los tríos* (x,y,z) *con* $0 \le x,y < 5,\ 0 \le z < w$ **do**
  $A'[x,y,z] \leftarrow A[x,y,z] \oplus D[x,z]$

Función ρ La función ρ se encarga de rotar los bits dentro de cada lane del estado tridimensional $A[x,y,z]$. Estas rotaciones son cruciales para la difusión de la información a lo largo de las rondas. Cada rotación depende de la posición (x,y) de la lane y sigue una secuencia pseudoaleatoria determinada por una función cuadrática sobre un número de rondas.

El algoritmo comienza con la lane $(0,0)$, que no se rota, y luego recorre el resto de las posiciones siguiendo una trayectoria cíclica definida por transformaciones sobre (x,y).

Algoritmo 3: $\rho(A)$

Input: Array de estado A

Output: Nuevo array de estado A'

for *todos los z con* $0 \le z < w$ **do**
  $A'[0,0,z] \leftarrow A[0,0,z]$

$(x,y) \leftarrow (1,0)$;

for $t \leftarrow 0$ **to** 23 **do**
  for *todos los z con* $0 \le z < w$ **do**
    $A'[x,y,z] \leftarrow A[x,y,(z-((t+1)(t+2)/2)) \bmod w]$
  $(x,y) \leftarrow (y,(2x+3y) \bmod 5)$;

return A'

El efecto de la función ρ es rotar los bits de cada lane por un número de posiciones que dependen únicamente de sus coordenadas (x,y). A cada lane se le aplica un desplazamiento circular (offset) sobre su eje z, es decir, dentro de su propio conjunto de w bits. La cantidad de desplazamiento se calcula mediante la fórmula cuadrática del Algoritmo 3, pero equivale a usar los valores definidos en la Tabla 2.

Tabla 2: Offsets aplicados por la función ρ para cada lane (x, y) [13].

y\x	0	1	2	3	4
0	0	1	190	28	91
1	36	300	6	55	276
2	3	10	171	153	231
3	105	45	15	21	136
4	210	66	253	120	78

Por tanto, la implementación práctica de ρ puede optimizarse aplicando directamente estos desplazamientos precomputados sobre cada lane, en lugar de calcular dinámicamente la fórmula para cada ronda.

Función π La función π se encarga de reorganizar la posición de las lanes dentro del estado tridimensional $A[x, y, z]$, moviéndolas a nuevas posiciones según una permutación fija. Esta operación no modifica los bits dentro de cada lane, sino que simplemente reubica su localización en el plano (x, y).

Esta permutación contribuye a la difusión global del algoritmo al romper la alineación estructural de las funciones anteriores, permitiendo que operaciones locales posteriores, como χ, afecten a posiciones distintas del estado en cada ronda. El algoritmo se describe a continuación:

Algoritmo 4: $\pi(A)$

Input: Array de estado A

Output: Nuevo array de estado A'

for *todos los tríos* (x, y, z) *con* $0 \leq x, y < 5$, $0 \leq z < w$ **do**

  $A'[y, z, x] \leftarrow A[(x + 3y) \bmod 5, x, z]$

return A'

Función χ La función χ introduce no linealidad en el estado mediante una combinación booleana de bits a lo largo de las filas del plano (x, y). Cada bit se actualiza tomando el valor original y aplicando un XOR con una función de los valores de otros dos bits en la misma fila.

En particular, para cada posición (x, y, z), el nuevo valor se calcula combinando $A[x, y, z]$ con un AND entre el complemento de $A[(x + 1) \bmod 5, y, z]$ y $A[(x + 2) \bmod 5, y, z]$. Esta operación asegura que cada bit quede influenciado por sus vecinos inmediatos, reforzando la complejidad no lineal del estado. El algoritmo se muestra a continuación:

Algoritmo 5: $\chi(A)$

Input: Array de estado A

Output: Nuevo array de estado A'

for *todos los tríos* (x,y,z) *con* $0 \leq x,y < 5,\ 0 \leq z < w$ **do**

  $A'[x,y,z] \leftarrow A[x,y,z] \oplus ((\neg A[(x+1) \bmod 5, y, z]) \cdot A[(x+2) \bmod 5, y, z])$

return A'

Función ι La función ι introduce una constante de ronda en el estado, específicamente en la lane $(0,0)$. Esta constante depende del índice de ronda i_r y tiene como objetivo romper simetrías y evitar que el comportamiento del algoritmo sea completamente lineal o estructurado.

A diferencia de otras funciones, ι afecta únicamente a los bits de $A[0,0,z]$, dejando el resto del estado inalterado. La constante de ronda *RC* se calcula mediante una función basada en un registro de desplazamiento lineal (LFSR), cuya implementación se detalla a continuación:

Algoritmo 6: $rc(t)$

Input: Entero t

Output: Bit $rc(t)$

if $t \bmod 255 = 0$ **then**

  return 1

$R \leftarrow 10000000$; `// En binario, 8 bits`

for $i \leftarrow 1$ **to** $t \bmod 255$ **do**

  $R[0] \leftarrow R[0] \oplus R[4]$;

  $R[1] \leftarrow R[1] \oplus R[5]$;

  $R[2] \leftarrow R[2] \oplus R[6]$;

  $R[3] \leftarrow R[3] \oplus R[7]$;

  $R[4] \leftarrow R[4] \oplus R[0]$;

  $R \leftarrow \texttt{Trunc}_8(R)$;

return $R[0]$

La función ι es la siguiente:

Algoritmo 7: $\iota(A, i_r)$

Input: Array de estado A, índice de ronda i_r

Output: Nuevo array de estado A'

for *todos los* (x,y,z) *con* $0 \leq x,y < 5,\ 0 \leq z < w$ **do**
  $A'[x,y,z] \leftarrow A[x,y,z]$

$RC \leftarrow 0^w$; `// Vector de w bits inicializados a cero`

for $j \leftarrow 0$ **to** l **do**
  $RC[2^j - 1] \leftarrow rc(j + 7i_r)$

for $z \leftarrow 0$ **to** $w - 1$ **do**
  $A'[0,0,z] \leftarrow A'[0,0,z] \oplus RC[z]$

return A'

Se observa que las constantes de ronda *RC* dependen exclusivamente del índice de ronda i_r, por lo que pueden precomputarse y almacenarse previamente. Esto permite aplicarlas directamente durante la ejecución del algoritmo, evitando su cálculo dinámico en cada iteración y mejorando así la eficiencia computacional, como se hacía en la función ρ.

Padding `pad10*1` Antes de aplicar la construcción sponge, el mensaje de entrada debe rellenarse para que su longitud total (mensaje más padding) sea múltiplo del parámetro rate r. Para ello, KECCAK emplea un esquema de padding denominado `pad10*1`, caracterizado por su simplicidad y seguridad.

Este esquema añade:

- Un bit inicial 1.
- Un número variable de ceros.
- Un bit final 1.

La cantidad de ceros depende de la longitud original del mensaje y del valor de r, de modo que el mensaje final tras el padding tenga una longitud múltiplo de r, como se muestra en el Algoritmo 8.

Algoritmo 8: `pad10*1`(x,m)

Input: Entero positivo x (el rate), entero no negativo m (longitud del mensaje)

Output: Cadena P tal que $m + \text{len}(P)$ sea múltiplo de x

$j \leftarrow (-m-2) \bmod x$;

return $P \leftarrow 1 \,\|\, 0^j \,\|\, 1$

Función KECCAK[c] Una vez definida la permutación KECCAK-p y el esquema de padding `pad10*1`, se puede construir la función completa KECCAK[c] como una instancia de la construcción sponge.

Dado un parámetro de capacidad c, se calcula el rate como $r = b - c$, siendo b la longitud total del estado interno (en SHA3, $b = 1600$). La función resultante se aplica sobre una entrada N de longitud arbitraria y genera una salida de d bits, aplicando la estructura sponge con los componentes ya descritos.

Formalmente, se define como:

$$\texttt{KECCAK}[c](N,d) = \texttt{SPONGE}[\texttt{KECCAK-p}[b,nr],\texttt{pad10*1},b-c](N,d),$$

donde:

- b es típicamente 1600.
- nr es el número de rondas aplicadas (en SHA3, $nr = 24$).
- c es la capacidad elegida según el nivel de seguridad deseado.

La seguridad de KECCAK[c] frente a colisiones y preimagen está directamente relacionada con el valor de c, proporcionando una resistencia de hasta $c/2$ bits para colisiones y c bits para preimagen, bajo los supuestos de un diseño ideal.

2.3.3. Funciones SHA3 y SHAKE

La familia SHA3 definida por el estándar FIPS 202 incluye dos tipos principales de funciones: las funciones hash clásicas de salida fija (SHA3-n) y las funciones de salida variable conocidas como XOF (eXtendable Output Functions), denominadas SHAKE.

Ambas se basan en la construcción sponge explicada anteriormente, utilizando como función interna KECCAK-p[1600, 24] y el padding `pad10*1`. La principal diferencia entre SHA3 y SHAKE reside en el tipo de salida generada y en el valor del sufijo de padding añadido al mensaje antes de aplicar el relleno.

Las funciones SHA3-224, SHA3-256, SHA3-384 y SHA3-512 generan salidas de longitud fija. Para diferenciarlas de las funciones SHAKE, se añade al mensaje un sufijo binario ‘01‘ antes de aplicar el padding `pad10*1`. El valor de r (rate) y c (capacidad) se elige en función de la longitud de salida y del nivel de seguridad requerido. Formalmente, se definen como:

$$\texttt{SHA3-n}(M) = \texttt{KECCAK}[2n](M \,\|\, 01, n),$$

donde n es la longitud de salida en bits.

Las funciones SHAKE128 y SHAKE256 son XOFs (eXtendable Output Functions), lo que significa que pueden generar salidas de longitud arbitraria. Se emplean, por ejemplo, en criptografía post-cuántica para generar claves, máscaras o semillas. En este caso, se añade un

sufijo binario ‘1111‘ antes del padding, lo que permite distinguirlas de las funciones SHA3-n. Formalmente, se definen como:

$$\texttt{SHAKE128}(M,d) = \texttt{KECCAK}[256](M \,\|\, 1111, d)$$

$$\texttt{SHAKE256}(M,d) = \texttt{KECCAK}[512](M \,\|\, 1111, d),$$

donde d es la longitud de salida deseada.

Se muestra en la Tabla 3 un resumen de los parámetros utilizados por cada función en la familia SHA3:

Tabla 3: Parámetros utilizados por las funciones SHA3 y SHAKE.

Función	**Longitud salida (bits)**	**Rate r (bits)**	**Capacidad c (bits)**	**Sufijo**
SHA3-224	224	1152	448	01
SHA3-256	256	1088	512	01
SHA3-384	384	832	768	01
SHA3-512	512	576	1024	01
SHAKE128	variable	1344	256	1111
SHAKE256	variable	1088	512	1111

2.3.4. Ventajas prácticas de SHA3 en hardware

Además de sus propiedades criptográficas, SHA3 destaca por su adecuación a implementaciones en hardware. Esta eficiencia proviene directamente del diseño del algoritmo KECCAK, el cual se apoya en una permutación fija (KECCAK-p[b, nr]) y evita el uso de funciones de compresión con estructuras de realimentación, como ocurre en la construcción Merkle–Damgård de SHA2.

Una de las principales ventajas para su implementación digital es que todas las operaciones que componen una ronda (rotaciones circulares, operaciones XOR, AND y NOT) son de tipo lógico y de complejidad constante. Estas pueden traducirse fácilmente a estructuras combinacionales simples en lenguajes de descripción hardware como VHDL, lo que reduce tanto el área ocupada como el retardo lógico de cada etapa [13].

Además, dado que cada paso de ronda es determinista, local y aplicable por bloques independientes, SHA3 permite ser implementado mediante arquitecturas segmentadas, lo que resulta especialmente ventajoso en aplicaciones que requieren alta velocidad o procesamiento en paralelo. La ausencia de dependencias de clave o constantes externas (más allá de las constantes de ronda ι) también favorece su integración en sistemas empotrados con recursos limitados.

Estas características han motivado su elección en numerosos esquemas de criptografía postcuántica, como Kyber, que utilizan funciones hash seguras como base criptográfica en entornos

donde el rendimiento hardware es crítico [2, 14].

3. Diseño e implementación

En esta sección se describen en detalle el proceso de diseño hardware y las distintas alternativas arquitectónicas implementadas para las funciones SHA3 y SHAKE, siguiendo las recomendaciones de los estándares FIPS 202 [13]. Se han implementado exclusivamente las variantes SHA3-256, SHA3-512, SHAKE128 y SHAKE256, dado que son las únicas requeridas por el algoritmo de encapsulación de claves post-cuántico ML-KEM (Kyber), estandarizado por el NIST [14].

Se presenta en primer lugar la motivación y los requisitos que han guiado la selección de variantes, así como el enfoque seguido para garantizar la compatibilidad funcional, la eficiencia de recursos y el rendimiento. A continuación, se detalla la estructura general de la arquitectura hardware, el desarrollo y organización del código VHDL, y las distintas variantes arquitectónicas del núcleo KECCAK-f[1600] que han sido evaluadas. Además, se describe el proceso de validación funcional de las implementaciones, incluyendo el desarrollo de testbenches específicos para cada módulo y la comprobación de los resultados frente a valores de referencia externos. También se expone el flujo de implementación en FPGA, con la metodología utilizada para la obtención de métricas de área, temporización y frecuencia máxima.

Esta sección proporciona el contexto y la base técnica necesaria para interpretar los resultados presentados en la Sección 4 y para valorar el impacto de cada decisión de diseño en el rendimiento y la eficiencia del sistema criptográfico resultante.

3.1. Requisitos de diseño

El diseño hardware de SHA3 y SHAKE presentado en este trabajo se ha orientado a cumplir una serie de criterios técnicos alineados con los estándares actuales y las necesidades de la criptografía post-cuántica. Para ello, se han desarrollado y evaluado varias variantes arquitectónicas, como implementaciones combinacionales y segmentadas con distinto número de etapas internas.

Los principales requisitos considerados en este análisis comparativo son los siguientes:

- **Compatibilidad funcional:** todas las variantes implementadas soportan los modos y parámetros principales definidos en FIPS 202 (SHA3-256, SHA3-512, SHAKE128 y SHAKE256).
- **Eficiencia de recursos:** se cuantifica y compara el consumo de recursos hardware (LUTs, flip-flops, slices, etc.) de cada arquitectura.
- **Rendimiento:** se mide y compara el throughput y la latencia de cada variante, analizando el impacto de la segmentación sobre la velocidad final.
- **Escalabilidad:** se considera la facilidad de adaptar cada arquitectura a diferentes requerimientos de área o rendimiento, y su viabilidad para integrar en sistemas criptográficos reales como Kyber o SPHINCS+.

Estos criterios permiten caracterizar de forma objetiva las ventajas y limitaciones de cada alternativa, facilitando la elección del diseño más adecuado según las restricciones del sistema final.

3.2. Descripción general de la arquitectura

La Figura 3 muestra el diagrama de bloques general de la arquitectura hardware implementada para SHA3 y SHAKE. El diseño sigue la estructura funcional definida por el estándar FIPS 202 [13], dividiendo el proceso en varias fases diferenciadas.

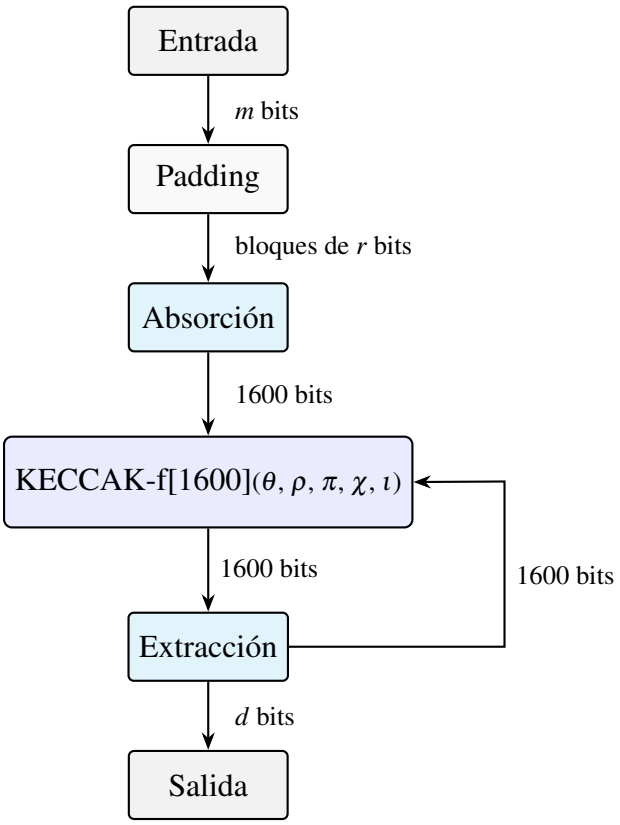

Figura 3: Diagrama de bloques compacto de SHA3/SHAKE.

En primer lugar, el mensaje de entrada se procesa mediante el bloque de padding, asegurando que su longitud sea múltiplo del parámetro r definido por el modo seleccionado (por ejemplo, $r = 1088$ para SHA3-256). El mensaje ya alineado se divide en bloques de r bits, que se absorben secuencialmente en el estado interno de 1600 bits. Esta absorción se realiza mediante operaciones XOR entre cada bloque y una parte del estado, seguida de la aplicación de la permutación KECCAK-f[1600].

El estado interno de 1600 bits constituye el núcleo del algoritmo, y es actualizado iterativamente en cada ronda de absorción y durante el proceso de extracción de salida. El bloque KECCAK-f[1600] implementa las cinco transformaciones internas $(\theta, \rho, \pi, \chi, \iota)$ y puede dividirse en varias etapas según la arquitectura concreta empleada.

La extracción (squeeze) se encarga de generar la salida final. Si la longitud de salida requerida es mayor que *r* bits (como puede ocurrir en SHAKE o en algunos modos de SHA3), se reutiliza el estado interno aplicando nuevamente la función KECCAK-f[1600], permitiendo extraer sucesivos bloques hasta alcanzar la longitud deseada.

Este esquema arquitectónico es común a todas las variantes implementadas (secuencial, segmentada, etc.) y proporciona la base para las comparativas de área, latencia y rendimiento desarrolladas en los apartados siguientes.

3.3. Estructura y desarrollo del código VHDL

Se ha desarrollado todo el código VHDL empleado en este trabajo, implementando desde cero los distintos bloques funcionales, de control y verificación necesarios para la realización y evaluación de las variantes arquitectónicas. Se ha seguido una estrategia de desarrollo modular y jerárquica, permitiendo la máxima flexibilidad, reutilización y facilidad de prueba de los componentes principales.

La funcionalidad central de los algoritmos SHA3 y SHAKE se divide en varios archivos y entidades VHDL independientes, estructurados del siguiente modo:

- **KECCAK_round**: Implementa una ronda completa del algoritmo KECCAK, con la posibilidad de configurar y probar diferentes arquitecturas segmentadas, como se muestra en la Subsección 3.5. Cada una de las transformaciones fundamentales (θ, ρ, π, χ, ι) se codifica como una función separada, permitiendo agruparlas de manera flexible en etapas distintas según la variante arquitectónica que se desee estudiar.

- **KECCAK_f_24**: Se encarga de aplicar secuencialmente las 24 rondas de KECCAK sobre el estado de 1600 bits. Incorpora el control de flujo para ejecutar la instancia de *KECCAK_round* en modo iterativo, gestionando las señales de inicio, fin y avance de ronda.

- **sponge**: Implementa la construcción sponge, integrando tanto el padding (incluido el sufijo y el relleno conforme al estándar) como el proceso de absorción y extracción de bloques de datos. Este módulo es completamente parametrizable y permite seleccionar tanto la longitud de entrada como el tamaño de la salida hash, lo que lo hace compatible con todas las variantes SHA3 (*SHA3-256*, *SHA3-512*) y SHAKE (*SHAKE128*, *SHAKE256*).

- **sha3_xxx / shake_xxx**: Estos módulos construyen las funciones SHA3 y SHAKE parametrizadas (*sha3_256*, *sha3_512*, *shake128*, *shake256*), empleando el módulo sponge con los parámetros apropiados para cada estándar (tamaño de entrada, longitud de salida, sufijo, etc.).

- **top_sha3_xxx / top_shake_xxx**: Proporcionan un entorno de integración y prueba, conectando la entrada de mensaje, la señal de inicio y la recogida de la salida hash o salida

extendida (squeeze). Permiten verificar el funcionamiento completo de la cadena hash o extendida desde un único archivo de nivel superior.

Cada módulo funcional se ha documentado y separado en archivos independientes, favoreciendo la claridad del código. El diseño modular permite la sustitución o modificación de componentes individuales sin afectar al resto del sistema.

Para las variantes SHAKE, se ha desarrollado una versión específica del módulo sponge adaptada para soportar una longitud de salida variable, tal y como requiere el estándar. Aunque la estructura general del módulo se mantiene, el mecanismo de extracción y control del truncamiento se ajusta para permitir la generación de una cantidad arbitraria de bits en la fase squeeze.

Este enfoque de desarrollo propio y modular no solo garantiza la completa comprensión y dominio del funcionamiento interno, sino que también ofrece una base sólida para futuras ampliaciones o integraciones con otros sistemas criptográficos.

3.4. Validación funcional mediante testbenches

Para garantizar la corrección funcional de todas las variantes implementadas (SHA3-256, SHA3-512, SHAKE128 y SHAKE256), se han desarrollado diferentes testbenches en VHDL adaptados a cada módulo y configuración. Estos testbenches automatizan el proceso de simulación, aplicando mensajes de prueba, gestionando el arranque y el reset.

Como referencia para la validación, se han utilizado tanto vectores de prueba oficiales como resultados generados por herramientas online como [10]. De este modo, en cada simulación se ha comparado la salida calculada por el hardware frente a los resultados ofrecidos por implementaciones ampliamente verificadas de SHA3/SHAKE.

Este enfoque de verificación exhaustiva proporciona la seguridad de que todos los módulos implementados son correctos desde el punto de vista funcional antes de proceder a la síntesis e implementación física en FPGA.

3.5. Variantes arquitectónicas implementadas

El núcleo KECCAK-f[1600] constituye el bloque computacionalmente más exigente en la arquitectura SHA3/SHAKE. Todas las operaciones de mezcla y permutación sobre el estado interno de 1600 bits se realizan mediante la secuencia de cinco transformaciones principales (θ, ρ, π, χ, ι) en cada ronda. La eficiencia global de la implementación, tanto en términos de latencia como de área ocupada, depende de la forma en la que se estructura internamente este bloque.

Si el núcleo se implementa de forma completamente combinacional, todas las transformaciones se realizan en una única etapa y dentro de un solo ciclo de reloj por ronda. Esto minimiza el área, pero puede limitar significativamente la frecuencia máxima alcanzable, debido a la acumulación de retardos lógicos.

Para mejorar el rendimiento, se puede segmentar el núcleo KECCAK-f en varias etapas, insertando registros intermedios entre grupos de transformaciones. Este enfoque, conocido como segmentación, permite acortar el recorrido crítico de cada ciclo de reloj y elevar la frecuencia máxima de operación. A cambio, el área ocupada aumenta, ya que se requieren más registros y lógica de control para gestionar el flujo de datos entre etapas.

A continuación se muestran varias variantes arquitectónicas del núcleo KECCAK-f:

- **Arquitectura combinacional**: Todas las transformaciones (θ, ρ, π, χ, ι) se realizan en una única etapa lógica, sin registros intermedios. Esta variante minimiza el consumo de recursos, pero puede presentar un recorrido crítico elevado.

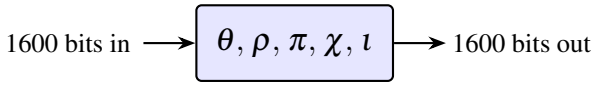

Figura 4: Arquitectura combinacional: todas las transformaciones en una única etapa lógica.

- **Segmentación de dos etapas**: El núcleo se divide en dos grupos de transformaciones, separados por un banco de registros. Por ejemplo, las tres primeras transformaciones pueden realizarse en la primera etapa y las dos restantes en la segunda. Esta organización permite reducir el recorrido crítico respecto a la variante combinacional, a costa de un mayor uso de registros.

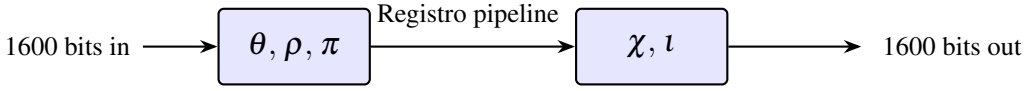

Figura 5: Segmentación de dos etapas: θ, ρ y π en la primera etapa, el resto en la segunda.

- **Segmentación de tres o más etapas**: El núcleo se segmenta en tres o más etapas, insertando registros tras cada transformación principal o grupo reducido de transformaciones. Cuanto mayor es el número de etapas, menor es el recorrido crítico, lo que posibilita operar a frecuencias más altas. Sin embargo, esto incrementa progresivamente el área ocupada por registros y lógica adicional.

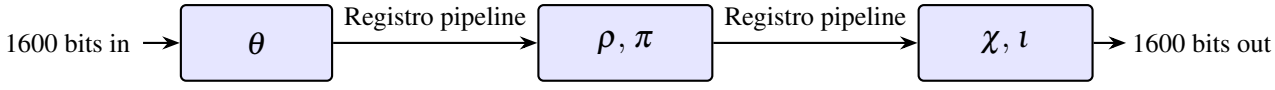

Figura 6: Segmentación de tres etapas: θ en la primera, ρ y π en la segunda y χ y ι en la tercera.

Cada una de estas variantes representa un compromiso diferente entre área y rendimiento. En este trabajo se han seleccionado nueve arquitecturas para compararlas entre sí. Estas etapas se muestran en la Tabla 4. Para describir las diferentes variantes arquitecturales implementadas, se

Tabla 4: Correspondencia entre variantes arquitecturales y agrupación de funciones.

Variante	Descripción	Agrupación de funciones
Combinacional	Una sola etapa, implementación completamente combinacional.	$[\theta, \rho, \pi, \chi, \iota]$
2-3	Dos etapas: dos funciones en la primera, tres en la segunda.	$[\theta, \rho] - [\pi, \chi, \iota]$
3-2	Dos etapas: tres funciones en la primera, dos en la segunda.	$[\theta, \rho, \pi] - [\chi, \iota]$
3-1-1	Tres etapas: tres funciones en la primera, una en la segunda, una en la tercera.	$[\theta, \rho, \pi] - [\chi] - [\iota]$
2-2-1	Tres etapas: dos funciones en la primera, dos en la segunda, una en la tercera.	$[\theta, \rho] - [\pi, \chi] - [\iota]$
1-2-2	Tres etapas: una función en la primera, dos en la segunda, dos en la tercera.	$[\theta] - [\rho, \pi] - [\chi, \iota]$
2-1-1-1	Cuatro etapas: dos funciones en la primera, luego tres etapas de una función.	$[\theta, \rho] - [\pi] - [\chi] - [\iota]$
1-1-1-2	Cuatro etapas: tres etapas de una función, luego una de dos.	$[\theta] - [\rho] - [\pi] - [\chi, \iota]$
1-1-1-1-1	Cinco etapas: segmentación de una función por etapa.	$[\theta] - [\rho] - [\pi] - [\chi] - [\iota]$

utiliza la notación **X-Y-Z-...**, donde cada número indica el número de funciones agrupadas en cada etapa de la segmentación.

La selección de estas variantes arquitectónicas responde a la intención de cubrir un espectro representativo de posibles implementaciones hardware del núcleo KECCAK-f[1600]. Se han incluido tanto la arquitectura completamente combinacional (mínima en registros y lógica de control, pero con el mayor recorrido crítico), como arquitecturas segmentadas de dos, tres, cuatro y cinco etapas, agrupando las funciones internas en diferentes combinaciones. Esto permite analizar de forma comparativa cómo afecta el reparto de las transformaciones entre etapas al rendimiento, la frecuencia máxima y el consumo de recursos.

La mejor arquitectura dependerá de las restricciones y requisitos del sistema en el que se vaya a integrar la función SHA3/SHAKE. En la sección 4 se presentan los resultados comparativos de implementación y rendimiento obtenidos para cada arquitectura implementada.

3.6. Implementación en FPGA

La implementación de las distintas variantes arquitectónicas se ha realizado sobre la FPGA Xilinx Artix-7 XC7A100T, utilizando la herramienta de síntesis y place & route de Vivado 2023.2. Todas las descripciones hardware se han desarrollado en VHDL, siguiendo una metodología modular. Cada función principal del algoritmo (θ, ρ, π, χ, ι) se ha implementado como un

módulo independiente, permitiendo así una reconfiguración sencilla de la estructura interna de la segmentación y la agrupación de funciones en cada etapa.

El código se ha organizado en varios archivos VHDL diferenciados, separando claramente los módulos funcionales de los testbench de validación. Además, para facilitar el intercambio de variantes, se ha empleado un archivo de top-level parametrizable, como se ha explicado en la Subsección 3.3.

El proceso de implementación en Vivado ha seguido las siguientes fases:

- Descripción y verificación funcional en simulación de cada variante arquitectónica, empleando testbenches específicos.
- Síntesis, implementación (place & route) y análisis de resultados mediante los informes automáticos de utilización de recursos y temporización generados por Vivado.
- Extracción de las métricas principales: número de LUTs, FFs y Worst Negative Slack (WNS).

En todos los casos, se han utilizado los mismos parámetros de síntesis y constraints. Para todas las pruebas se ha fijado inicialmente un periodo de reloj objetivo de 5,5 ns. Todos los valores de temporización y área presentados se han obtenido directamente a partir de los informes generados por la herramienta de implementación tras cada síntesis y place & route.

Es importante matizar cómo se determina el periodo de reloj límite alcanzable por cada arquitectura. Teóricamente, el periodo de reloj mínimo se podría estimar restando el valor del Worst Negative Slack (WNS) al periodo de reloj impuesto en la implementación, ya que el Worst Negative Slack (WNS) representa la diferencia entre el periodo de reloj impuesto y el retardo del camino crítico del diseño.

Sin embargo, en la práctica esto no es exacto, ya que las herramientas de implementación (como Vivado) adaptan el esfuerzo de optimización en función del periodo de reloj objetivo especificado. Cuanto más restrictivo es el tiempo de reloj, más recursos y optimizaciones aplicará Vivado para intentar cumplir la restricción, lo que puede dar lugar a implementaciones diferentes aunque el diseño fuente sea el mismo.

Por ejemplo, para uno de los ejemplos de la Tabla 4, concretamente para la arquitectura 2-3 de SHA3-256, si se fija un periodo de reloj de 10 ns se obtiene un WNS de 1,875 ns, lo que significa que el retardo del camino crítico es realmente de 8,125 ns. Sin embargo, si se reduce el periodo de reloj impuesto a 5,1 ns, tras la implementación se obtiene un WNS de 0,001 ns, obteniendo un retardo del camino crítico real (tiempo de reloj crítico) de 5,099 ns.

Por este motivo, obtener un WNS positivo para un determinado periodo simplemente indica que la implementación cumple la restricción de temporización, pero no garantiza que se haya alcanzado la máxima frecuencia posible.

Para determinar el periodo de reloj mínimo (y, por tanto, la máxima frecuencia de funcionamiento alcanzable), se ha seguido un procedimiento iterativo: tras cada síntesis e implementación,

se ajusta el valor del periodo de reloj objetivo y se repite el proceso, afinando hasta encontrar aquel valor para el cual el WNS es lo más cercano posible a cero, menor que 0,1 ns (pero no negativo). De este modo, se obtiene una estimación precisa de la frecuencia máxima real que puede alcanzar cada arquitectura, independientemente de optimismos en los márgenes de temporización. Este enfoque garantiza una comparación justa y realista del rendimiento de las distintas variantes.

Antes de presentar los resultados obtenidos, conviene destacar que, entre las cinco funciones que conforman la permutación KECCAK-f, las transformaciones θ y χ son, en términos de lógica combinacional, las más complejas. La función θ implica operaciones de suma y desplazamiento entre todas las columnas del estado, mientras que χ aplica una operación no lineal bit a bit sobre cada fila, introduciendo dependencia entre bits del mismo plano. Debido a esta complejidad, es razonable anticipar que las arquitecturas que segmenten de forma más precisa estas funciones, es decir, que eviten agruparlas con otras en la misma etapa de la segmentación, podrían alcanzar mayores frecuencias de funcionamiento. Así, se espera que configuraciones como 3-1-1 o 2-3, que aíslan parcialmente θ y χ, ofrezcan un mejor compromiso entre rendimiento y coste en área.

4. Resultados y Análisis

En esta sección se presentan y analizan los resultados obtenidos tras la implementación de las diferentes variantes arquitectónicas desarrolladas para las variantes de SHA3 y SHAKE. El objetivo principal es evaluar el impacto que tiene cada estrategia de segmentación en el rendimiento, área y eficiencia global del diseño hardware. Para ello, se comparan distintas configuraciones en términos de frecuencia máxima alcanzada, recursos utilizados y throughput, permitiendo extraer conclusiones sobre las ventajas y limitaciones de cada enfoque.

La metodología seguida para la obtención de los resultados mostrados en esta sección es la descrita previamente en la Sección 3.6. En particular, se ha utilizado el entorno de desarrollo Vivado 2023.2 para sintetizar las distintas variantes arquitectónicas, y se ha ajustado iterativamente el periodo de reloj hasta alcanzar un valor de Worst Negative Slack (WNS) próximo a cero. Este procedimiento permite determinar de forma precisa la frecuencia máxima que puede alcanzar cada diseño, garantizando que los resultados sean representativos del rendimiento real en implementación física. Todos los resultados implementados se han obtenido tras la síntesis y el proceso de place-and-route utilizando los ajustes predeterminados de Vivado.

4.1. Métricas de Evaluación

Con el fin de valorar objetivamente el rendimiento y la eficiencia de cada una de las arquitecturas implementadas, se han definido una serie de métricas clave que permiten comparar los resultados desde diferentes perspectivas. Estas métricas no solo reflejan el comportamiento del diseño en términos de velocidad, sino también su coste en recursos y su adecuación a posibles restricciones impuestas por el entorno hardware.

Dado que el objetivo del proyecto es optimizar tanto el rendimiento como el área en FPGA, las métricas seleccionadas permiten identificar configuraciones que ofrecen un buen compromiso entre ambos factores. A continuación, se describen detalladamente las métricas utilizadas en este análisis:

- **Área ocupada**: se mide en términos del número de Look-Up Tables (LUTs), Flip-Flops (FFs) y slices utilizados.
    - Una **LUT** es un bloque de lógica combinacional configurable que implementa funciones booleanas. El número total de LUTs da una estimación directa de la complejidad lógica del diseño.
    - Un **Flip-Flop (FF)** es un elemento secuencial que almacena un bit. Se utiliza para sincronizar datos o implementar máquinas de estados. El número de FFs utilizados indica la cantidad de lógica secuencial presente.
    - Un **slice** es una unidad física del FPGA que agrupa varios LUTs y FFs (por ejemplo, hasta 4 LUTs y 8 FFs en la arquitectura Artix-7). Por tanto, el número total de slices

utilizados refleja la cantidad de bloques físicos ocupados:

$$\text{slices utilizados} \geq \text{máx}\left(\frac{\text{LUTs}}{\text{LUTs por slice}}, \frac{\text{FFs}}{\text{FFs por slice}}\right).$$

- **Frecuencia máxima alcanzada**: es la mayor frecuencia de reloj para la cual el diseño cumple con todas las restricciones temporales. Se determina obteniendo un valor de Worst Negative Slack (WNS) mayor o igual que cero, y lo más cercano a cero posible:

$$f_{\text{max}} = \frac{1}{T_{\text{clk}}} \quad \text{siempre que WNS} \geq 0, \tag{1}$$

  donde T_{clk} es el periodo de reloj mínimo alcanzable sin violar restricciones.

- **Throughput**: mide la cantidad de datos procesados por segundo. Para un diseño basado en la función sponge como SHAKE o SHA3, donde se extraen r bits cada N ciclos de reloj, se define como:

$$\text{Thpt} = \frac{r \cdot f_{\text{max}}}{N}, \tag{2}$$

  donde:

  - r: número de bits producidos por bloque de salida. Para SHA3-256, el parámetro de absorción es $r = 1088$, por lo que se procesan 1088 bits por cada llamada al núcleo KECCAK. Este valor determina cuántos bits se absorben o extraen por bloque.
  - N: número de ciclos de reloj requeridos para completar la operación, es decir, para ejecutar las 24 rondas de la permutación KECCAK-f[1600]. Este valor depende directamente de la arquitectura del diseño.

    Por ejemplo, si cada ronda se implementa de forma completamente combinacional, se tendría $N = 1$. Si cada ronda se ejecuta secuencialmente en un ciclo (una por ciclo), entonces $N = 24$.

    En implementaciones con segmentación parcial, donde cada ronda se divide en varias etapas que requieren múltiples ciclos, el número de ciclos se incrementa. Por ejemplo, si cada ronda se ejecuta en dos ciclos (una etapa por ciclo), el total sería:

$$N = 24 \times 2 = 48.$$

  - f_{max}: frecuencia de reloj máxima alcanzada, expresada en Hz.

- **Eficiencia**: se define como la relación entre el throughput y el área ocupada (por ejemplo, en Slices). Esta métrica cuantifica cuántos megabits por segundo es capaz de procesar el diseño por cada unidad lógica utilizada:

$$\text{Eff} = \frac{\text{Thpt}}{\text{Slices}} \quad \left[\frac{\text{Mbps}}{\text{Slice}}\right]. \tag{3}$$

Cuanto mayor sea este valor, mejor aprovechamiento se hace de los recursos lógicos del FPGA en términos de rendimiento.

La eficiencia también puede calcularse en función de otros recursos, como los Flip-Flops (FFs) o las LUTs, si se desea evaluar el aprovechamiento de lógica secuencial o de los bloques físicos ocupados, respectivamente:

$$\text{Eff (por FF)} = \frac{\text{Thpt}}{\text{FFs}}; \qquad \text{Eff (por LUT)} = \frac{\text{Thpt}}{\text{LUTs}}.$$

4.2. Resultados Obtenidos

Antes de presentar los resultados de implementación, conviene recordar la organización interna de cada arquitectura. En la Tabla 4 se resume cómo se reparten las funciones θ, ρ, π, χ e ι en cada una de las configuraciones analizadas. Para cada una de ellas se recogen las métricas descritas anteriormente: área ocupada, frecuencia máxima alcanzada y throughput. Estos resultados se han extraído directamente del informe de implementación generado por Vivado 2023.2.

4.2.1. SHA3-256

En la Tabla 5 se presentan los resultados de implementación obtenidos para el algoritmo SHA3-256 implementado con distintas arquitecturas. Se muestra el número de recursos utilizados en términos de LUTs, FFs y slices, así como la frecuencia máxima alcanzada tras la implementación. A partir de esta frecuencia y del número de ciclos necesarios para completar una operación, se calcula el rendimiento (Throughput) en megabits por segundo , teniendo en cuenta que $r = 1088$, junto con una métrica de eficiencia relativa expresada en Gbps por unidad de área lógica (slice).

Como se puede observar, el número de LUTs permanece relativamente constante entre arquitecturas, mientras que el número de FFs aumenta progresivamente a medida que se introducen más etapas de segmentación. La frecuencia máxima de funcionamiento también tiende a incrementarse con arquitecturas más segmentadas. En este caso se ha obtenido una frecuencia máxima para la arquitectura 3-1-1, obteniendo 198,22 MHz. Sin embargo, el rendimiento en Mbps y la eficiencia muestran una tendencia decreciente en la mayoría de las arquitecturas más profundas. Estos resultados reflejan cómo las decisiones arquitectónicas afectan simultáneamente a los recursos empleados y al rendimiento alcanzado.

Tabla 5: Resultados de síntesis para SHA3-256 ($r = 1088$) en distintas arquitecturas.

Arquitectura	N (ciclos)	LUTs	FFs	Slices	Freq. (MHz)	Thpt (Mbps)	Eff. (Gbps/Area)
Combinacional	24	6615	4867	1785	189,79	8,604	4,820
3-2	48	6556	8156	2010	196,12	4,445	2,212
2-3	48	6554	8154	1992	197,04	4,466	2,242
2-2-1	72	6556	9754	2231	196,16	2,964	1,329
1-2-2	72	6554	9755	2251	196,31	2,966	1,318
3-1-1	72	6597	9794	2258	198,22	2,995	1,326
2-1-1-1	96	6559	11357	2488	192,68	2,176	0,875
1-1-1-2	96	6558	11394	2492	194,93	2,207	0,886
1-1-1-1-1	120	6535	12952	2904	194,55	1,763	0,607

4.2.2. SHA3-512

La Tabla 6 recoge los resultados obtenidos para las distintas arquitecturas implementadas del algoritmo SHA3-512. Se incluyen, como en el caso anterior, el número de recursos lógicos utilizados (LUTs, FFs y slices), la frecuencia máxima alcanzada tras la implementación y el rendimiento expresado en Mbps.

En este caso se observa un patrón similar al que se ha obtenido para SHA3-256: mientras que el número de LUTs se mantiene en valores próximos entre las distintas versiones arquitectónicas, el número de flip-flops aumenta considerablemente a medida que se incrementa la segmentación. En este caso, el valor de la frecuencia máxima se ha obtenido para la arquitectura `2-1-1-1`, obteniendo una frecuencia de 197,36 MHz.

Tabla 6: Resultados de síntesis para SHA3-512 ($r = 576$) en distintas arquitecturas.

Arquitectura	N (ciclos)	LUTs	FFs	Slices	Freq. (MHz)	Thpt (Mbps)	Eff. (Gbps/Area)
Combinacional	24	5065	4868	1473	188,68	4,532	3,077
3-2	48	6354	8101	1845	195,84	2,350	1,274
2-3	48	6353	8101	1861	192,12	2,305	1,239
2-2-1	72	6357	9703	2037	192,20	1,538	0,755
1-2-2	72	6344	9693	2110	190,01	1,520	0,720
3-1-1	72	6356	9701	2029	194,36	1,554	0,766
2-1-1-1	96	5913	11308	2307	197,36	1,184	0,513
1-1-1-2	96	5921	11319	2299	195,35	1,172	0,510
1-1-1-1-1	120	5906	12902	2809	192,23	0,923	0,329

4.2.3. SHAKE128

En la Tabla 7 se recogen los resultados obtenidos para las distintas arquitecturas evaluadas en la implementación del algoritmo SHAKE128. Al igual que en los casos anteriores, se presentan los valores de recursos utilizados, la frecuencia máxima alcanzada y el rendimiento expresado

en Mbps, así como una métrica de eficiencia relativa, que en este caso se ha definido como el cociente entre el rendimiento y el número de LUTs (Mbps/LUT).

Los datos muestran un comportamiento similar a los observados anteriormente. El número de LUTs se mantiene aproximadamente constante entre arquitecturas, mientras que el número de flip-flops crece de forma significativa conforme se incrementa el número de etapas de la segmentación. En consecuencia, aunque la frecuencia tiende a aumentar, el rendimiento disminuye progresivamente, afectando también a la eficiencia relativa. En este caso, la frecuencia máxima se ha obtenido para la arquitectura 3-1-1, obteniendo una frecuencia de 213,56 MHz.

Tabla 7: Resultados de síntesis para SHAKE128 ($r = 1344$) en distintas arquitecturas.

Arquitectura	N (ciclos)	LUTs	FFs	Slices	Freq. (MHz)	Thpt (Mbps)	Eff. (Gbps/Area)
Combinacional	24	7611	4893	1473	206,23	11,549	7,840
3-2	48	7621	8160	2025	212,40	5,947	2,937
2-3	48	7628	8167	2042	210,13	5,883	2,881
2-2-1	72	7639	9782	2275	213,40	3,983	1,751
1-2-2	72	7601	9737	2241	202,68	3,783	1,688
3-1-1	72	7648	9777	2219	213,56	3,982	1,795
2-1-1-1	96	7663	11394	2486	211,10	2,955	1,205
1-1-1-2	96	7657	11391	2515	210,88	2,952	1,174
1-1-1-1-1	120	7031	13038	2971	210,04	2,352	0,792

4.2.4. SHAKE256

La Tabla 8 muestra los resultados obtenidos para la implementación del algoritmo SHAKE256 en diversas arquitecturas con distintos niveles de segmentación. Al igual que en los casos anteriores, se presenta el uso de recursos lógicos (LUTs, FFs y slices), la frecuencia máxima alcanzada, el rendimiento en Mbps, y una métrica de eficiencia relativa, expresada como Gbps por unidad de área.

Los valores de LUTs permanecen relativamente constantes entre las distintas versiones, mientras que el número de flip-flops aumenta de forma notable en aquellas arquitecturas con una mayor profundidad de la segmentación. En este caso, la frecuencia máxima se ha obtenido para la arquitectura 1-1-1-1-1, con un valor de 209,56 MHz.

Tabla 8: Resultados de síntesis para SHAKE256 ($r = 1088$) en distintas arquitecturas.

Arquitectura	N (ciclos)	LUTs	FFs	Slices	Freq. (MHz)	Thpt (Mbps)	Eff. (Gbps/Area)
Combinacional	24	7423	4891	1924	197,82	8,968	5,661
3-2	48	7361	8162	2054	205,47	4,657	2,267
2-3	48	7360	8161	2011	208,56	4,728	2,351
2-2-1	72	6844	9785	2247	208,51	3,151	1,402
1-2-2	72	6792	9735	2205	198,33	2,997	1,359
3-1-1	72	6765	9713	2336	201,53	3,045	2,177
2-1-1-1	96	6843	11380	2583	201,96	2,300	0,890
1-1-1-2	96	6851	11395	2580	203,05	2,301	0,892
1-1-1-1-1	120	6859	12997	2846	209,56	1,900	0,668

4.3. Análisis de los Resultados

Una vez obtenidos los resultados de implementación para las distintas arquitecturas y variantes del algoritmo SHA3 y SHAKE, es posible realizar un análisis comparativo global que permita extraer conclusiones sólidas sobre el impacto de la segmentación en el rendimiento y la eficiencia de cada función.

4.3.1. Análisis de SHA3

Tendencias en SHA3-256 En el caso de SHA3-256, se observa que las arquitecturas parcialmente segmentadas como `3-1-1` y `2-3` permiten alcanzar las mayores frecuencias, con valores de hasta 198,22 MHz y 197,04 MHz respectivamente. Este comportamiento se explica por el hecho de que estas configuraciones consiguen aislar las funciones más costosas en términos de retardo lógico (θ y χ) sin introducir una arquitectura excesivamente segmentada. La función θ presenta un retardo elevado debido a su dependencia de todos los bits de cada columna del estado, mientras que χ, al aplicar operaciones no lineales entre bits adyacentes, también contribuye de forma significativa al retardo crítico. Cuando estas funciones se agrupan con otras más ligeras, el retardo total de la etapa correspondiente tiende a limitar la frecuencia alcanzable.

Por el contrario, al aumentar el número de etapas más allá de tres, como en las arquitecturas `2-1-1-1` o `1-1-1-1-1`, no se observan mejoras adicionales en la frecuencia, e incluso se produce un descenso en esta. Esto se debe al sobrecoste introducido por los registros intermedios y la lógica de control de la segmentación [7], que penaliza el rendimiento global cuando la segmentación no aporta una mejora clara del camino crítico.

Desde el punto de vista del área, se obtiene lo que ya se esperaba. Las arquitecturas con más etapas presentan un incremento considerable en el número de slices utilizados, alcanzando hasta 2904 en `1-1-1-1-1`, frente a los 1785 de la versión combinacional. Este crecimiento en recursos no se traduce necesariamente en un aumento proporcional del rendimiento, y penaliza notablemente la eficiencia relativa. En este sentido, las configuraciones `2-3` y `3-2` destacan por ofrecer una buena relación entre frecuencia y área, con eficiencias superiores a 2 Gbps/slice.

Tendencias en SHA3-512 En este caso, el comportamiento difiere del observado en SHA3-256, donde a partir de tres etapas la frecuencia comenzaba a disminuir debido al impacto del aumento de registros intermedios. En SHA3-512, sin embargo, la frecuencia máxima se alcanza precisamente en una arquitectura con cuatro etapas (`2-1-1-1`), logrando un valor de 197,36 MHz.

En primer lugar, se observa que las arquitecturas de dos etapas, `3-2` y `2-3`, presentan una mejora de frecuencia considerable respecto a la versión combinacional (188,68 MHz), alcanzando 195,84 MHz y 192,12 MHz respectivamente. A partir de la arquitectura `3-2`, si se segmenta el segundo bloque (que contiene χ y ι) en dos etapas, dando lugar a la arquitectura `3-1-1`, la frecuencia no mejora e incluso se reduce ligeramente a 194,36 MHz. Esto sugiere que el camino crítico se encuentra en ese primer bloque de tres funciones. Esta hipótesis se confirma al dividir dicho bloque en dos etapas (arquitectura `2-1-1-1`), lo cual sí proporciona una mejora notable de frecuencia, alcanzando el mejor valor de frecuencia.

Por otro lado, si partimos de la arquitectura `2-3`, que agrupa π, χ e ι en un solo bloque, se observa que su frecuencia es inferior a la de `3-2`, lo que sugiere que en este caso el retardo de χ puede ser mayor que el de θ. Al dividir ese bloque en dos etapas (`2-2-1`), se obtiene una ligera mejora de frecuencia (192,20 MHz), a pesar de introducir más registros. Al segmentar aún más, dando lugar a la arquitectura `2-1-1-1`, donde χ e ι se ejecutan en etapas separadas, se alcanza una frecuencia de 197,36 MHz, superior a las anteriores.

Comparando las arquitecturas `1-1-1-2` y `2-1-1-1`, se observa que esta última alcanza una frecuencia mayor, lo que indica que el retardo de $\chi + \iota$ (que permanecen agrupadas en `1-1-1-2`) es superior al de $\theta + \rho$ (agrupadas en `2-1-1-1`). Por último, al segmentar completamente el diseño en la arquitectura `1-1-1-1-1`, no se aprecia una mejora de frecuencia respecto a las anteriores, lo que sugiere que esta última división no aporta beneficios adicionales y que el retardo ya estaba suficientemente equilibrado en las configuraciones previas.

En cuanto al área, se obtiene el comportamiento esperado: las arquitecturas con mayor segmentación presentan un incremento progresivo en el número de slices y flip-flops utilizados. No obstante, al igual que en el caso anterior, este crecimiento no se traduce en una mejora proporcional del rendimiento, y penaliza la eficiencia relativa cuando la segmentación es excesiva.

Sha3: 256 vs 512 Al comparar SHA3-256 con SHA3-512, una de las diferencias más relevantes es la forma en la que responde cada una a la segmentación. En SHA3-256, la frecuencia obtenida no depende tanto de la arquitectura específica empleada, sino más bien de que las funciones θ y χ estén separadas y de que el número de etapas no sea excesivo. De hecho, siempre que estas dos funciones críticas se encuentren en bloques distintos y se evite una segmentación demasiado profunda, la frecuencia alcanzada es bastante elevada.

En cambio, en SHA3-512 la arquitectura concreta sí influye de forma más decisiva. No basta con separar únicamente θ y χ para garantizar un buen rendimiento, es necesario que la distribución global de la segmentación esté equilibrada. Esta diferencia puede atribuirse a que,

en SHA3-512, el retardo crítico está más distribuido entre varias funciones, y no tan concentrado en una sola como ocurre en SHA3-256 con θ. Aunque las funciones ρ, π e ι no cambian estructuralmente entre versiones, su impacto relativo en el retardo total es mayor en SHA3-512, ya que ninguna etapa destaca de forma clara sobre las demás. Como consecuencia, las etapas 2, 3 y 5 (correspondientes a las funciones ρ, π e ι) toman un mayor peso, lo que hace que el rendimiento dependa no solo de separar las funciones más complejas, sino de cómo se equilibran todas las etapas en el diseño de la segmentación.

Esta diferencia se observa bien en la Figura 7, donde para SHA3-256 se obtienen frecuencias altas para arquitecturas de menos de cuatro etapas, mientras que para SHA3-512 depende más de la arquitectura en concreto que se esté usando.

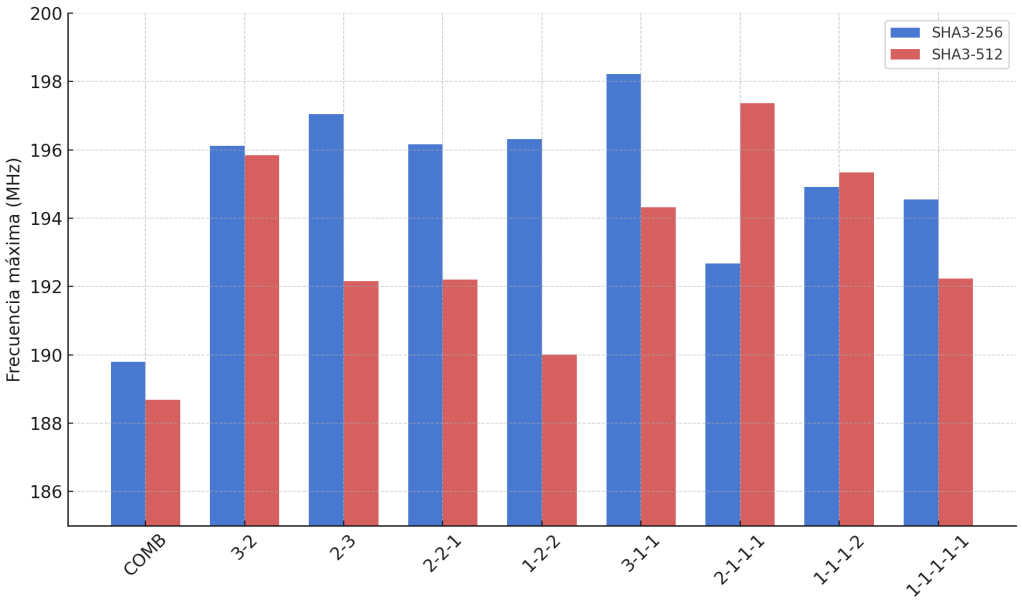

Figura 7: frecuencia para diferentes arquitecturas de SHA3-256 y SHA3-512.

4.3.2. Análisis de SHAKE

Tendencias en SHAKE128 En el caso de SHAKE128, se observa que la arquitectura `3-2` alcanza una frecuencia de 212,40 MHz, superior a los 210,13 MHz obtenidos con `2-3`. Esta diferencia sugiere que el cuello de botella se encuentra en el segundo bloque, que agrupa las funciones χ e ι. Esta hipótesis se refuerza al segmentar la arquitectura `3-2`, pasando a la `3-1-1`, donde se segmenta dicho bloque y se obtiene la frecuencia más alta registrada para este algoritmo, de 213,56 MHz. A partir de esta arquitectura, si se continúa la segmentación dividiendo también el bloque inicial a `2-1-1-1` la frecuencia desciende a 211,10 MHz. Esta reducción indica que la nueva segmentación no compensa el coste adicional en registros y control lógico.

Siguiendo otro camino, si se parte de la arquitectura `2-3` (210,13 MHz) y se segmenta el segundo bloque en dos partes, se obtiene la configuración `2-2-1`, con una frecuencia ligeramente superior de 213,40 MHz, similar a la mejor. Esto concuerda con la idea de que χ e ι son respon-

sables del retardo principal. Sin embargo, si se continúa esta segmentación, dividiendo también el bloque inicial (arquitectura `1-1-1-2`), la frecuencia disminuye a 211,10 MHz, evidenciando de nuevo que el aumento de registros no aporta una mejora proporcional del rendimiento.

Finalmente, al comparar `2-1-1-1` (211,10 MHz) y `1-1-1-2` (210,88 MHz), se observa que la primera ofrece un ligero mejor rendimiento. Esto confirma que, en SHAKE128, el retardo crítico está más asociado al bloque $\chi + \iota$, aunque la diferencia sea mínima. Es por ello que una segmentación eficiente debe centrarse en separarlo sin introducir excesiva fragmentación en el resto del diseño. Por último, la arquitectura completamente segmentada en cinco etapas alcanza una frecuencia de 210,04 MHz, algo inferior a las de cuatro etapas. Esta diferencia relativamente pequeña se explica porque las funciones ρ e ι, que se aíslan en la quinta etapa, introducen un retardo muy bajo. Por tanto, el coste adicional en registros y control no se ve compensado por una mejora significativa en el retardo crítico.

Tendencias en SHAKE256 En SHAKE256, se observa que la arquitectura `3-2` alcanza una frecuencia de 205,47 MHz, inferior a la obtenida por `2-3` (208,56 MHz). Esta diferencia sugiere que el cuello de botella se encuentra en la primera etapa, es decir, en la función θ. Esta hipótesis se refuerza al analizar la arquitectura `3-1-1`, donde se segmenta el segundo bloque de la segmentación, donde la frecuencia disminuye aún más hasta los 201,53 MHz, lo que confirma que el bloque crítico no se encuentra al final. Posteriormente, al segmentar también el bloque inicial y pasar a `2-1-1-1`, se obtiene una frecuencia prácticamente idéntica (201,96 MHz), lo que indica que la mayor segmentación no ha logrado mejorar el rendimiento, probablemente por no haber suficiente retardo que compensar frente al costo adicional de registros y control.

Por otro lado, si se parte de `2-3` y se segmenta el segundo bloque, obteniendo la arquitectura `2-2-1`, la frecuencia apenas varía (208,52 MHz), lo que también sugiere que el retardo en ese bloque no es determinante. Al continuar la segmentación hasta `2-1-1-1`, se alcanza una frecuencia inferior de 201,96 MHz. La comparación entre `2-1-1-1` (201,96 MHz) y `1-1-1-2` (203,05 MHz) refuerza la idea de que el retardo en θ es dominante respecto al bloque final.

Finalmente, a diferencia de lo observado en SHA3-256 o SHAKE128, en SHAKE256 la arquitectura completamente segmentada `1-1-1-1-1` sí proporciona una mejora clara, alcanzando la mejor frecuencia registrada, de 209,56 MHz. Esto indica que en este caso funciones como ρ e ι, que normalmente tienen un impacto marginal, podrían estar introduciendo mayor retardo relativo, haciendo que una segmentación completa resulte beneficiosa.

SHAKE: 128 vs 256 Al comparar SHAKE128 con SHAKE256, se aprecia una diferencia significativa en la influencia que tiene la arquitectura sobre la frecuencia máxima alcanzada. En SHAKE128, las distintas variantes arquitectónicas obtienen resultados muy similares en cuanto a frecuencia, con valores que oscilan ligeramente entre 210 y 213 MHz. Esto sugiere que en SHAKE128 el retardo crítico está muy localizado, probablemente en las funciones θ y χ, y que, mientras estas se encuentren correctamente separadas, la segmentación del resto de funciones

no tiene un impacto notable sobre el rendimiento. Así, arquitecturas como 3-1-1 o incluso 2-3 alcanzan frecuencias similares a arquitecturas más profundas como 1-1-1-1-1, sin necesidad de una segmentación extrema.

En cambio, en SHAKE256 el comportamiento es más sensible a la arquitectura concreta. La frecuencia máxima varía de forma más acusada entre arquitecturas, y no basta con separar θ y χ para obtener un buen resultado. De hecho, en este caso, la arquitectura que mejores resultados ofrece es precisamente la 1-1-1-1-1, donde todas las funciones están segmentadas y distribuidas en etapas independientes. Este comportamiento sugiere que, en SHAKE256, el retardo está más repartido entre varias funciones del núcleo KECCAK-f, y que una segmentación más profunda permite reducir el recorrido crítico global de manera más efectiva.

Esta diferencia puede deberse a que SHAKE256, al tener una capacidad mayor, implica una carga de datos más compleja en cada ronda. Como consecuencia, las funciones que en SHAKE128 tienen un impacto marginal (como ρ, π o ι) adquieren un mayor peso relativo en el retardo total, lo que hace que la distribución de todas las funciones dentro de la segmentación cobre una importancia crítica para maximizar el rendimiento.

Todo lo mencionado se observa claramente en la Figura 8, donde se representa la frecuencia máxima alcanzada para cada arquitectura, comparando SHAKE128 (en azul) y SHAKE256 (en rojo). En el caso de SHAKE128, se aprecia que la frecuencia se mantiene prácticamente constante para todas las variantes, con valores que oscilan levemente entre 210 y 213 MHz. En cambio, en SHAKE256 se observa una mayor dispersión en las frecuencias obtenidas, siendo la arquitectura 1-1-1-1-1 la que alcanza el mejor resultado. Este comportamiento sugiere que en SHAKE256 el retardo está más repartido entre varias funciones, y que una segmentación profunda permite mejorar el rendimiento de forma más significativa.

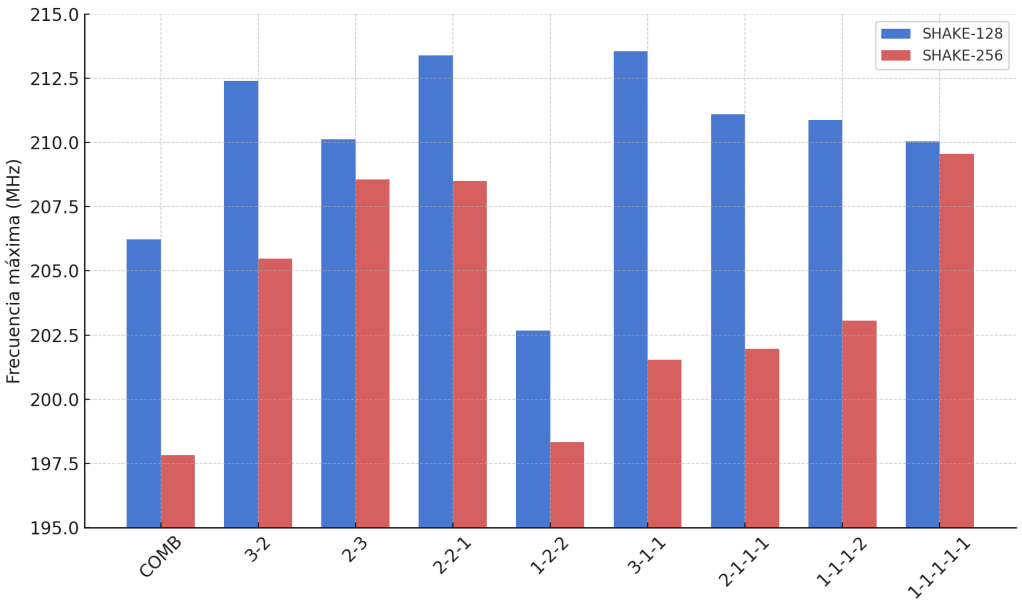

Figura 8: frecuencia para diferentes arquitecturas de SHAKE128 y SHAKE256.

4.3.3. Elección de arquitectura según los requisitos del sistema

A partir del análisis realizado, se puede afirmar que la mejor arquitectura depende en gran medida de lo que se necesite en cada sistema. Maximizar el rendimiento, minimizar el área, o alcanzar un equilibrio entre ambos.

- **Alta velocidad (máxima frecuencia):** Para funciones como SHAKE256 o SHA3-512, donde el retardo está más distribuido entre varias funciones del núcleo KECCAK, las arquitecturas completamente segmentadas (como 1-1-1-1-1) permiten alcanzar las frecuencias más altas. En SHAKE256, por ejemplo, esta arquitectura supera los 209 MHz, mientras que otras como 2-3 o 3-1-1 se quedan en torno a los 201–205 MHz. Por tanto, si el objetivo principal es alcanzar la máxima velocidad de operación, esta opción es la más adecuada, aunque con un mayor coste en recursos.

- **Eficiencia equilibrada (Mbps por slice):** En casos como SHA3-256 o SHAKE128, donde la frecuencia no varía significativamente entre arquitecturas, las configuraciones intermedias como 2-3 o 3-1-1 ofrecen una eficiencia muy competitiva. Por ejemplo, en SHAKE128, la arquitectura 3-1-1 alcanza 213.56 MHz con un uso de área no muy extenso, lo que la convierte en una de las más eficientes globalmente.

- **Área reducida o bajo consumo:** Cuando las restricciones de área o consumo son prioritarias, como en sistemas empotrados o FPGAs de gama baja, las arquitecturas combinacionales o poco segmentadas (como 2-3 o incluso 2-2-1) son especialmente útiles. Aunque su frecuencia es ligeramente inferior, permiten reducir significativamente el número de flip-flops y slices. En SHA3-256, por ejemplo, la arquitectura combinacional alcanza más de 185 MHz sin segmentación, lo cual puede ser suficiente en muchos contextos con necesidades moderadas.

Aunque las segmentaciones más profundas incrementan ligeramente la frecuencia máxima al acortar el retardo crítico, el throughput efectivo tiende a disminuir a medida que la segmentación crece. En FPGA esto se debe probablemente a que cada nueva etapa introduce registros y lógica de control adicional, alargando el número de ciclos necesarios para procesar un bloque completo. Además, al mapearse las funciones sobre los Configurable Logic Blocks (CLBs), la programabilidad interna (multiplexores, ruteo, ...) diluye la reducción teórica de retardo, de modo que la ganancia en frecuencia no compensa el sobrecoste de latencia y recursos, empeorando el rendimiento agregado.

Por lo tanto, si únicamente se busca eficiencia bruta, entendida como bits procesados por segundo y por slice, la arquitectura combinacional es la opción más eficiente en todas las funciones analizadas.

En la Tabla 9 se muestra un resumen con las arquitecturas más rápidas y más equilibradas para cada caso.

Tabla 9: Resumen de arquitecturas óptimas por función.

Función	**Arquitectura más rápida**	**Arquitectura más equilibrada**	**Arquitectura más eficiente**	**Observaciones**
SHA3-256	3-1-1	2-3	Combinacional	La frecuencia apenas varía entre arquitecturas. Separar θ y χ es suficiente.
SHA3-512	2-1-1-1	3-2	Combinacional	Más sensible a la segmentación. Requiere un equilibrio en la distribución del retardo.
SHAKE128	3-1-1 o 2-2-1	3-1-1 o 2-2-1	Combinacional	Muy estable en frecuencia. Variaciones mínimas entre arquitecturas.
SHAKE256	1-1-1-1-1	2-3	Combinacional	Se beneficia claramente de una segmentación profunda. Frecuencia muy dependiente de la arquitectura.

De esta forma, los datos obtenidos para los diferentes diseños permiten seleccionar de forma informada la arquitectura más apropiada en función de las características del objetivo.

5. Conclusiones

Este trabajo presenta una propuesta sólida para la aceleración de SHA3/SHAKE en hardware e integración en esquemas post-cuánticos. En primer lugar, se ha desarrollado una familia modular de núcleos VHDL que implementa cuatro de las variantes de FIPS 202. Todas comparten la misma jerarquía de ficheros y permiten ajustar el grado de segmentación de manera sencilla, lo que facilita su reutilización y mantenimiento.

En segundo lugar, se ha definido y automatizado una metodología de evaluación reproducible. Los diseños se sintetizan en Vivado 2023.2 con un flujo que ajusta el timing hasta alcanzar WNS ≈ 0 ns, de modo que las comparaciones de frecuencia y área no dependan de opciones de compilación arbitrarias. La tabla de métricas resultantes (LUTs, FFs, slices, $f_{\max}$, throughput y eficiencia) sirve como plantilla para tener una idea rápida de los datos.

A continuación se ha realizado un análisis cuantitativo del impacto de la segmentación. El estudio de nueve configuraciones indica que el compromiso área/frecuencia varía según cuál de los cuatro casos se considere. En algunos, aumentar de forma notable la segmentación (más de cuatro etapas) eleva significativamente la frecuencia máxima. En otros, se alcanzan frecuencias altas con segmentaciones reducidas (menos de 3 etapas), de modo que añadir registros adicionales sólo penaliza el área sin beneficio claro de rendimiento.

- **SHA3-256**. – Máxima frecuencia: 3–1–1 (198,22 MHz).
  – Mejor eficiencia: combinacional, 4,82 Gb/s·Slice^{-1}.
- **SHA3-512**. – Máxima frecuencia: 2–1–1–1 (197,4 MHz).
  – Mejor eficiencia: combinacional, 3,08 Gb/s·Slice^{-1}.
- **SHAKE128**. – Máxima frecuencia: 3–1–1 (213,6 MHz).
  – Mejor eficiencia: combinacional, 7,84 Gb/s·Slice^{-1}.
- **SHAKE256**. – Máxima frecuencia: 1–1–1–1–1 (209,6 MHz).
  – Mejor eficiencia: combinacional, 5,66 Gb/s·Slice^{-1}.

Estos resultados permiten extraer una pauta clara. Si el parámetro crítico es la frecuencia, conviene aislar las funciones θ y χ (3–1–1 para SHA3-256/SHAKE128, 2–1–1–1 para SHA3-512 y segmentación completa para SHAKE256). Si, en cambio, lo prioritario es el throughput o la eficiencia, la arquitectura totalmente combinacional sigue siendo la más ventajosa en todos los casos. Las cifras de referencia sobre Artix-7 constituyen, por tanto, una base sólida para comparar futuros portados a ASIC o a FPGA de gama superior.

5.1. Limitaciones del trabajo

Aunque los resultados son coherentes y se repiten al replicarlos, hay dos factores a tener en cuenta antes de sacar conclusiones generales:

Dependencia de la familia FPGA. Todo el estudio se ha realizado en dispositivos Xilinx Artix-7. Esto implica que las métricas reportadas (slices, LUTs, $f_{\max}$, eficiencia) están ligadas a la arquitectura de esta familia. En otras plataformas, la situación cambia:

- **FPGAs de gama alta** (Virtex UltraScale+, Stratix 10): ofrecen redes de interconexión más rápidas y LUTs combinados con registros, lo que suele elevar $f_{\max}$ pero también incrementa el consumo estático y modifica la relación throughput/área.
- **FPGAs de bajo consumo** (Lattice iCE/Certus): emplean LUTs de 4 entradas y matrices de routing simplificadas [6]. La misma lógica puede ocupar más celdas y alcanzar menor frecuencia.
- **Otros fabricantes** (Intel, Gowin): difieren en bloques especializados (DSP, BRAM) y en la granularidad de los recursos, alterando la forma de mapear la función χ o la memoria del estado KECCAK.

En consecuencia, las comparaciones directas con diseños de la literatura deben hacerse siempre mediante la normalización que propone este trabajo ($\mathrm{Gb/s} \cdot \mathrm{Slice}^{-1}$) o, preferiblemente, repitiendo la síntesis en la familia objetivo.

Ausencia de prototipo ASIC. Un ASIC (Application-Specific Integrated Circuit) es un circuito integrado diseñado para realizar una tarea específica de forma eficiente, cuya implementación final depende del nodo de fabricación seleccionado. El diseño al no haberse sometido al flujo físico completo (place & route) en un nodo de fabricación (por ejemplo, GF 28 nm o TSMC 22 nm), se introducen dos incertidumbres:

1. **Área y consumo reales**. Las LUTs y los multiplexores de configuración desaparecen en un ASIC, de modo que el núcleo ocuparía pocas décimas de mm^2 y su consumo estático caería.
2. **Frecuencia máxima**. Al llevar el diseño a un chip a medida, las interconexiones son mucho más cortas y cada puerta queda optimizada para su carga, de modo que la velocidad de reloj suele subir. Aun así, hasta hacer la colocación y el enrutado completos no se pueden descartar nuevos límites propios de los chips reales.

5.2. Líneas de investigación futura

El próximo paso más importante es llevar el núcleo a un ASIC real. Ejecutar un place & route completo en un nodo moderno (28–22 nm) permitiría saber con exactitud cuánta superficie ocupa, cuánta energía consume y, sobre todo, a qué frecuencia puede llegar un chip hecho a medida. Esto serviría también para comprobar si las segmentaciones más eficientes, más rápidas y más equilibradas elegidas para FPGA y para cada caso, lo siguen siendo cuando las interconexiones son mucho más cortas y los retardos se distribuyen de forma distinta.

En paralelo conviene evaluar la portabilidad del diseño a otras familias FPGA. Repetir la síntesis automática en dispositivos de gama alta (Virtex UltraScale+, Stratix 10) mostraría hasta qué punto una red de interconexión más rápida puede elevar la frecuencia máxima, mientras que probar el núcleo en soluciones de bajo consumo como Lattice CertusPro-NX ayudaría a cuantificar el compromiso entre velocidad y eficiencia energética en entornos con presupuesto de potencia muy reducido.

Otro frente interesante de ampliación es explorar cómo estas arquitecturas pueden integrarse en los nuevos estándares de criptografía poscuántica (PQC) del NIST. En particular, resulta natural estudiar su aplicación a esquemas ya estandarizados como ML-KEM (Kyber), donde SHA3/SHAKE desempeñan un papel central tanto en la generación de claves como en los procesos de encapsulado y desencapsulado. De este modo, se podría analizar en qué medida las mejoras obtenidas en cada caso (SHA3/SHAKE) se trasladan a los estándares completos.

Referencias

[1] Daniel J. Bernstein, Johannes Buchmann y Erik Dahmen. «Introduction to post-quantum cryptography». En: *Post-Quantum Cryptography*. Springer, 2009, págs. 1-14.

[2] Guido Bertoni et al. *Cryptographic Sponge Functions*. http://sponge.noekeon.org/CSF-0.1.pdf. 2011.

[3] Lov K Grover. «A fast quantum mechanical algorithm for database search». En: *Proceedings of the 28th annual ACM symposium on Theory of computing*. ACM. 1996, págs. 212-219.

[4] Yiming Huang et al. «A Pure Hardware Implementation of CRYSTALS-KYBER PQC Algorithm through Resource Reuse». En: *IEICE Electronics Express* 17.20200234 (2020), págs. 1-6. DOI: 10.1587/elex.17.20200234.

[5] Andreas Hülsing et al. *SPHINCS+ Submission to the NIST Post-Quantum Project*. https://sphincs.org. 2023.

[6] Lattice Semiconductor Corporation. *iCE40 LP/HX Family Data Sheet*. Inf. téc. DS1040. Rev. 1.12, https://www.latticesemi.com/Products/FPGAandCPLD/iCE40. Lattice Semiconductor, 2017.

[7] Harris E. Michail, Lenos Ioannou y Artemios G. Voyiatzis. «High Performance Pipelined FPGA Implementation of the SHA-3 Hash Algorithm». En: *Proc. 4th Mediterranean Conference on Embedded Computing (MECO 2015)*. 2015, págs. 67-74. URL: https://www.artemiosv.info/hosted/meco2015.pdf.

[8] National Institute of Standards and Technology (NIST). *Secure Hash Standard (SHS)*. https://doi.org/10.6028/NIST.FIPS.180-4. 2015.

[9] NIST. *Post-Quantum Cryptography Standardization*. https://csrc.nist.gov/Projects/post-quantum-cryptography. 2024.

[10] *Online Hash Calculator*. Accedido: 27-mayo-2025. 2024. URL: https://emn178.github.io/online-tools/.

[11] Nikolaos Sklavos et al. «A Pipelined Hardware Architecture Supporting All the Four SHA-3 Modes of Operation». En: *Proceedings of the 2015 4th Mediterranean Conference on Embedded Computing (MECO)*. 2015, págs. 214-217. DOI: 10.1109/MECO.2015.7181931.

[12] National Institute of Standards y Technology. *Digital Signature Standard for Module-Lattice-Based Signatures*. Federal Information Processing Standards Publication FIPS 204. Dilithium (ML-DSA) Standard. Gaithersburg, MD, USA: U.S. Department of Commerce, 2024. URL: https://doi.org/10.6028/NIST.FIPS.204.

[13] National Institute of Standards y Technology. *FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Available at https://doi.org/10.6028/NIST.FIPS.202. 2015.

[14] National Institute of Standards y Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Federal Information Processing Standards Publication FIPS 203. Kyber (ML-KEM) Standard. Gaithersburg, MD, USA: U.S. Department of Commerce, 2024. URL: https://doi.org/10.6028/NIST.FIPS.203.