Diseño e implementación de aceleradores en hardware para funciones de seguridad y cifrado, utilizando bibliotecas Vitis y plataformas SoC-FPGA, y evaluando su rendimiento computacional.

Rodriguez Sevilla, Paolo Estefano

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Telecomunicaciones

Trabajo de integración curricular, previo a la obtención del título de Ingeniero en Telecomunicaciones

Ing. Navas Viera Byron Roberto, Ph.D.

09 de marzo del 2026

INFORME DE ANÁLISIS
magister

MIC_Rodriguez_Paolo_EscritoFinal_v04

7% Textos sospechosos

- < 1% Similitudes
  - < 1 % similitudes entre comillas
  - < 1 % entre las fuentes mencionadas
- 2% Idiomas no reconocidos (ignorado)
- 7% Textos potencialmente generados por la IA

Nombre del documento: MIC_Rodriguez_Paolo_EscritoFinal_v04.pdf
ID del documento: 7ebf5e8511639171cfdf2a3ad56c88956dcd2b99
Tamaño del documento original: 5,06 MB

Depositante: BYRON ROBERTO NAVAS VIERA
Fecha de depósito: 18/2/2026
Tipo de carga: interface
fecha de fin de análisis: 18/2/2026

Número de palabras: 22.111
Número de caracteres: 159.740

Ubicación de las similitudes en el documento:

Fuentes de similitudes

Fuente principal detectada

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	Documento de otro usuario #aa7693 Viene de de otro grupo	< 1%		Palabras idénticas: < 1% (21 palabras)

Fuentes con similitudes fortuitas

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	hdl.handle.net | Implementación de inteligencia artificial para el reconocimiento ... https://hdl.handle.net/10902/38655	< 1%		Palabras idénticas: < 1% (26 palabras)
2	deee.espe.edu.ec | 2020-PIC-015-INV (CRaA: Computación Reconfigurable y Auto-... https://deee.espe.edu.ec/2020-pic-015-inv/	< 1%		Palabras idénticas: < 1% (11 palabras)

Fuentes mencionadas (sin similitudes detectadas)

Estas fuentes han sido citadas en el documento sin encontrar similitudes.

1 https://docs.amd.com/r/2022.2
2 https://www.amd.com/de/products/adaptive-socs-and-fpgas/soc/zynq
3 https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html
4 https://docs.amd.com/r/en
5 https://www.amd.com/en/products/software/adaptive-socs-and

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
BYRON ROBERTO NAVAS VIERA

Ing. Navas Viera Byron Roberto, Ph.D.

C.C.: 1709079287

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Telecomunicaciones

Certificación

Certifico que el trabajo de integración curricular: "**Diseño e implementación de aceleradores en hardware para funciones de seguridad y cifrado, utilizando bibliotecas Vitis y plataformas SoC-FPGA, y evaluando su rendimiento computacional.**" fue realizado por el señor **Rodriguez Sevilla, Paolo Estefano,** el mismo que cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, además fue revisado y analizada en su totalidad por la herramienta de prevención y/o verificación de similitud de contenidos; razón por la cual me permito acreditar y autorizar para que se lo sustente públicamente.

Sangolquí, 09 marzo del 2026

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
BYRON ROBERTO NAVAS VIERA

Ing. Navas Viera Byron Roberto, Ph.D.

C.C.: 1709079287

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Telecomunicaciones

Responsabilidad de Autoría

Yo, **Rodriguez Sevilla, Paolo Estefano**, con cédula de ciudadanía n°1727734640, declaro que el contenido, ideas y criterios del trabajo de integración curricular: **Diseño e implementación de aceleradores en hardware para funciones de seguridad y cifrado, utilizando bibliotecas Vitis y plataformas SoC-FPGA, y evaluando su rendimiento computacional** es de mi autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos, y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Sangolquí, 09 marzo del 2026

Rodriguez Sevilla, Paolo Estefano

C.C.: 1727734640

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Telecomunicaciones

Autorización de Publicación

Yo **Rodriguez Sevilla, Paolo Estefano**, con cédula de ciudadanía n°1727734640,autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de integración curricular: **Título: Diseño e implementación de aceleradores en hardware para funciones de seguridad y cifrado, utilizando bibliotecas Vitis y plataformas SoC-FPGA, y evaluando su rendimiento computacional.** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Sangolquí, 09 de marzo del 2026

Rodriguez Sevilla, Paolo Estefano

C.C.: 1727734640

Índice de contenidos

Índice de tablas

Índice de figuras

Resumen

Las redes de telecomunicaciones de nueva generación, como 5G e IoT, requieren nuevas medidas de seguridad que impliquen el manejo y control de la información, equilibrando la seguridad y el uso eficiente de los recursos computacionales. El uso de la misma arquitectura de computación (CPU) para la ejecución de procesos de algoritmos criptográficos, como el cifrado simétrico AES-128 y la función hash SHA-256, provoca el sobrecalentamiento de la CPU, lo que limita el uso de la red para otras aplicaciones. Se describe el diseño, implementación y verificación de aceleradores de hardware en SoC-FPGA, utilizando bibliotecas de AMD Vitis y HLS. La arquitectura diseñada se encarga de migrar de forma automática el trabajo computacional que se realiza en el Componente de Procesamiento hacia la Lógica Programable y el trabajo de control de gestión de datos en el sistema operativo Linux embebido mediante el uso de AXI de alto rendimiento y controladores de flujo. El diseño también contempla la verificación de la funcionalidad usando vectores de prueba de NIST y la comparación de la latencia y el uso de recursos lógicos. De acuerdo con los resultados experimentales, el sistema presenta un rendimiento de 59.5 MB/s para el algoritmo de cifrado AES-128 y 12.6 MB/s utilizando el algoritmo SHA-256. Aunque la frecuencia de operación de la FPGA es menor que la de un microprocesador de escritorio, la arquitectura dedicada libera la carga de trabajo del microprocesador y brinda un comportamiento estable y determinístico al procesar grandes cantidades de datos. Por lo anterior, la diversidad de arquitecturas de soporte es factible para la implementación de nodos de edge computing donde la disponibilidad del sistema y la eficiencia en el uso de la energía es lo más importante.

Palabras clave: SoC-FPGA, aceleración en hardware, criptografía, Vitis Libraries, síntesis de alto nivel High-Level Synthesis.

Abstract

Establishing new security techniques for protection mechanisms on telecommunications systems such as 5G and IoT must be strategically planned to determine how information can be best protected while optimizing the value of available computing resources. Typical encryption and hashing blanket implementations on processors such as AES-128 and SHA-256, create a saturation of the central processing unit and can cause network application bandwidth to be lost. The present degree assignment explains the design, implementation, and evaluation of the hardware accelerators on the Zynq-7000 SoC-FPGA, using the AMD Vitis and High Level Synthesis (HLS) bibliographic method. The created architecture transfers processing-intensive activities from the Processing System (PS) to the Programmable Logic (PL) and performs data transfers using high throughput AXI interfaces under the control of an Embedded Linux Operating System. The method addresses the functional verification of the design using NIST test vectors and latency, and logic (LUT and DSP) resources) utilization comparative analysis. It has been shown that the designed system achieves a throughput of 59.5 MB/s and 12.6 MB/s for AES-128 and SHA-256, respectively. Although the operating frequency of the FPGA is lower than that of a desktop CPU, the developed architecture can relieve the system's main processor and provide steady and predictable performance when handling large blocks of data. The design is suitable for edge computing nodes, as it meets design flexibility and power efficiency constraints.

Keywords: SoC-FPGA, Hardware acceleration, Cryptography, Vitis Libraries, High-Level Synthesis.

Capítulo I : Introducción

Antecedentes

En las redes modernas de telecomunicaciones, especialmente en las críticas como 5G, IoT, comunicaciones satelitales y sistemas de defensa, la protección de los datos y la codificación del canal son elementos esenciales. Los procesadores convencionales enfrentan el reto de que los algoritmos de corrección de errores y criptográficos son cada vez más complejos. La ejecución en software puro tiene restricciones con respecto a la eficiencia energética y la latencia.

En este escenario, investigaciones recientes como AHA: y proyectos anteriores como CRaA (Computación Reconfigurable y Auto-Adaptiva) [1], realizados en la ESPE, han establecido fundamentos para el empleo de hardware reconfigurable. La viabilidad de utilizar herramientas de síntesis de alto nivel (HLS) para optimizar las tareas intensivas en FPGA ha sido demostrada por Design and Evaluation of Compute-Intensive Hardware Accelerators for AMD-Xilinx Zynq SoCs Using HLS IP Flow [2]. Estas herramientas han conseguido disminuir considerablemente el consumo energético y la latencia. Sin embargo, en la carrera de Telecomunicaciones todavía no se han llevado a cabo proyectos de titulación que examinen el empleo de Vitis Libraries para acelerar algoritmos de cifrado y seguridad en SoC-FPGA. Por lo tanto, esta investigación se propone como una nueva línea de indagación aplicada en el hardware embebido, con un enfoque en aplicaciones críticas, de acuerdo con la nota conceptual sugerida por el ingeniero Navas [3].

Estado del Arte

El interés por la aceleración de algoritmos criptográficos en hardware reconfigurable ha aumentado en los últimos años debido a la necesidad de asegurar ciberseguridad en telecomunicaciones con latencias reducidas y eficiencia energética elevada [3]. Los SoC-FPGA, a diferencia de las implementaciones en software tradicional, que tienen restricciones de

cómputo en los procesadores, posibilitan la optimización y el paralelismo de recursos y así se obtienen avances importantes en el desempeño de funciones críticas para la seguridad.

Se ha confirmado en diferentes estudios que los algoritmos AES, SHA-256 y RSA tienen un throughput más alto y un consumo menor cuando se ejecutan en FPGA, en vez de hacerlo en CPU convencionales [4]. Esta situación ha promovido que se aplique en bancos digitales, redes móviles y el Internet de las Cosas (IoT) sistemas críticos; en estos escenarios, la fiabilidad y la protección de datos sensibles son condiciones necesarias.

Con la necesidad de alta disponibilidad y baja latencia de tecnologías de red emergentes, como 5G y 6G, ha crecido la necesidad de sistemas criptográficos que puedan trabajar en tiempo real. En este escenario, la integración de aceleradores criptográficos en sistemas embebidos se presenta como una opción viable para este tipo de problemas [5].

Por otra parte, la necesidad de facilitar la creación de aceleradores ha llevado a mejoras en las herramientas de síntesis de alto nivel (HLS) de modo que en el usuario no haya necesidad de programar en HDL. En este sentido, las bibliotecas Vitis de AMD-Xilinx ofrecen bloques de compresión, codificación y seguridad que facilitan la inclusión de funcionalidades de cifrado y verificación de datos en sistemas SoC-FPGA [2].

Pese a estos avances, persiste la necesidad de llevar a cabo investigaciones comparativas que documenten las discrepancias de desempeño entre las implementaciones en hardware y software, considerando indicadores como la utilización de recursos, la latencia y el rendimiento. Esta disertación se ubica dentro de esta tendencia, dado que desarrolla e implementa aceleradores de hardware destinados a funciones de cifrado y seguridad utilizando Vitis Libraries, los integra en plataformas SoC-FPGA y analiza su desempeño computacional en contextos vinculados a las comunicaciones.

Planteamiento del problema

Los sistemas de telecomunicaciones actuales requieren de sistemas criptográficos complejos que aseguran la privacidad, la autenticidad y la integridad de la información. Esta

necesidad se hace más relevante en entornos como las aplicaciones del Internet de las Cosas (IoT), sistemas de misión crítica, plataformas financieras y redes 5G. En todos los ejemplos citados, el nivel de confianza en el procesamiento, la cantidad de energía consumida, y la latencia son una condición necesaria para el buen funcionamiento del sistema.

No obstante, la aplicación tradicional mediante software presenta un obstáculo de rendimiento que no se puede superar. Los procesadores de propósito general presentan cuellos de botella inmediatos por su gestión de recursos compartidos y su arquitectura secuencial. A medida que aumenta la cantidad de datos, esta limitante por estas razones, aumenta la latencia y disminuye el rendimiento de la energía. Por lo tanto, las CPUs estándar no logran mantener el procesamiento en tiempo real que necesitan los estándares de hoy.

Las plataformas SoC-FPGA son una opción para el desarrollo de aplicaciones de computación que requieren de un especial hardware, pero su uso para la aceleración de funciones criptográficas no ha sido empleado en los contextos académicos locales, ni se encuentra documentado en gran medida. En el diseño de aceleradores criptográficos se nota la diferencia en el uso de bibliotecas avanzadas, como por ejemplo, las AMD Vitis Libraries. También hay una diferencia en el cálculo del impacto cuantitativo que estas implementaciones tienen en comparación con las estandarizadas en el software.

Esta limitación técnica reclama la construcción de aceleradores de hardware sobre plataformas SoC-FPGA. En este sentido, nuestra propuesta utiliza el paralelismo y la eficiencia energética de los dispositivos para optimizar las funciones de cifrado. No obstante, la construcción por sí sola no asegura el éxito. Para esto, deberemos establecer métricas comparativas estrictas. Así, validaremos las ventajas reales del hardware en comparación con las soluciones de software de telecomunicaciones.

Justificación e importancia

La razón por la cual se llevó a cabo este trabajo de titulación es el creciente requerimiento de contar con sistemas de telecomunicaciones que sean seguros, confiables y

con baja latencia. En un escenario en el que las comunicaciones corporativas, los servicios digitales y las transacciones financieras emiten cada segundo millón de datos sensibles, contar con mecanismos de cifrados efectivos ya no es solo una cuestión técnica; también se ha vuelto una necesidad a nivel estratégico, económico y social. Incorporar aceleradores en un SoC-FPGA resulta ser la ruta más efectiva para mejorar la seguridad informática cuando la velocidad de respuesta es vital.

Lo destacable de este trabajo es su transferencia directa al mundo real, abarcando desde plataformas digitales hasta redes de telecomunicaciones críticas. Se demuestra algo clave: no hace falta invertir en equipos impagables. Un sistema embebido bien optimizado basta para asegurar la información al vuelo. El hardware eficiente hace posible desarrollar herramientas de seguridad más accesibles y escalables para cualquier operador de red. Por lo cual, las entidades públicas y privadas hoy en día tienen la mejor tecnología para combatir cualquier ciber amenaza por más compleja que sea.

Más allá del aula, la propuesta busca independencia tecnológica. Crear soluciones propias en FPGA permite prescindir de hardware comercial costoso y fomenta la ingeniería local, cumpliendo con los ODS (4, 9, 16) y los lineamientos de Navas [3]. El valor real del trabajo es tangible: probar y estresar aceleradores de cifrado hasta validar su funcionamiento. No se trata de cumplir un requisito académico, sino de entregar módulos de seguridad listos para implementarse donde realmente hacen falta.

Objetivos

Objetivo general:

- Diseñar e implementar aceleradores en hardware para funciones de seguridad y cifrado, utilizando bibliotecas Vitis y plataformas SoC-FPGA, y evaluando su rendimiento computacional.

Objetivos específicos:

- Investigar y seleccionar al menos dos algoritmos criptográficos (e.g., AES-128, SHA-256, RSA, ECC).
- Verificar funcionamiento mediante modelos en MATLAB o scripts Python.
- Implementar en C/C++ usando Vitis Libraries (e.g., xf::security, xf::compression).
- Integrar en plataforma Zynq SoC con sistema operativo definido (e.g. bare metal o Linux).
- Validar funcionamiento y realizar pruebas comparativas con versión en C sobre ARM o Intel.

Resumen de contenidos

Este estudio de titulación se organiza en seis capítulos. Los dos primeros se ocupan del marco conceptual y de la formulación del sistema. En ellos se describe la sistematización del proceso de diseño en la plataforma SoC-FPGA. Los aceleradores criptográficos se analizan en el capítulo tercero y la valoración experimental de los resultados obtenidos en el capítulo cuatro. Esta organización permite diseccionar la solución técnica a fondo, cubriendo sistemáticamente cada etapa del diseño.

El **Capítulo I** establece el marco de la investigación. En él se analizan los antecedentes para evidenciar un cuello de botella real, como lo es la necesidad de una solución que implementen una seguridad robusta en las telecomunicaciones, y que a la vez, sea de bajo consumo y con un rápido tiempo de respuesta. También, por la justificación académica y los objetivos, se traza el camino de los resultados esperados del proyecto.

El **Capítulo II** esboza los fundamentos de la criptografía aplicada y los campos de aceleración de hardware aplicados a SoC-FPGAs. El enfoque del estudio es la arquitectura Zynq-7000 y su comunicación PS-PL a través de buses AXI. El flujo de análisis del diseño se

explica utilizando Síntesis de Alto Nivel (HLS). También se explica cómo las Bibliotecas Vitis ayudan a crear aceleradores diseñados para arquitecturas heterogéneas.

En el **Capítulo III,** se describe la metodología junto con el diseño técnico para la implementación del acelerador criptográfico para los algoritmos AES-128 y SHA-256. El primer paso es describir la arquitectura general del sistema y el modelo de referencia que se ejecutará en software para evaluar el correcto funcionamiento y el rendimiento final. Luego se describe la generación de los bloques de IP utilizando Vitis HLS, su empaquetado con interfaces AXI y su integración en la lógica programable del SoC y el WSL del sistema de procesamiento. Finalmente, se describe la verificación funcional con vectores de prueba estándar del correcto funcionamiento de los aceleradores.

El **Capítulo IV** presenta las pruebas de desempeño, estableciendo la diferencia entre el procesamiento tradicional por software versus aceleración directa por hardware. A lo largo de la tabulación de los escenarios de prueba en la tarjeta de desarrollo con un control adecuado, se puede considerar una variedad de condiciones de datos y ejecución. Se mide previamente un conjunto de métricas críticas, que incluyen la latencia, el throughput, el flujo de utilización de recursos de los recursos lógicos luts, brams y dpx, y los resultados con el supuesto de que el intercambio de datos se aborda entre el PS y el PL.

El **Capítulo V** presenta el análisis de los resultados. Estas dos implementaciones hardware y software se comparan de manera estructurada mediante tablas y gráficos. Se consideran varios aspectos de diferencias de rendimiento, la eficiencia de elección del sistema, posibles cuellos de botella y la influencia de SoC-FPGA arquitectura en el comportamiento global del sistema.

Finalmente, en el **Capítulo VI,** se entregan las conclusiones generales del presente trabajo, mencionando las contribuciones técnico-científicas logradas con respecto al diseño e integración de aceleradores criptográficos a plataformas reconfigurables. Adicionalmente, se presentan las futuras líneas de investigación que involucran la expansión del sistema, la

utilización de otros algoritmos de seguridad y la integración de los aceleradores con escenarios reales de las telecomunicaciones y la computación en el borde.

Capítulo II: Marco Teórico

Estado del arte

En las aplicaciones de Internet de las Cosas (IoT) y Blockchain, donde la integridad de los datos bajo restricciones energéticas es crítica, Santos et al. (2024) presentan una arquitectura de hardware destinada a la optimización del algoritmo SHA-256, con el propósito de llevar a cabo verificaciones criptográficas de bajo consumo [4]. El diseño utiliza un enfoque multinúcleo con paralelización a nivel de bloques, lo que permite mejorar el rendimiento sin aumentar proporcionalmente los recursos lógicos utilizados. Los autores proporcionan métricas exhaustivas de frecuencia, utilización de LUT y rendimiento, alcanzando hasta 1.4 Gb/s usando 16 núcleos en paralelo, junto con una gran reducción del consumo dinámico respecto a soluciones anteriores. Los resultados muestran que el uso de funciones hash criptográficas en FPGA es una solución realista para asegurar la integridad de los datos en tiempo real en sistemas embebidos y sistemas distribuidos con restricciones de energía [4].

A diferencia de los algoritmos hash, los esquemas criptográficos de clave pública son más costosos computacionalmente, lo cual representa un problema en aplicaciones IoT con restricciones de recursos. En ese sentido, Al-Khaleel et al. en 2024 realizan una revisión del diseño de procesadores de criptografía de curva elíptica optimizados para aplicaciones IoT, implementados en FPGA [6]. La investigación utiliza curvas de Edwards y aprovecha los recursos embebidos del dispositivo, tales como bloques de procesamiento digital de señales (DSP) y memorias internas, con el fin de optimizar las operaciones críticas de multiplicación escalar. Se presentan arquitecturas tanto en configuración serial como paralela, lo que permite examinar los compromisos entre área, velocidad y consumo energético. Además, se introducen técnicas de multiplicación en el dominio de la frecuencia con el propósito de reducir la presión sobre la lógica general. Los ensayos han revelado progresos de hasta 2.5 veces en relación con las aplicaciones previas. Esta conclusión muestra que es posible aplicar la criptografía de

curva elíptica en FPGAs de forma efectiva, incluso con restricciones severas de recursos y potencia [6].

Reducir la latencia y el consumo de energía es fundamental para las aplicaciones de borde y los sistemas IoT con altas demandas de procesamiento, en el marco del cifrado simétrico. En 2025, Karakchi et al. presentan SPIME [5], una arquitectura de cifrado AES que es muy paralela y se basa en un método de procesamiento en memoria. El diseño cuenta con varios núcleos AES-128 distribuidos que funcionan simultáneamente, lo que reduce la dependencia de un procesador central y limita el movimiento de datos hacia la memoria externa. Las evaluaciones realizadas sobre plataformas FPGA indican que, en dispositivos de gama alta, la latencia es constante y equivale a once ciclos de reloj, el rendimiento sostenido supera los 25 Gb/s y la utilización de recursos se mantiene por debajo del 4 %. Estos hallazgos demuestran que las arquitecturas paralelas especializadas permiten escalar el cifrado simétrico sin poner en riesgo la eficiencia energética ni el rendimiento en contextos de computación perimetral [5].

Para diseñar de manera efectiva aceleradores criptográficos, más allá de las arquitecturas específicas, se requieren metodologías que reduzcan la complejidad del desarrollo sin poner en riesgo el rendimiento. Conforme a esta línea de pensamiento, Berrazueta-Mena y Navas proponen en 2025 el uso de la metodología AHA para diseñar e implementar aceleradores hardware que requieren un alto nivel de computación sobre SoC AMD-Xilinx Zynq por medio de flujos de síntesis con alto nivel [2]. La investigación muestra que el empleo sistemático de HLS posibilita la creación de aceleradores criptográficos con avances notables en términos de latencia y consumo energético, en comparación con soluciones únicamente basadas en software. La generación de IPs AXI integrables en arquitecturas PS-PL constituye el mecanismo de validación, lo que demuestra la factibilidad de este enfoque para sistemas embebidos orientados a aplicaciones de telecomunicaciones seguras y computación en el borde [2].

Al comparar los trabajos más actuales, que han sido publicados desde 2024, se puede observar que las arquitecturas tienden a ser altamente paralelas y especializadas, pero con estrategias diferentes dependiendo de la criptografía primitiva abordada. Para asegurar la integridad de los datos en tiempo real a través del procesamiento ininterrumpido por bloques, Santos et al. utilizan el paralelismo multinúcleo en el caso de SHA-256 [4]. Además, Karakchi et al. demuestran que el cifrado simétrico AES puede beneficiarse de arquitecturas distribuidas y de CPU-in-memory para reducir la latencia y el consumo de vacíos [5].

Al-Khaleel et al. demuestran que las operaciones de clave pública, en este caso ECC, necesitarían un mayor paralelismo y dependencia en DSP embebido recursos para proporcionar un rendimiento máximo en plataformas restringidas [6]. En conjunto, la literatura respalda que trasladar funciones criptográficas hacia hardware reconfigurable es esencial para cumplir exigencias estrictas de eficiencia energética y latencia, y que el diseño óptimo debe ajustarse al tipo de algoritmo, constituyendo la base directa del enfoque adoptado en esta tesis [2], [4], [5], [6].

Plataforma reconfigurable basada en Zynq-7000 All Programmable SoC

Las tecnologías FPGA y SoC abordan los retos del diseño digital desde frentes opuestos. Las tecnologías FPGA y SoC enfrentan los problemas de diseño digital de formas contrarias. Por un lado, la FPGA tiene partes que se pueden cambiar para hacer tareas más rápido, pero el SoC junta el procesamiento y la memoria para gastar menos energía. La arquitectura Zynq une lo bueno de los dos porque pone un Sistema de Procesamiento (PS) ARM junto a la Lógica Programable (PL). Para conectar estas dos partes, el chip usa cables AXI estándar de alta velocidad [7].

Este modelo le da las tareas de control y del sistema operativo al PS. Al mismo tiempo, el PL hace el trabajo pesado de cálculo usando muchos procesos al mismo tiempo. El resultado junta la capacidad de cambio del hardware con la facilidad del software. Los cables internos AXI aseguran que pase mucha información por ambos lados. Esta base sirve para los diseños

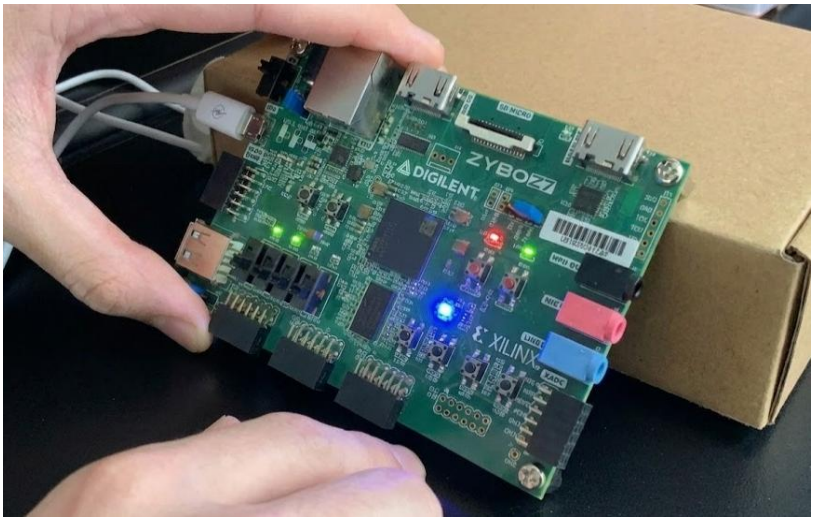
nuevos de telecomunicaciones y seguridad. Podemos mencionar la arquitectura que se puede optimizar del dispositivo Xilinx Zynq−7010: bloques de lógica (LUT), memorias internas (Block RAM) y bloques de procesamiento de señales (DSP) [7].

Arquitectura reconfigurable del dispositivo Xilinx Zynq-7010: bloques de lógica (LUT), memorias internas (Block RAM) y bloques DSP

Un chip configurable, que también es denominado FPGA, posee la gran habilidad de producir circuitos digitales que son personalizados. La arquitectura mencionada se basa en una estructura organizada con bloques lógicos configurables (CLB) junto a caminos de interconexión. Cada CLB está formado por flip-flops. Además, los segmentos que son tablas de búsqueda, también llamadas LUT (por sus siglas en inglés), lo forman. Estos elementos constituyen la lógica combinatoria. Son necesarios, además, los registros para el funcionamiento del sistema operativo. Para terminar, una matriz de enrutamiento crea un enlace entre todos estos recursos y cada una de las entradas y salidas del chip. Con este diseño, se evita cualquier modificación en la estructura física del aparato y se facilita la conversión de código en hardware paralelo real [7]. En la Figura 1 se presenta la tarjeta Zybo Z7 empleada como plataforma de implementación hardware-software dentro de este trabajo.

Figura 1

Tarjeta Zybo Z7 utilizada como plataforma de implementación hardware-software.



Nota. La tarjeta integra un SoC Zynq-7000 que permite ejecutar software en el PS y acelerar hardware en el PL.

La capacidad para el procesamiento interno de la FPGA depende de tres componentes que son esenciales. Las LUT (o Tablas de Búsqueda, por su acrónimo en inglés) cumplen funciones lógicas. Asimismo, operan como memorias adaptables. Las Block RAM guardan datos de gran densidad. Estos datos se encuentran cerca del área computacional. Asimismo, existen bloques para realizar operaciones aritméticas complejas con el propósito de liberar recursos lógicos. Esta combinación de elementos convierte a la FPGA en un motor de cálculo paralelo que se puede adaptar a cualquier actividad [7].

Concepto de SoC (System-on-Chip): integración CPU + FPGA

Un sistema en este tipo de chip (SoC) abarca todos los elementos funcionales de un computador en un único chip de silicio. Esta arquitectura fusiona un microprocesador, un módulo de memoria y algunas de las interfaces periféricas, y, por tanto, sustituye varios circuitos en una placa. Dicha construcción, debido a la miniaturización de todos los elementos de hardware, comunicación, y la mejora en el consumo energético, permite grandes saltos en el orden de magnitud. Sin embargo, los SoC tradicionales, como ASIC, no ofrecen flexibilidad. En mercados donde las actualizaciones deben realizarse de manera continua, los altos costos de ingeniería no recurrente (NRE) restringen su viabilidad [7].

La metodología System-on-Programmable-Chip convierte la arquitectura SoC en un aparato completamente reconfigurable. Las FPGAs, en este escenario, ofrecen la suficiente flexibilidad para renovar el sistema con un riesgo técnico bajo. La perspectiva se concreta en el sistema Zynq a través de la combinación de un procesador ARM Cortex-A9 (PS) y lógica programable (PL). Esta estructura híbrida permite una integración eficaz del diseño de hardware y software. El descubrimiento posibilita la puesta en marcha de aceleradores y optimizaciones concretas durante toda la duración del ciclo de vida del producto [7].

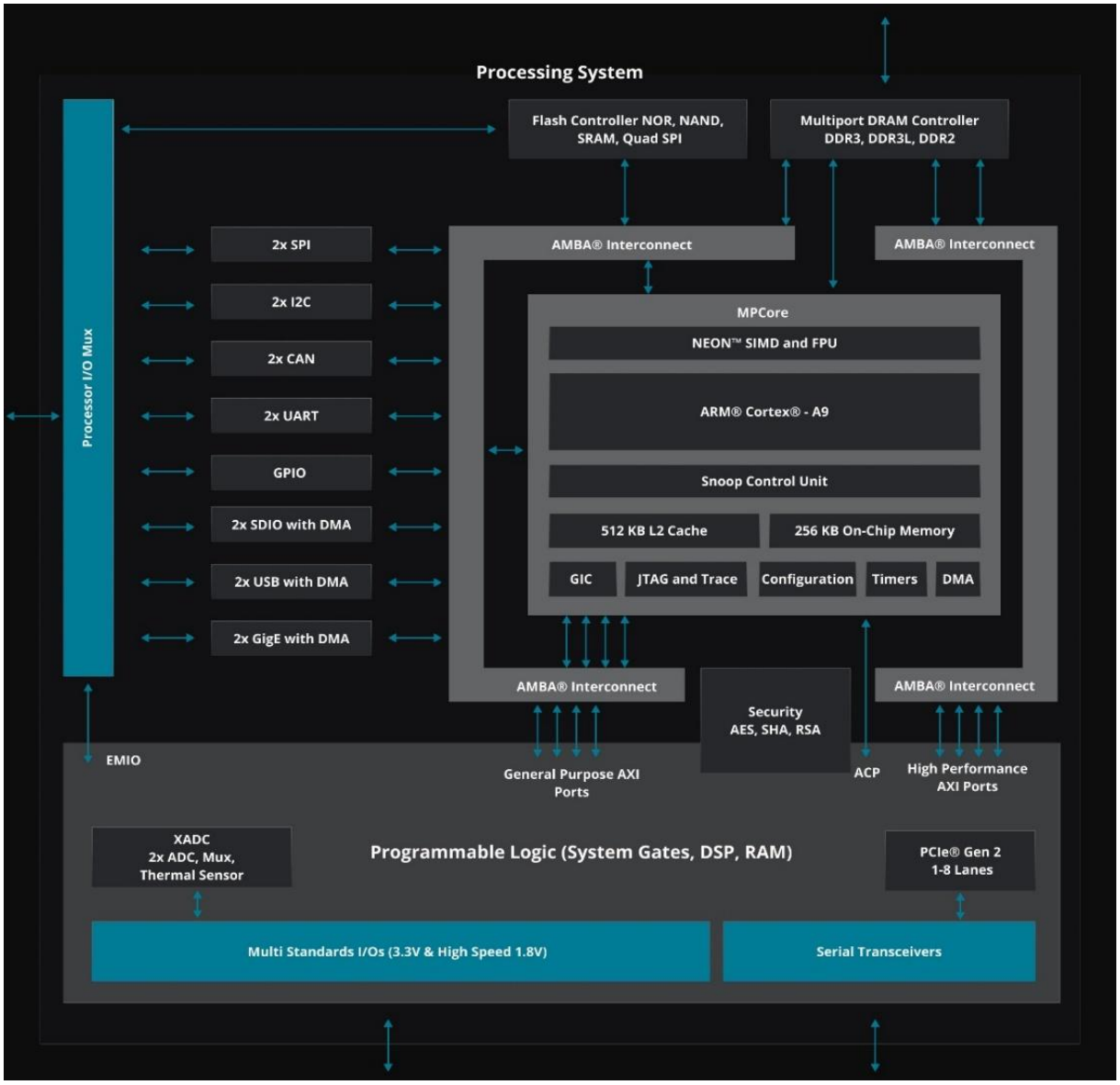
Arquitectura Zynq-7000: Processing System y Programmable Logic

La arquitectura Zynq-7000 combina a la vez una Lógica Programable (PL) y además un Sistema de Procesamiento (PS) en el mismo y único silicio. Dentro del dominio PS se encuentran los núcleos ARM Cortex-A9, las interfaces estándar y los controladores de memoria. A la vez, el dominio PL proporciona los bloques DSP además del tejido FPGA nativo que se requieren para la implementación de aceleradores específicos. Esta división arquitectónica mejora la forma. Por medio de esta mejora se distribuyen las cargas de trabajo. Finalmente, el procesador administra el sistema operativo y el flujo de control, mientras que el hardware reconfigurable asume las tareas críticas de paralelismo masivo y baja latencia [7].

El dispositivo se conecta a través de una red interna que utiliza el estándar AMBA AXI. Esta interfaz asegura vías de baja latencia hacia los núcleos y rendimiento elevado hacia la memoria. El sistema lleva a cabo la gestión de las transacciones y presenta puertos directos hacia la RAM DDR. Los aceleradores del PL distribuyen estos recursos de memoria con el procesador sin provocar conflictos de acceso, tal como se observa en la Figura 2. El resultado final es un SoC programable que combina la fuerza del hardware con la eficiencia del software [7].

Figura 2

Arquitectura del SoC Zynq-7000 y su interconexión PS-PL mediante buses AXI.



Nota. El PS basado en ARM Cortex-A9 se comunica con la lógica programable mediante interfaces AXI GP, HP y ACP, habilitando control, transferencia de datos de alto rendimiento y coherencia de caché para aceleradores hardware como AES y SHA-256. [8]

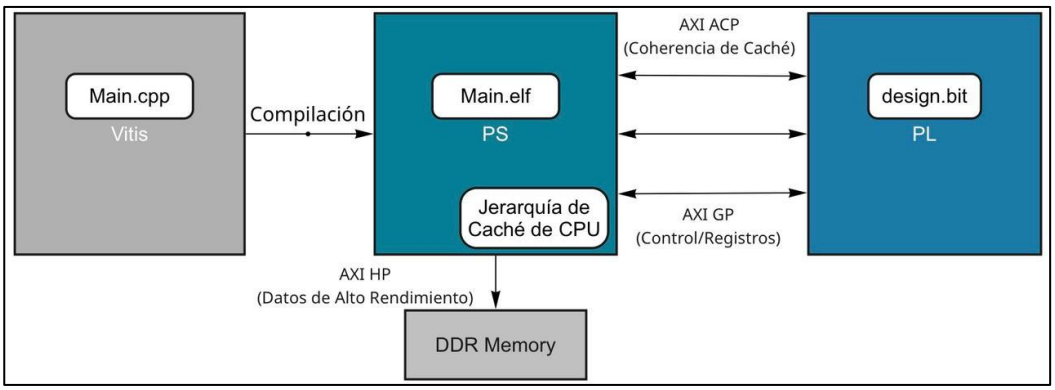
Comunicación PS–PL mediante buses AXI

La comunicación de datos entre la lógica programable y el sistema de procesamiento se basa en tres tipos de puertos AXI especializados. Los puertos de propósito general (AXI GP) se encargan de gestionar el control y los registros que tienen un ancho de banda bajo. Los puertos de alto rendimiento (AXI HP) permiten que se transfieran grandes volúmenes de datos a la

memoria DDR mediante búferes. Finalmente, el puerto de consistencia (AXI ACP) permite que el acelerador acceda a la jerarquía de caché del procesador de manera directa. Este esquema de integración software-hardware y la interacción entre PS y PL se ilustran en la Figura 3, donde se evidencia el flujo completo de ejecución en Vitis. Si se eligen adecuadamente estas interfaces, se mejora el diseño y se evita que surjan cuellos de botella en la transmisión de datos [7].

Figura 3

Flujo de integración software-hardware en Vitis con ejecución sobre PS y aceleración en PL.



Nota. El archivo ejecutable ".elf" usa el acelerador sintetizado ".bit" por medio de puertos AXI usando la memoria DDR externa para procesar datos con eficiencia.

Un conjunto de conmutadores gestiona el flujo de datos en el Sistema de Procesamiento, y esto en el plano interno. Este sistema envía solicitudes a las unidades centrales (CPU) y a las unidades de memoria secundarias (DDR). La topología no es completamente cruzada, lo que restringe el acceso a determinados subconjuntos de destinos. Esta modalidad de estructura ofrece baja latencia para núcleos, y gran eficiencia en el procesamiento en lotes. Este diseño proporciona la base física que se requiere para balancear el procesamiento en el plano de software y en el plano de hardware [7].

Flujo de diseño HLS (High-Level Synthesis) y herramientas de desarrollo

Bibliotecas AMD Vitis™: concepto general y niveles de abstracción

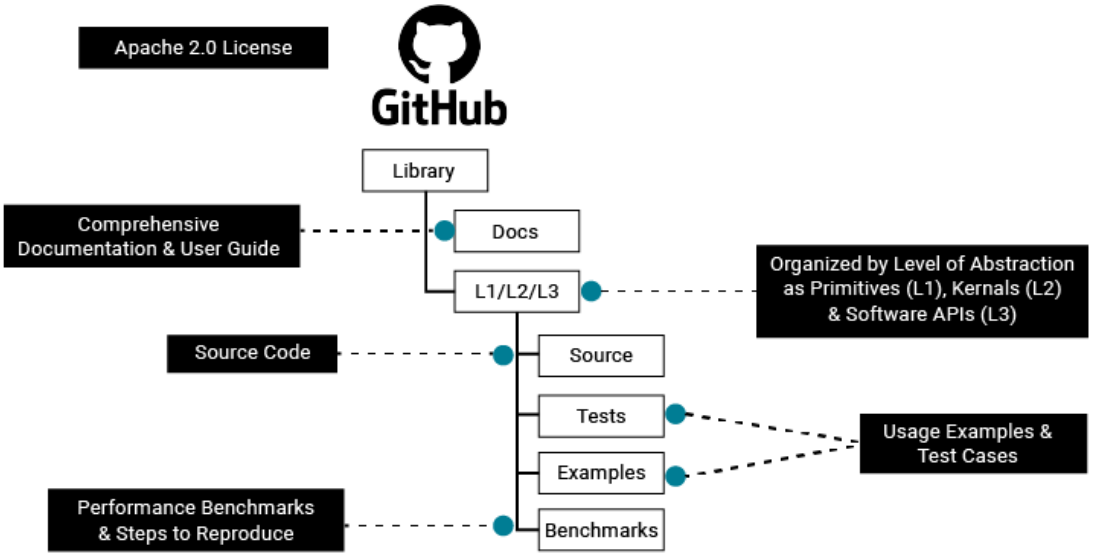
Las bibliotecas AMD Vitis™ ofrecen un ecosistema completo para la aceleración de cálculo para la transformación de funciones de alto nivel en lógica RTL optimizada para ser ejecutada en dispositivos FPGA y SoC-FPGA. Por el contrario a lo que hace una biblioteca de software estándar, estas bibliotecas encapsulan algoritmos y son preparadas para ser sintetizadas por medio de HLS, lo que permite mover la carga computacional intensiva desde el Sistema de Procesamiento (PS) hacia la Lógica Programable (PL). Como resultado se potencializan el masivo paralelismo, pipelining y la eficiencia en el consumo de energético que el hardware reconfigurable. Esto se alinea con la finalidad principal de la investigación en cómo implementar aceleradores de funciones criptográficas con control de la arquitectura [9], [10].

Además, la plataforma Vitis™ incluye un conjunto amplio de bibliotecas aceleradas y optimizadas que cubren dominios como procesamiento de señales (DSP), álgebra lineal, análisis de datos, visión artificial, compresión, bases de datos y seguridad, entre otros. Estas bibliotecas pueden invocarse como APIs en C/C++ y en ciertos casos en Python, ofreciendo aceleración “plug-and-play” con cambios mínimos en el código existente [11]. La Figura 5 ilustra este ecosistema global, mostrando la organización por dominios específicos, bibliotecas comunes y modalidades de despliegue desde el edge hasta la nube. En el contexto específico de este MIC, el dominio de interés corresponde precisamente a las Security Libraries, las cuales son empleadas para implementar los algoritmos criptográficos AES-128 y SHA-256 en hardware acelerado [11].

Las Vitis™ Libraries se organizan en tres niveles jerárquicos de abstracción según una perspectiva estructural y metodológica. Estos niveles son L1, L2 y L3. Esta jerarquía no solo define el grado de control sobre el hardware generado, además de la forma en que el repositorio oficial está estructurado, incluyendo documentación, código fuente, pruebas, ejemplos y benchmarks, tal como se muestra en la Figura 4.

Figura 4

Jerarquía de abstracción de las Vitis Libraries.



Nota. El diagrama muestra la organización del repositorio, incluyendo documentación técnica, código fuente, pruebas, ejemplos de uso y métricas de desempeño reproducibles. Asimismo, evidencia la división jerárquica en L1 (primitivas), L2 (kernels) y L3 (APIs de software), que determina el nivel de abstracción y control sobre la implementación hardware [12].

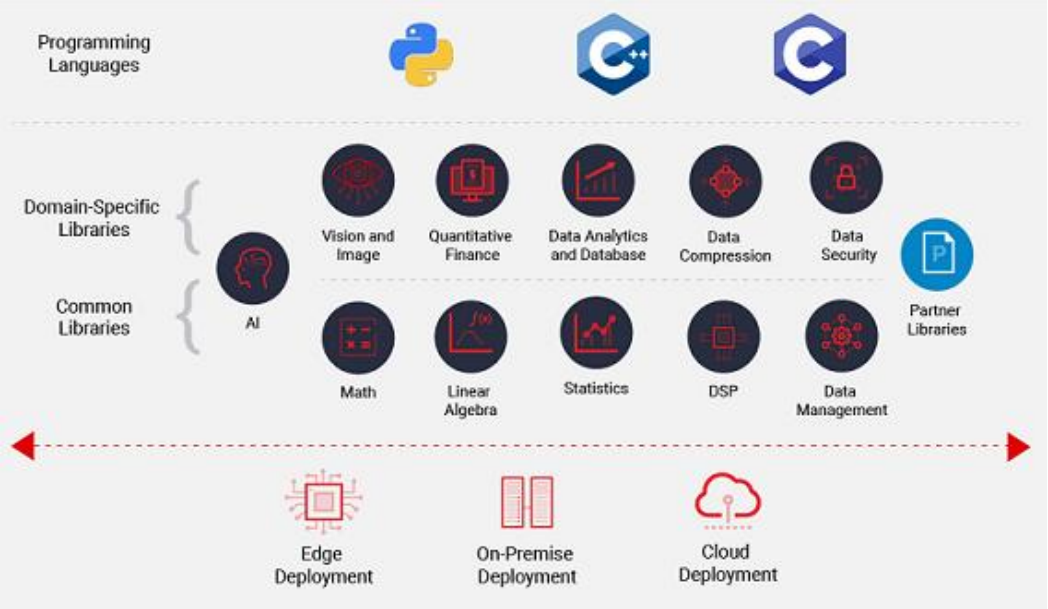
Como se observa en la Figura 4, el nivel L1 contiene las primitivas fundamentales optimizadas para hardware. Estas funciones trabajan de forma simple y ayudan a cambiar la estructura del chip con instrucciones de HLS. En este trabajo, el nivel L1 es muy importante porque permite controlar el tiempo, el trabajo en paralelo y el uso de recursos como LUTs y BRAMs. Así se puede ajustar el diseño a los límites del Zynq−7000 [10].

El nivel L2 se construye sobre la base anterior y junta las piezas básicas. De esta forma, ofrece algoritmos completos como los sistemas de seguridad AES−128 o SHA−256 [13]. Estos bloques están listos para usar porque sus mejoras ya fueron revisadas y probadas con pruebas técnicas. Esto ayuda a que haya menos errores al instalarlos. El nivel L3 da herramientas para que el software se conecte con el hardware. Esto es importante para comparar de forma directa el funcionamiento del software con el hardware [10].

Por su parte, el nivel L3 integra marcos de interacción entre el software host y los núcleos acelerados. Sin embargo, en arquitecturas SoC Zynq-7000 donde se busca control determinista sobre la transferencia de datos y la latencia extremo a extremo, la integración suele realizarse directamente mediante interfaces AXI gestionadas desde el PS. Por esta razón, el presente trabajo prescinde del nivel L3 y prioriza un control explícito sobre la comunicación hardware/software [10]. Para gestionar estos niveles, el sistema sigue una pila de software que conecta la aplicación host con los núcleos implementados en la lógica programable.

Figura 5

Ecosistema de dominios y flujo de aceleración en las AMD Vitis™ Libraries.



Nota. El diagrama muestra la organización de bibliotecas comunes y específicas por dominio, así como las opciones de despliegue (edge, on-premise y cloud). En esta investigación se emplea el dominio de seguridad para la implementación de aceleradores criptográficos sobre SoC-FPGA [12].

En síntesis, la adopción de los niveles L1 y L2 dentro de la Vitis™ Security Library permite reutilizar bloques optimizados sin sacrificar control micro arquitectónico. Esta decisión

metodológica es coherente con el objetivo de diseñar aceleradores criptográficos en hardware reconfigurable y evaluar cuantitativamente su rendimiento frente a soluciones puramente software [14].

Biblioteca AMD Vitis™ Security para aceleración criptográfica

La biblioteca AMD VitisTM Security es un conjunto de funciones creadas en C++ que tienen como objetivo la ejecución directa de la criptografía en la lógica programable (PL). Esta herramienta facilita la transferencia de la carga computacional del Sistema de Procesamiento (PS) a aceleradores hardware que se especializan en arquitecturas Zynq-7000. La biblioteca se divide en dos niveles: las primitivas fundamentales (L1) y los estándares completos, como SHA o AES (L2) [3], [12]. La puesta en práctica de estos módulos, que han sido validados con anterioridad, reduce de forma considerable los riesgos de seguridad y hace innecesaria la codificación a bajo nivel (RTL). Por lo tanto, el ingeniero tiene la posibilidad de darle mayor importancia a la gestión de memoria y a la arquitectura del sistema por encima de la verificación manual funcional [12], [14], [14].

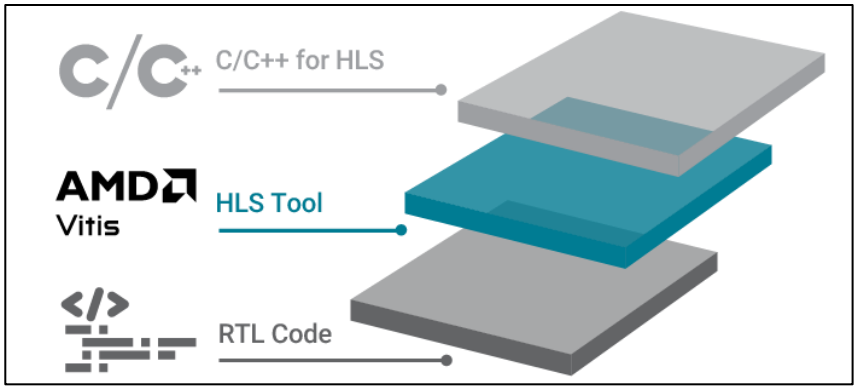
HLS (High-Level Synthesis) y su rol en SoC

La Síntesis de Alto Nivel (HLS) es un proceso de diseño que transforma código de alto nivel (generalmente escrito en C, C++ o SystemC) en una descripción RTL (Register-Transfer Level). Este es un paso crítico en las arquitecturas SoC-FPGA porque reduce drásticamente el tiempo de diseño y verificación en comparación con el desarrollo manual en RTL [9]. En un SoC como el Zynq-7000, HLS permite especificar la lógica de un acelerador, por ejemplo, un motor AES utilizando estructuras de programación familiares, cuyo proceso de conversión de C/C++ a RTL para su implementación en FPGA se muestra en la Figura 6. La herramienta Vitis HLS, integrada en el VitisTM Unified IDE, transforma el código C++ en bloques de propiedad intelectual (IP) optimizados para la lógica programable (PL), empleando directivas (pragmas) para la paralelización, canalización (pipelining) y múltiples modalidades de mapeo de memoria (RAM). El producto de HLS representa un núcleo de hardware que puede integrarse de manera

directa en la interconexión AXI dentro de la Programación Lógica (PL), lo que facilita la transmisión de datos con el Sistema de Procesamiento (PS) [16].

Figura 6

Proceso de síntesis de alto nivel de C/C++ a RTL para implementación en FPGA.



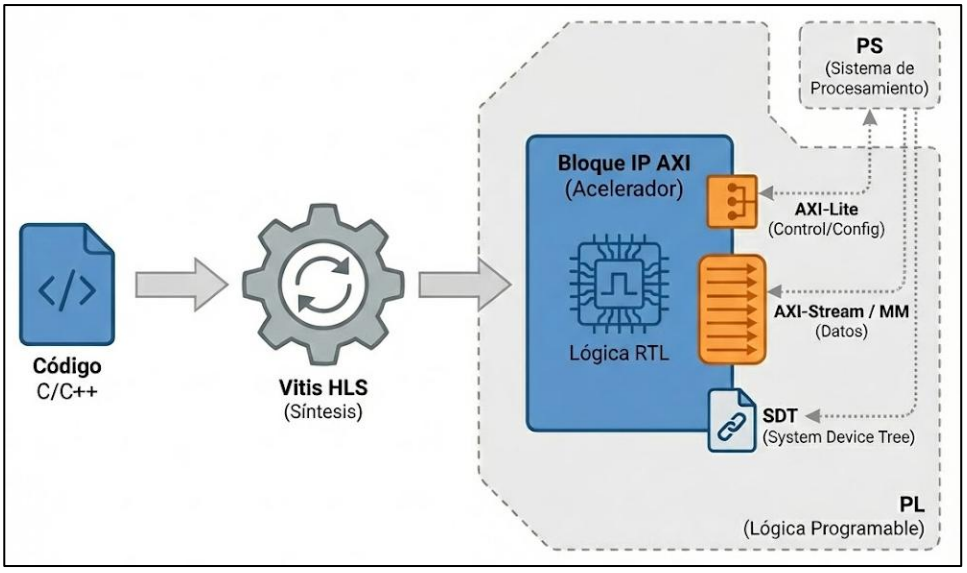
Nota. Vitis HLS traduce el código C/C++ del acelerador en lógica RTL optimizada para el PL, permitiendo ajustes en paralelismo, latencias y recursos mediante directivas. [9]

Generación y empaquetado de IPs AXI en AMD Vitis™ Unified IDE (UID)

HLS genera un bloque IP con interfaz AXI que se convierte en la pieza esencial del acelerador de hardware dentro del SoC, cuyo flujo de síntesis para la generación de un IP AXI mediante Vitis HLS se presenta en la Figura 7. En ese bloque queda encapsulado el algoritmo escrito en C o C++, ya traducido a lógica RTL, y su comunicación se realiza mediante AXI, que es el protocolo habitual para mover datos y señales de control dentro de la arquitectura. Gracias a este enlace, el Sistema de Procesamiento (PS) puede interactuar sin interrupciones con los módulos que se implementan en la Lógica Programable (PL) [17]. En la práctica, esto quiere decir que el resultado de HLS es un módulo que se integra directamente en el sistema restante y tiene la capacidad de funcionar con él sin necesidad de realizar modificaciones adicionales.

Figura 7

Flujo de síntesis de alto nivel con Vitis HLS para generación de IP AXI.



Nota. Vitis HLS convierte el código C/C++ en un IP con interfaz AXI para integrarlo como acelerador en la plataforma embebida.

Vitis HLS, tras la creación del RTL, lo encapsula en un bloque IP AXI que ya tiene incorporadas las conexiones requeridas para conectar con el SoC. En la práctica, AXI-Lite se encarga de las tareas de control y configuración, mientras que AXI Memory-Mapped y AXI Stream gestionan el flujo de datos cuando se requiere mayor rendimiento [9]. Junto con estas interfaces, el empaquetado incorpora la información que usa el flujo unificado de Vitis, administrada por el System Device Tree (SDT). Dichos metadatos permiten que el software embebido construya los drivers adecuados y mantenga una comunicación estable entre el PS y el acelerador implementado en la PL [16].

Vivado: síntesis e implementación de IPs

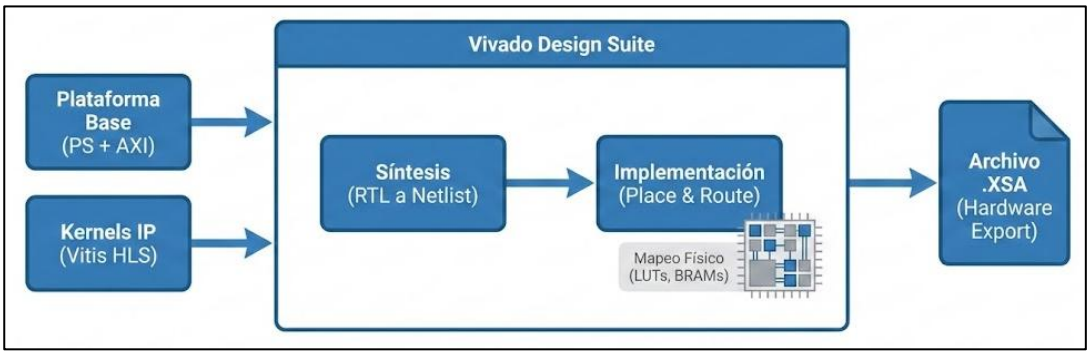
Vivado™ Design Suite actúa como la integración final y el mapeo físico de todos los componentes del SoC-FPGA en el PL [18]. Vivado toma el RTL de la plataforma base (que incluye el PS y la trama de interconexión AXI) y los núcleos de cifrado AXI de Vitis HLS generados previamente [17].

La síntesis se divide en otros dos procesos, en primer lugar, en la síntesis RTL se realiza la conversión del diseño RTL completo en su versión netlist, posteriormente, esta netlist

se optimiza según la arquitectura Zynq. En segundo lugar, en la implementación se realiza el place & route de las compuertas, se refiere a la asignación física de la lógica del acelerador a los elementos variables de la FPGA (LUTs, y BRAMs) [18]. Este proceso se ilustra en la Figura 8. Las continuas mejoras en Vivado, incluyendo las optimizaciones en el ruteo y colocación, buscan mejorar el Quality of Results (QoR) para diseños complejos de alta frecuencia [19]. El resultado de la implementación es un archivo Xilinx Shell Archive (XSA), que es indispensable para crear la plataforma de desarrollo del software embebido en Vitis UID [17].

Figura 8

Proceso de síntesis e implementación de hardware en Vivado Design Suite.



Nota. Vivado integra y sintetiza el Zynq con los kernels para el desarrollo en Vitis.

Software embebido en el sistema de procesamiento (PS) y control de IPs

Los núcleos ARM del Sistema de Procesamiento (PS) ejecutan el software embebido. Este componente actúa como el centro de control del SoC, orquestando y gestionando todos los aceleradores de hardware. Este software host interactúa con el acelerador de cifrado en la PL a través de las APIs de driver que se generan automáticamente para los IPs AXI. El control del acelerador sigue una secuencia bien definida: en primer lugar, el software debe transferir los datos a cifrar desde la memoria principal (DDR) hacia los buffers del acelerador en la PL; en segundo lugar, se controla el IP enviando el comando de inicio (ap_start) vía la interfaz AXI-Lite; y finalmente, el software se sincroniza esperando la señal de finalización (ap_done) antes de recuperar y procesar los resultados cifrados [20].

Aceleradoras mediante AMD Vitis™ Unified IDE (UID)

El propósito primordial de esta tesis es el diseño del acelerador. Se caracteriza como un núcleo de hardware optimizado que se despliega en la PL con el fin de descargar del PS las tareas de cifrado que requieren mucha capacidad, lo cual permite incrementar significativamente el rendimiento. El Vitis™ Unified IDE reúne todas las herramientas requeridas para este desarrollo acelerado, abarcando desde la síntesis HLS hasta el perfilado y la depuración de todo el sistema [21].

La metodología de aceleración dentro de Vitis es exhaustiva. Comienza estableciendo una línea base de rendimiento, midiendo la latencia y el throughput de la función criptográfica en software puro (ejecutándose en el ARM del PS) [20]. Luego, el diseño se paraleliza en la PL para maximizar el throughput y optimizar la transferencia de datos de E/S entre el PS y la PL, una gestión de E/S que reduce el tiempo de inactividad de la CPU y asegura que la aceleración por hardware se traduzca en una mejora del rendimiento general del sistema [20].

Python como referencia y verificación de algoritmos criptográficos

El lenguaje de programación Python es indispensable en las fases iniciales de diseño y para la validación comparativa de los resultados. Adicionalmente, Python ofrece un ambiente de modelización que permite la simulación funcional de algoritmos criptográficos como AES o SHA previo a su implementación en hardware [22].

El autor verifica que el modelo Python genere las salidas correctas para datos de prueba conocidos usando bibliotecas criptográficas de alto nivel (como pyca/cryptography) o NIT's (vectores de prueba) [22]. La verificación temprana asegura que el diseño funcione como se espera, en lugar de perder tiempo implementando un algoritmo incorrecto en hardware. Además, el modelo en Python establece la línea base de rendimiento para poder comparar limpiamente el algoritmo ejecutado solo en software (CPU) con la versión final acelerada en la PL del SoC-FPGA [20].

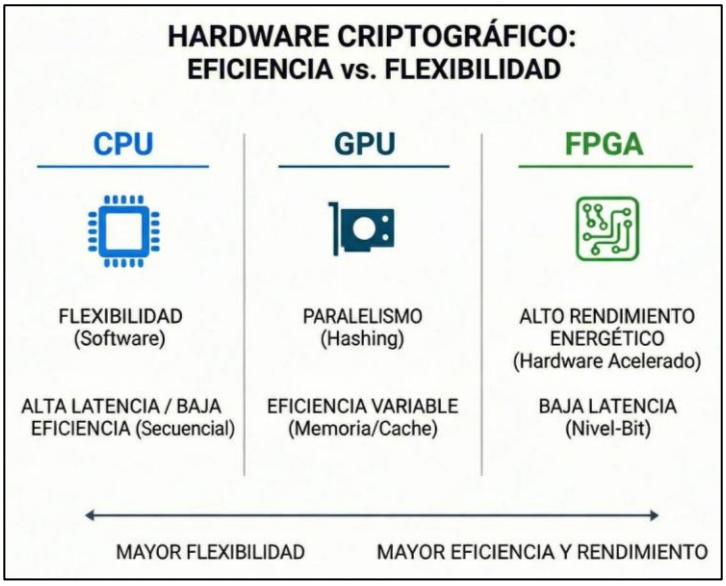
Aceleración criptográfica en hardware reconfigurable

Diferencia entre CPU, GPU y FPGA en tareas criptográficas

Desde el punto de vista energético, la plataforma hardware sobre la que se implementen los algoritmos criptográficos define el punto de equilibrio entre flexibilidad, eficiencia y rendimiento del sistema. Mediante un software de uso general, un CPU hace posible la puesta en marcha de algoritmos como AES o SHA, lo que proporciona una notable versatilidad. Sin embargo, su arquitectura secuencial a la vez limita el paralelismo y eleva la latencia en situaciones con un alto rendimiento, como las redes de telecomunicaciones [23], [24]. Un GPU es apropiado para tareas con un alto grado de paralelismo, como el hashing, ya que optimiza el rendimiento a través del aprovechamiento de este tipo de paralelismo. Sin embargo, su latencia y eficiencia energética dependen del patrón de acceso a la memoria y del tipo de carga [5]. Por el contrario, en aplicaciones criptográficas embebidas, una FPGA posibilita la creación de aceleradores hardware especializados que controlan con exactitud el pipelining y el paralelismo a nivel de bit, lo cual permite un rendimiento superior por watt y una latencia menor en comparación con CPU y GPU [2], tal como se sintetiza en la Figura 9.

Figura 9

Comparativa de hardware criptográfico: CPU vs. GPU vs. FPGA.



Nota. Se compara el desempeño criptográfico de la Unidad Central de Procesamiento (CPU), la Unidad de Procesamiento Gráfico (GPU) y las FPGA, destacando que estas últimas son las que tienen un mejor rendimiento.

Ventajas del paralelismo en hardware para algoritmos AES y SHA

El AES-128, que opera con secuencias de diez rondas de transformación sobre bloques de datos que tienen 128 bits, es el estándar de facto del cifrado simétrico por bloques [24]. El cifrado lleva a cabo iterativamente operaciones de permutación, mezcla y sustitución con la clave; por su parte, el descifrado realiza las transformaciones inversas empleando la misma clave secreta. La posibilidad de implementar la encriptación y desencriptación de manera efectiva en hardware reconfigurable es factible gracias a esta simetría, lo que resulta ventajoso para las arquitecturas pipeline que consiguen un throughput alto con una latencia baja [4]. SHA-256, en cambio, es una función hash de naturaleza criptográfica que emplea una función de compresión iterativa y padding para procesar mensajes de longitud arbitraria y generar un resultado fijo de 256 bits [19]. Debido a estas propiedades estructurales, SHA-256 se beneficia significativamente de la aceleración en FPGA, particularmente en aplicaciones que necesitan una verificación de integridad ágil [2], [25].

Criptografía simétrica y hashing sobre hardware reconfigurable (AES y SHA)

Dado que hashing y criptografía simétrica son las funciones fundamentales para asegurar los datos de alto throughput en telecomunicaciones, la aceleración en hardware reconfigurable se enfoca sobre todo en ellas. AES se utiliza como cifrado simétrico estándar para asegurar la privacidad de los flujos de datos ininterrumpidos, y su estructura interna favorece las aplicaciones altamente paralelas en FPGA [2], [24]. SHA-256, por otro lado, emplea una huella digital de longitud fija para garantizar la autenticidad y la integridad de los datos, lo que posibilita identificar cualquier modificación en los datos enviados [23]. Bibliotecas como Vitis Security, en plataformas AMD-Xilinx, ofrecen implementaciones documentadas y

optimizadas de estos algoritmos, lo que disminuye el esfuerzo dedicado a verificar y diseñar sistemas SoC-FPGA [9], [18].

Fundamentos y algoritmos criptográficos utilizados

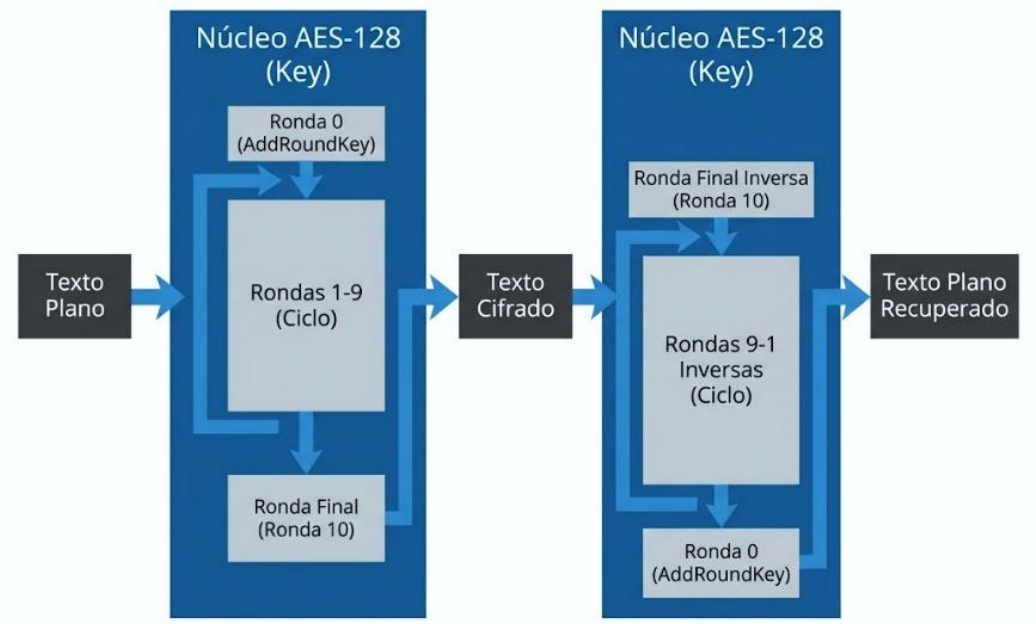
La criptografía se clasifica según el uso de claves y la forma en que transforma los datos dentro de un sistema de comunicación. La criptografía simétrica utiliza una única clave para cifrar y descifrar información, lo que la hace adecuada para grandes volúmenes de datos, como ocurre con AES. La criptografía asimétrica usa pares de claves y se usa principalmente para autenticación e intercambio seguro de claves, pero es más costosa computacionalmente que los esquemas simétricos [24]. Los algoritmos pueden ser en bloque o en flujo, dependiendo de cómo procesen los datos, y los esquemas AEAD, como AES, ofrecen confidencialidad e integridad en una sola construcción, simplificando el diseño de protocolos seguros contemporáneos [26].

Algoritmos AES-128 y SHA-256

AES-128 es el algoritmo estándar de cifrado simétrico en bloques sobre bloques de datos de 128 bits, mediante 10 rondas de transformación [24]. El cifrado aplica repetidas veces una transformación de sustitución, permutación y mezcla con la clave; el descifrado hace lo contrario con la misma clave secreta. Esta simetría permite una implementación eficiente tanto de la encriptación como de la encriptación en hardware reconfigurable, para arquitecturas pipeline de alto rendimiento y baja latencia [2], cuyo esquema funcional se presenta en la Figura 10.

Figura 10

Diagrama funcional del cifrado autenticado AES-128.

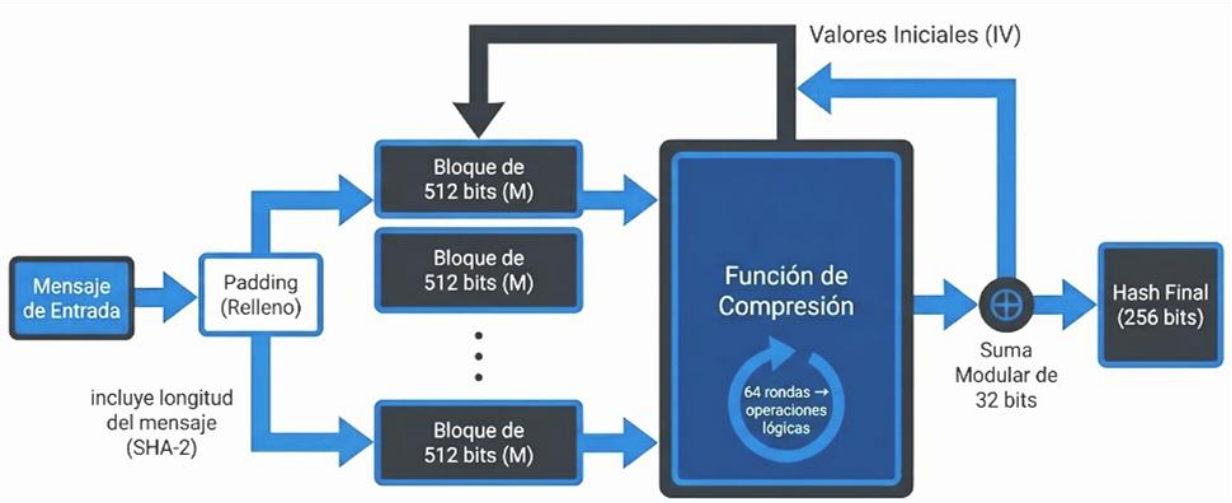


Nota. Se ilustra el cifrado AES que combina confidencialidad y autenticación.

Por otro lado, SHA-256 es una función hash criptográfica que recibe como entrada mensajes de longitud variable y genera una salida de longitud fija de 256 bits, mediante relleno y una función de compresión iterativa [23], cuyas etapas principales se ilustran en la Figura 11. Estas propiedades estructurales son la causa de que SHA-256 aproveche la aceleración en FPGA para aplicaciones de verificación rápida de integridad [4].

Figura 11

Etapas del algoritmo hash SHA-256 para generación del digest.



Nota. Se muestran las etapas del proceso hash hasta obtener un digest de 256 bits.

Aplicaciones de la criptografía en telecomunicaciones

Ambos algoritmos SHA-256 y AES se utilizan en combinación en muchos protocolos de seguridad aplicados en el área de telecomunicaciones. La encriptación simétrica basada en AES en TLS y otros protocolos de seguridad de red garantiza la confidencialidad de la información, y las funciones hash criptográficas como SHA-256 garantizan la integridad y autenticidad de la información [27].

Si bien protocolos como IPsec y VPNs modernas emplean esquemas de cifrado autenticado como AES-GCM para integrar ambas propiedades, en implementaciones hardware con recursos limitados es habitual separar el cifrado y la autenticación en bloques funcionales independientes, permitiendo optimizar el uso de lógica reconfigurable y cumplir con requisitos de latencia y consumo energético en escenarios como redes 5G, IoT y edge computing [25].

Capítulo III: Diseño e Implementación del Sistema

Arquitectura general del sistema

Se elaboró una arquitectura integral del sistema que posibilita la comparación imparcial de implementaciones a nivel de hardware y software en un entorno controlado, con el objetivo de implementar y evaluar aceleradores criptográficos sobre una plataforma SoC-FPGA. La arquitectura heterogénea del sistema es Zynq-7000, en la que el sistema de procesamiento ARM se encarga de gestionar y controlar la memoria, y la lógica programable realiza operaciones criptográficas intensivas a través de paralelismo hardware. Esta metodología sigue los principios de computación reconfigurable que ya se habían implementado en ambientes académicos e investigativos [1], [7]. Este método posibilita disminuir la carga del procesador y beneficiarse de la eficacia propia del hardware especializado, como se ha evidenciado en investigaciones recientes acerca de aceleradores en SoC-FPGA [2].

Desde la perspectiva funcional y del rendimiento, la arquitectura posibilita el procesamiento de algoritmos criptográficos aplicados a telecomunicaciones, en particular AES-128. Asegura que los resultados alcanzados en hardware sean similares a los modelos de referencia en software, de acuerdo con las normas establecidas por NIST [24], [26]. Distinguir entre control y cómputo intensivo hace más fácil calcular métricas como el uso de recursos, la latencia y el rendimiento. Esto permite que se comparen directamente las ejecuciones puramente de software con las aceleradas en hardware. Esta clase de arquitectura ha probado ser particularmente eficaz en entornos de computación perimetral, IoT y sistemas que tienen limitaciones de energía y latencia, en los que la aceleración criptográfica en FPGA proporciona beneficios evidentes con respecto a soluciones convencionales [4], [5].

La organización del proceso de procesamiento del sistema se realiza de manera gradual y reproducible, comenzando con la creación y verificación funcional de los algoritmos en Python, empleando vectores de prueba reconocidos y bibliotecas criptográficas avanzadas para definir una línea base confiable. Después, las implementaciones se transfieren a C/C++

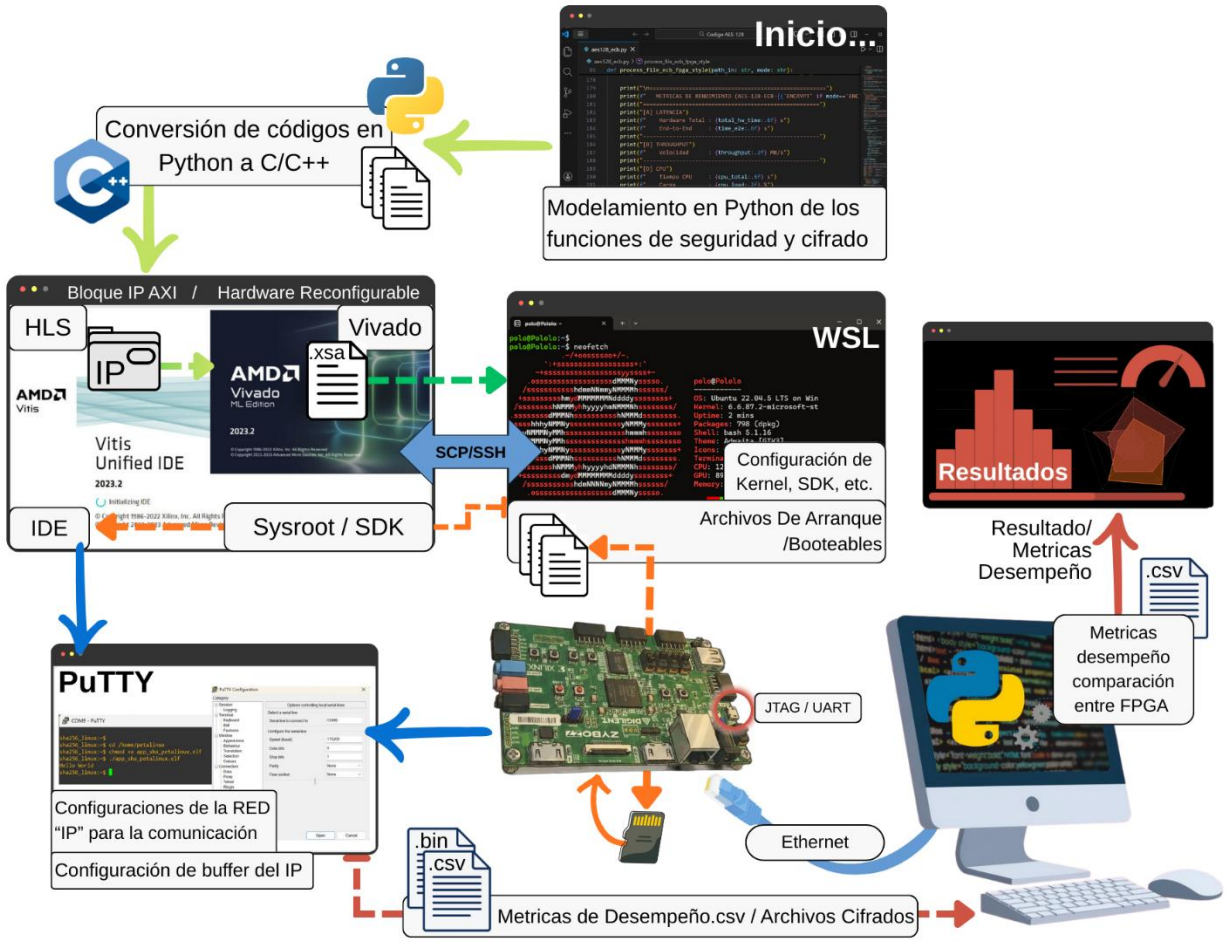
utilizando AMD Vitis Libraries y se sintetizan con Vitis HLS para crear núcleos hardware optimizados, de acuerdo con el flujo de diseño sugerido por AMD-Xilinx [3], [9]. Estos núcleos son incorporados en Vivado como bloques IP con interfaces AXI y, por último, son gestionados por software embebido creado en Vitis Unified IDE. De esta forma, se asegura un flujo lógico desde la modelación inicial hasta la ejecución acelerada en hardware [18], [20].

El flujo completo de la implementación, detallado en la Figura 12, integra diseño de hardware y ejecución de software. El proceso comienza en la fase de síntesis. En esta etapa, se realiza la validación del código C/C++, y se hace la exportación de este código a un núcleo IP (Intellectual Property). Posteriormente, Vivado incluye este componente y con esto define las conexiones de memoria y de interrupciones que serán necesarias para la generación del archivo de hardware correspondiente (.xsa). Este documento define la lógica programable y ajusta el sistema para que se pueda operar en un entorno de montaje.

La fase operativa conecta el hardware con la aplicación de gestión. Para esto, la ejecución del sistema Linux se realiza con los archivos de arranque que se encuentran grabados en una tarjeta microSD. Este sistema establece comunicación con la computadora a través de un enlace de Ethernet SSH que utiliza direcciones IP estáticas. En el entorno de Vitis, la aplicación de software invoca el controlador de la IP para que el procesamiento de los datos no se realice en la CPU, y para que el cálculo criptográfico sea llevado a cabo por el hardware. Finalmente, el sistema genera y exporta los resultados cifrados (.bin) y las métricas de rendimiento (.csv) para la validación de un contraste con los modelos en Python.

Figura 12

Proceso de diseño, integración y validación del acelerador criptográfico en una plataforma SoC-FPGA.



Nota. Se muestra el proceso total: la creación del IP, su incorporación en Zynq y la comparación de desempeño entre la FPGA y la CPU.

Modelo de referencia en software para AES-128 y SHA-256 (Python)

El procedimiento para la verificación de ambos modelos se basa en la integración de casos producidos bajo condiciones de control y vectores de prueba estándar (NITS). Durante las etapas iniciales, se emplean vectores de prueba suministrados por fuentes confiables, como el NIST. En el marco del protocolo AES-128, se seleccionan tanto las claves como los pares preexistentes de texto plano y cifrado para su selección. El software encripta el texto sin formato a través de una clave preestablecida, garantizando una alineación precisa de la salida con el texto encriptado de manera bit a bit. Con el objetivo de corroborar la correspondencia

entre la cadena de prueba estándar ("abc") y otras de mayor envergadura, tal como se observa en SHA-256.

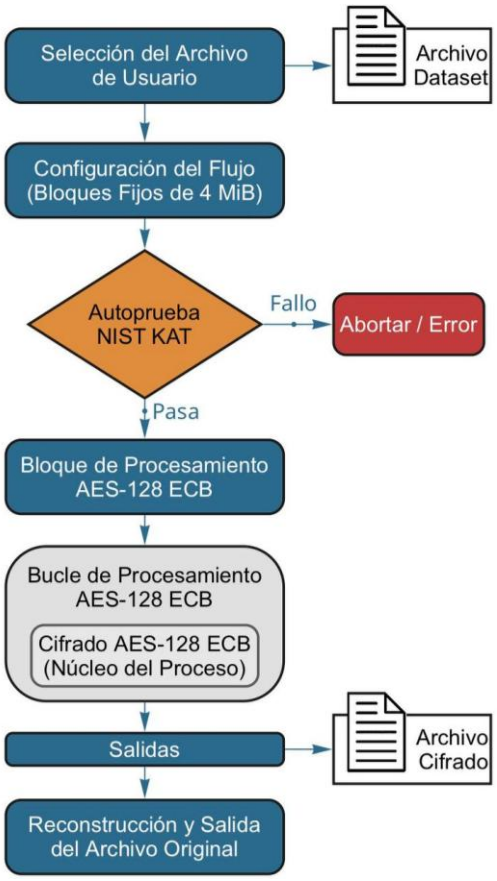
Algoritmo AES-128

La biblioteca PyCryptodome se usó para poner en práctica el modelo de referencia en software del algoritmo AES-128, porque permite llevar a cabo operaciones de cifrado con menos carga que la librería estándar de Python y proporciona primitivas criptográficas optimizadas en C. Esta implementación tiene como función verificar el funcionamiento del sistema antes del proceso de aceleración por medio de hardware , siguiendo el flujo funcional que se resume en la Figura 13.

La arquitectura del sistema procesa los datos de entrada en bloques de tamaño fijo, lo que permite administrar archivos grandes sin tener que guardar toda la información en la memoria. Esta metodología mantiene un procesamiento continuo de "streaming" que es similar al comportamiento previsto en la plataforma FPGA. Además, se llevó a cabo una etapa de auto-verificación previa al cifrado por medio de los vectores de prueba KAT (Known Answer Test) que el NIST definió en SP 800-38A. Esto garantiza que los bloques funcionales son apropiados antes de ser trasladados a la implementación tangible y que los procedimientos de encriptación y desencriptación se realizan de acuerdo con el estándar internacional.

Figura 13

Flujo funcional del modelo software AES-128



Nota. Se muestran las etapas de entrada, validación KAT, cifrado/descifrado y salida de datos.

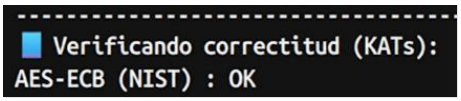
El algoritmo de cifrado que esta implementación utiliza es AES-128 en modo ECB. Este modo de operación realiza el cifrado de manera independiente en cada bloque de 128 bits (cada bloque es un bloque de datos) usando una misma clave secreta para el cifrado y el descifrado.

Esta implementación comprueba que el cifrado AES-128 modo ECB está funcionando correctamente antes de proceder a encriptar archivos completos. AES-NI y los vectores de prueba definidos por el NIST, como se ilustra en la Figura 14, que incluyen la clave criptográfica, el texto plano, y el texto cifrado, son utilizados. La validación de la salida obtenida por el software y los valores de la norma se expresa en la salida de la consola.

Este procedimiento permite confirmar que el proceso de cifrado se ejecuta de manera correcta y conforme al estándar criptográfico, asegurando la correcta transformación de los datos antes de su procesamiento posterior.

Figura 14

Validación KAT para AES 128-ECB



Nota. Resultado de la verificación y validación del cifrado.

El modelo software constituye la referencia funcional para la siguiente fase de diseño.

Los bloques lógicos desempeñarán un papel orientador en la generación de las IPs criptográficas y su integración subsecuente en el flujo AXI de la FPGA.

Algoritmo SHA-256

Se estableció el modelo de integridad a través del empleo del algoritmo SHA-256, un sistema criptográfico que asegura que la información procesada no sufra alteraciones en su tránsito o almacenamiento. Para desarrollarla, se utilizó la biblioteca estándar hashlib, ya que posibilita el uso directo de las mejoras de hardware que están presentes en el procesador del sistema. De esta manera, se asegura una implementación eficaz y completamente compatible con cualquier entorno Python contemporáneo.

El programa fue diseñado teniendo en cuenta una estructura orientada a flujos de datos. El sistema procesa el archivo de manera constante en bloques de 4 MiB, en vez de cargarlo entero en la memoria. Esta estrategia posibilita la administración de archivos muy grandes sin perjudicar los recursos del anfitrión. Este método es coherente con el mecanismo de streaming que se emplea más adelante en la implementación acelerada por hardware.

El sistema se somete a una evaluación funcional utilizando vectores de prueba aprobados (KAT) publicados previamente por el NIST como se muestra en la Figura 15 antes de la generación del hash final. Esta comprobación implica cotejar el digest obtenido con el que

se anticipaba. La prosperidad se manifiesta en la ausencia de errores en las operaciones internas de concatenación y compresión, así como en la fluidez total del hash.

Figura 15

Validación KAT del modelo software SHA-256

Nota. Coincidencia exacta entre el digest computado localmente y el valor estándar del NIST.

Una vez validado, el sistema posibilita que se determine el hash de cualquier archivo que el usuario necesite. El digest resultante es producido en formato hexadecimal. Esto hace más fácil la verificación automatizada de la integridad, el rastreo del contenido almacenado y las auditorías que se realizan después. Esta salida se empleará como parámetro de comparación para validar la integridad del archivo tras haber acelerado el sistema en FPGA.

Esta implementación garantiza que cualquier cambio en la información, ya sea accidental o deliberado, pueda ser detectado de inmediato si el valor del hash no coincide con el original. Por lo tanto, el modelo de software desarrollado durante esta etapa será la base de integridad para validar funcionalmente la implementación del hardware que se realice más adelante. Esto asegura que el sistema en su totalidad continúe manteniendo el nivel de fiabilidad definido en términos criptográficos.

Generación de IPs criptográficas en Vitis HLS

Vitis HLS produce identidades criptográficas que posibilitan la conversión de una descripción funcional de alto nivel en un bloque hardware que puede ser sintetizado e incorporado a un sistema SoC basado en Zynq. Esta metodología permite comenzar con un modelo que ya ha sido validado en el entorno del software y avanzar poco a poco hasta su implementación física, supervisando desde las etapas iniciales el rendimiento temporal del diseño y la utilización de los recursos [28].

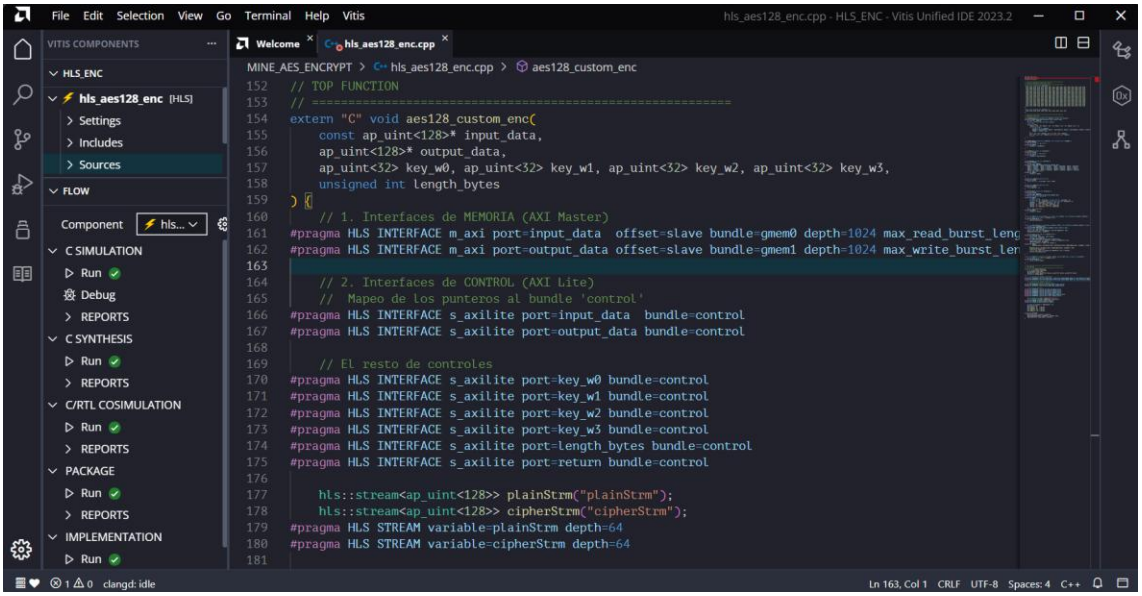
El protocolo empieza cuando se configura el componente HLS, en la que se definen la duración del reloj, las rutas de inclusión necesarias y el dispositivo destinatario. En este estudio, se eligió a la familia Zynq-7000 como plataforma de destino y se configuró el flujo de salida como IP Catalog; esto simplifica su integración en Vivado en el futuro. Asimismo, se establece la conexión con la Vitis Security Library, la cual proporciona una implementación especializada del algoritmo SHA-256 mediante funciones xf_security, garantizando el cumplimiento del estándar criptográfico requerido [15].

Como parte del desarrollo, el autor implementó y ajustó el código HLS de los aceleradores AES-128 y SHA-256, incorporando directivas pragma que modelan explícitamente la arquitectura hardware resultante. Estas directivas no son decorativas, ya que definen interfaces, paralelismo, partición de arreglos y flujo de datos, y con ello condicionan directamente la latencia, el intervalo de inicio y la utilización de LUTs y BRAM durante la síntesis [16], [29]. Por esta razón, se presentan extractos representativos del código, los cuales evidencian el trabajo de programación y la complejidad de configurar correctamente el comportamiento del núcleo en hardware sin incluir el código completo.

La parte del acelerador AES-128 que incluye instrucciones como #pragma HLS INTERFACE se encuentra en la figura 16. AXI4-Lite se usa para ilustrar el plano de control, mientras que AXI4-Master se emplea para representar el plano de datos hacia la memoria DDR. Al incorporar el SoC, estos patrones son comunes [30], [31], [32]. El gráfico 17, además, presenta un resumen del acelerador SHA-256, resaltando los pragmas que se utilizan para optimizar el procesamiento por medio de pipelining, paralelismo o flujo de datos y las instrucciones para fraccionar arreglos cuando sea necesario. En conjunto, estos pragmas permiten convertir la descripción C/C++ en una microarquitectura concurrente adecuada para FPGA, manteniendo equivalencia funcional con el modelo de referencia [33], [34].

Figura 16

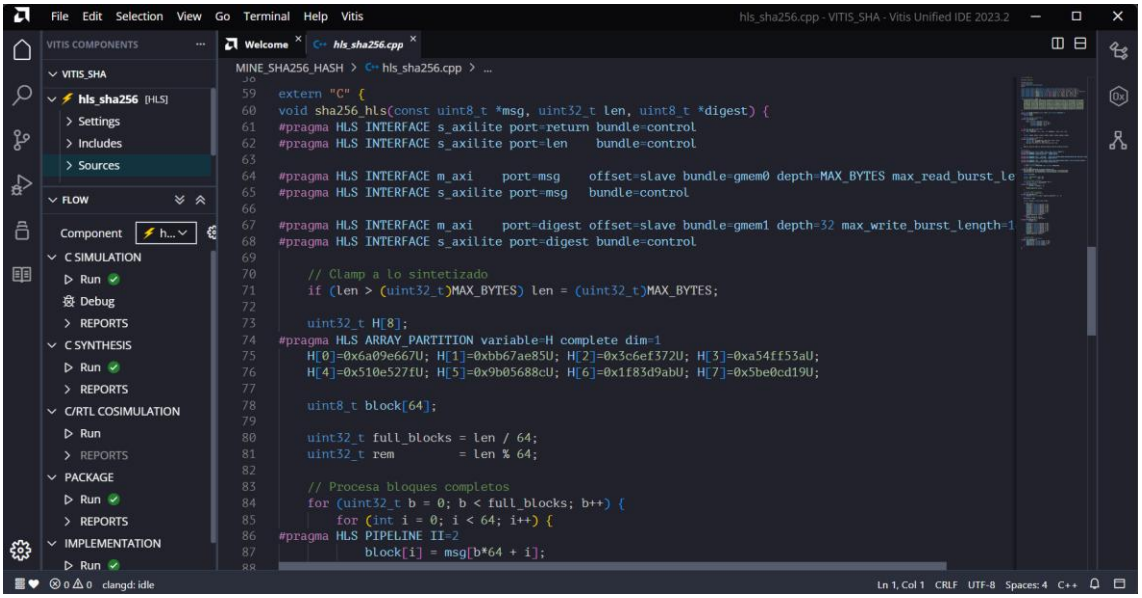
Extracto del código HLS del acelerador AES-128 (implementación y pragmas).



Nota. Se presenta un extracto representativo donde se observan directivas HLS para interfaces AXI y optimización del flujo de datos, sin incluir el código completo.

Figura 17

Extracto del código HLS del acelerador SHA-256 (implementación y pragmas).

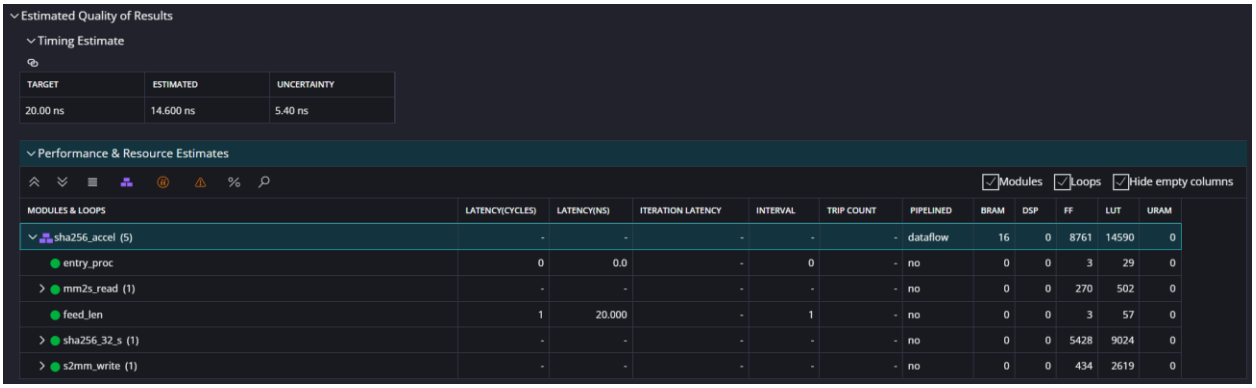


Nota. Se muestra un extracto representativo con directivas HLS asociadas a interfaces y optimización, empleadas para modelar el núcleo hardware.

La configuración del componente HLS se muestra en la Figura 18, donde se observan los parámetros que condicionan la síntesis, incluyendo el dispositivo objetivo, el periodo de reloj configurado en 20 ns y las rutas de inclusión necesarias [28].

Figura 18

Archivo de configuración del componente HLS en Vitis.



Nota. Se muestran el dispositivo objetivo, el periodo de reloj configurado en 20ns, rutas de inclusión y parámetros del flujo de síntesis [28].

La principal función corresponde a sha256_hl, que incorpora el acelerador de hardware desarrollado. Las interfaces AXI se establecen a través de directivas HLS con la finalidad de diferenciar entre el plano de datos y el plano de control. AXI4-Master posibilita la transferencia directa entre el acelerador y la memoria DDR, en contraposición a AXI4-Lite, que se ocupa de los registros vinculados con la configuración y el inicio del procedimiento. Este esquema sigue el modelo recomendado para la integración de núcleos en arquitecturas SoC con memoria compartida [29], [30], [31].

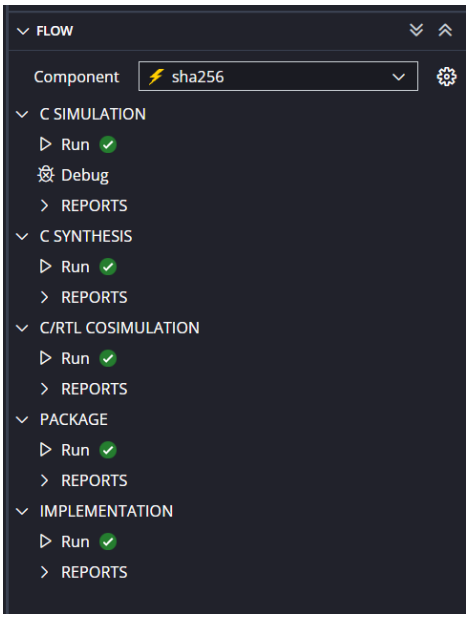
El proceso de validación y generación de la IP sigue un flujo secuencial estructurado dentro del Flow Navigator, tal como se muestra en la Figura 19. Este flujo define las etapas formales que transforman el modelo en C/C++ en un núcleo hardware listo para integrarse en el sistema. En primer lugar, la etapa C Simulation verifica el comportamiento funcional del algoritmo a nivel de alto nivel, comparando los resultados con los vectores de prueba definidos

previamente. Esta fase asegura corrección algorítmica antes de cualquier transformación estructural [34].

En segundo lugar, C Synthesis convierte el código en una descripción RTL optimizada, estimando latencia, intervalo de inicio y utilización de recursos. En esta etapa se generan los primeros reportes cuantitativos que permiten evaluar el impacto del diseño sobre LUTs, BRAM, etc. Posteriormente, la etapa C/RTL Co-Simulation compara ciclo a ciclo el modelo en C con la implementación RTL generada, garantizando equivalencia funcional y consistencia [33]. Una vez validado el diseño, la fase Package encapsula el núcleo como un bloque IP compatible con AXI e incorpora los metadatos necesarios para su integración en Vivado. Finalmente, la etapa Implementation consolida las estimaciones de temporización y recursos, generando métricas que fundamentan el análisis de desempeño presentado en esta tesis [34].

Figura 19

Flujo de validación y empaquetado del componente SHA-256 en Vitis Unified IDE.



Nota. El flujo secuencial incluye simulación en C, síntesis RTL, co-simulación, empaquetado e implementación antes de generar la IP final [28].

Produce un sistema de tres pasos. En el primer sistema, se describe el marco de trabajo de la lógica de RTL y también describe un sistema de tres pasos integrados. Esto se debe a que el sistema de tres pasos es un marco universitario. Esto se debe a que, a diferencia de otros productos técnicos, el sistema de tres pasos no se utiliza como un marco técnico aplicable a otros productos técnicos. Por ejemplo, en la lógica técnica se describe. Por este motivo, en la lógica de tres pasos se describe el marco técnico y se basa en el marco de sistema.

La estructura interna del acelerador se verifica mediante diagramas jerárquicos, tras la transposición del diseño al hardware, cuya organización modular generada por Vitis HLS se muestra en la Figura 20. Estos permiten la identificación de las funciones fundamentales del cálculo: la preprocesación del mensaje, la generación del programa de mensajes y el cómputo iterativo del resumen final. Esta entidad cumple con la arquitectura oficial preestablecida para la función hash SHA-256 en el artículo [15]. Por su parte, el analizador Dataflow, representado en la Figura 21, especifica la secuencia de activación de los módulos internos y las dependencias de datos que tienen entre sí. La orquestación a través de streams posibilita que los bloques del mensaje se procesen sin interrupciones, lo que permite un flujo continuo y eficaz [33].

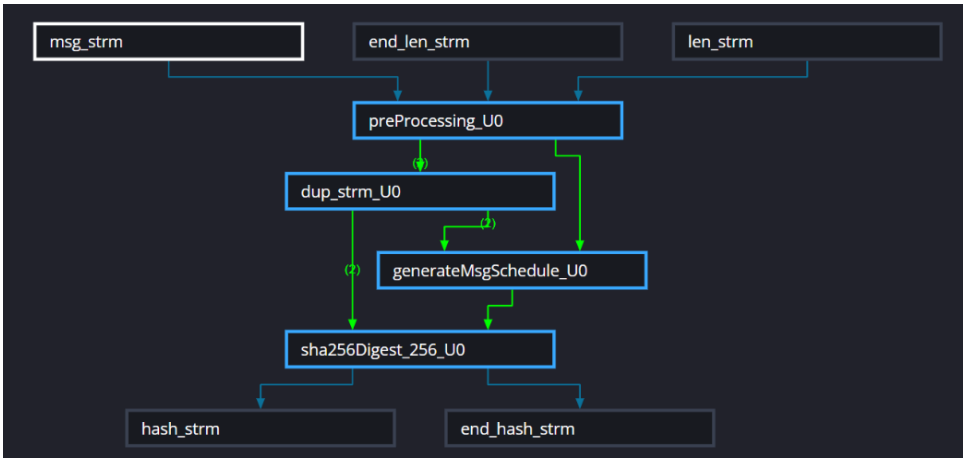
Figura 20

Estructura jerárquica del componente sha256_hls generada por Vitis HLS.

Nota. Los submódulos se generan de acuerdo con la división funcional del algoritmo hash.

Figura 21

Vista de Dataflow del módulo SHA-256 en HLS.



Nota. Los flujos de operación msg_strm, len_strm, end_len_strm y hash_strm simbolizan la secuencia de operaciones dentro del módulo.

Adicionalmente, el visor de cronograma (Schedule Viewer) detalla la ejecución ciclo a ciclo de las funciones aceleradas, como se aprecia en la Figura 22. La gráfica evidencia que la lectura de memoria (mm2s_read) y la configuración de longitud (feed_len) operan en paralelo entre los ciclos 1 y 3. Tras esta fase de ingesta, el núcleo criptográfico sha256_32_s inicia el procesamiento exclusivo del hash en el ciclo 4. Finalmente, el bloque s2mm_write escribe el resumen resultante en la memoria externa a partir del ciclo 5. Esta secuencia escalonada valida la ausencia de conflictos de datos y confirma la latencia determinista del diseño [28].

Figura 22

Análisis temporal de operaciones en el Schedule Viewer de Vitis HLS

Nota. Cronograma de ejecución que evidencia la secuencia de lectura (mm2s), procesamiento (sha256) y escritura (s2mm) a lo largo de los ciclos de reloj.

Al finalizar todas las fases de verificación, Vitis HLS exporta un bloque IP totalmente operativo. Este componente mantiene las interfaces AXI estandarizadas para su conexión inmediata en Vivado. El resultado es un módulo hardware que encapsula la complejidad criptográfica bajo una interfaz accesible. De este modo, garantizamos la portabilidad del acelerador y simplificamos su integración final en la arquitectura del SoC [28], [29].

Decisiones de diseño y criterios de selección de parámetros

Los definen como sistema de tres pasos. En el primer sistema, se describe el marco de trabajo de la lógica RTL y también la lógica de un sistema de tres pasos, se describe el marco técnico de un sistema de tres pasos. Definen un sistema de tres pasos. En el primer sistema, la lógica técnica describe los sistemas técnicos de tres pasos y también el marco técnico Describe el sistema de tres pasos.

El sistema estableció 100 MHz como frecuencia de operación, en calidad de referencia mundial. Esta opción garantiza la integridad de la temporización en el contexto de la lógica programable del Zynq-7010, evitando condiciones marginales de ruteo. Si bien elevar la frecuencia podría mejorar el rendimiento teórico, al mismo tiempo podría incrementar la propensión a fallos intermitentes por medio de la incorporación de varios motores.

El reloj en Vitis HLS fue configurado en función de cada acelerador. Los núcleos AES tienen como objetivo un periodo de 10 ns por la frecuencia del sistema. Por otro lado, el acelerador SHA-256 Custom fue sintetizado con un periodo de 20ns (50 MHz). Esta reducción ayuda a cerrar la temporización por el camino crítico de la estructura iterativa, sin afectar la estabilidad funcional.

El proceso de síntesis toma en cuenta una incertidumbre de reloj de hasta el 27%. Esta incertidumbre modela variaciones reales en el retardo debido al enrutamiento y las condiciones físicas del dispositivo. Incorporar incertidumbre resulta en que las estimaciones predictivas de

la síntesis sean más conservadoras y permite una alineación más estrecha entre las predicciones de la síntesis de alto nivel y el comportamiento real de la implementación final.

A todos los aceleradores se les puso un límite de hasta 4 MB en el tamaño del bloque de procesamiento. Este límite se fijó mediante constantes en el hardware y sirve para equilibrar la velocidad de transferencia con el uso de memoria. Usar bloques fijos permite que los datos fluyan. Además, esto mantiene la igualdad con los modelos de software de referencia.

El sistema de acceso a datos hace uso de interfaces AXI4−Master. Esto con el fin de que pueda leer directamente desde la memoria DDR. Este método hace el procesamiento de los bloques de uno en uno, y de esta forma evita el llenado excesivo de los buffers internos de la FPGA.

Aísla la selección de interfaces AXI del plano de control y el plano de datos. La configuración del acelerador, comprendiendo la longitud del mensaje más las direcciones base, es administrada mediante las interfaces AXI4-Lite. Las interfaces AXI4-Master se encargan de gestionar la transferencia masiva de datos. Esto es lo único que hacen estas interfaces. Esta división disminuye la congestión en el autobús y maximiza el rendimiento real del sistema.

Según la naturaleza de cada algoritmo, los mecanismos de sincronización fueron modificados. El algoritmo SHA-256 utiliza un sistema de polling gracias a que sus tiempos de ejecución son largos. AES-128 hace uso de interrupciones durante el cifrado. Esto es para liberar al procesador y optimizar la eficiencia general del CPU.

Un criterio esencial para la evaluación fue este: la utilización de librerías en lugar de código propio. Los núcleos verificados que disminuyen el tiempo de desarrollo están disponibles en las implementaciones que se fundamentan en la Biblioteca de Seguridad AMD Vitis. No obstante, las implementaciones a medida ofrecen un control absoluto sobre la administración de la memoria y el ajuste interno. Esta capacidad de adaptación permitió que el diseño se ajustara a las limitaciones particulares del Zynq-7010, facilitando la implementación de mejoras que no eran viables al emplear bibliotecas de carácter general.

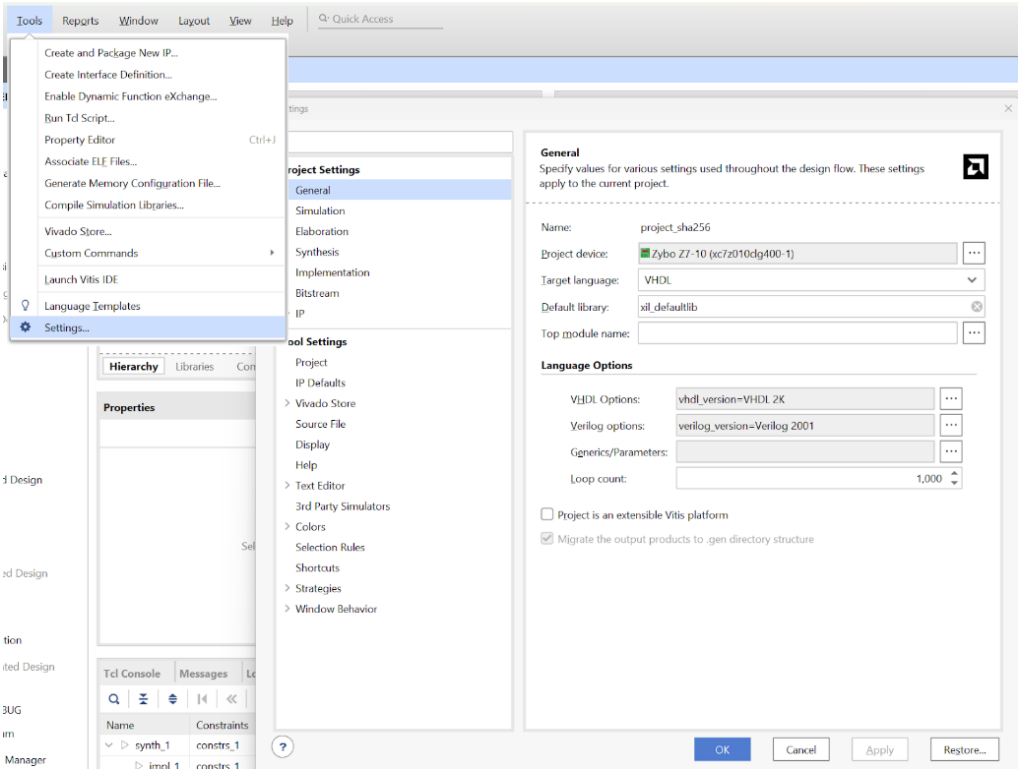
Por último, el diseño hace uso de un grado moderado de paralelismo por medio del empleo de pipeline y la división de registros. Si se hubiera aplicado una estrategia más ofensiva, el empleo de recursos lógicos habría aumentado y la finalización de la temporización habría estado amenazada. Por ende, con el objetivo de asegurar un equilibrio ideal entre la rapidez del procesamiento y la utilización del área en el chip, se restringió el paralelismo.

Integración en Vivado: configuración de IPs y conexión AXI

Integrarse en Vivado implica incluir la IP criptográfica creada en Vitis HLS en un sistema SoC que tenga como base Zynq, lo cual permite que la lógica programable (PL) y el procesador ARM del Sistema de Procesamiento (PS) se comuniquen a través de interfaces AXI. Esta etapa favorece la creación de un sistema operativo y el fortalecimiento del diseño de hardware a través de software embebido [34]. La metodología comienza con la puesta en marcha del proyecto en Vivado y la elección del dispositivo Zynq, proceso de configuración que se ilustra en la Figura 23, donde se selecciona el dispositivo de la familia Zynq-7000 compatible con la plataforma empleada en esta investigación. Esta disposición asegura la consistencia entre el procedimiento de implementación de la FPGA seleccionada y la IP sintetizada [35].

Figura 23

Configuración del proyecto en Vivado con selección del dispositivo objetivo Zynq-7000.

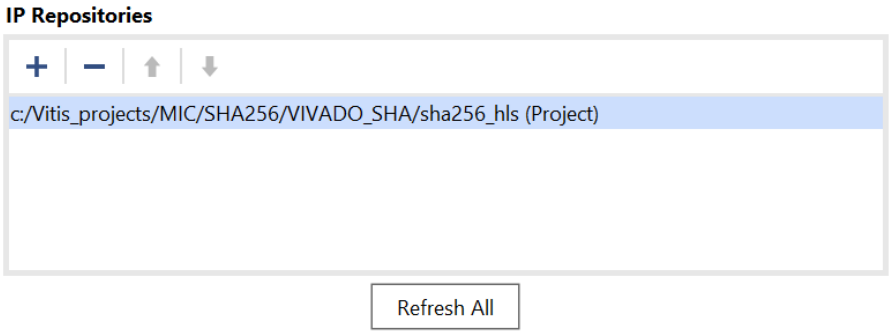


Nota. Se establecen parámetros iniciales de síntesis y compatibilidad del diseño.

Después, se registra el repositorio local que contiene la dirección IP empaquetada previamente en Vitis HLS, procedimiento que se ilustra en la Figura 24. Tras la actualización del catálogo interno de Vivado, la IP criptográfica se presenta como un bloque accesible y tiene la capacidad de ser implementada en el sistema como un componente estándar [36].

Figura 24

Adición del repositorio de IPs y registro de la IP SHA-256 generada en Vitis HLS.

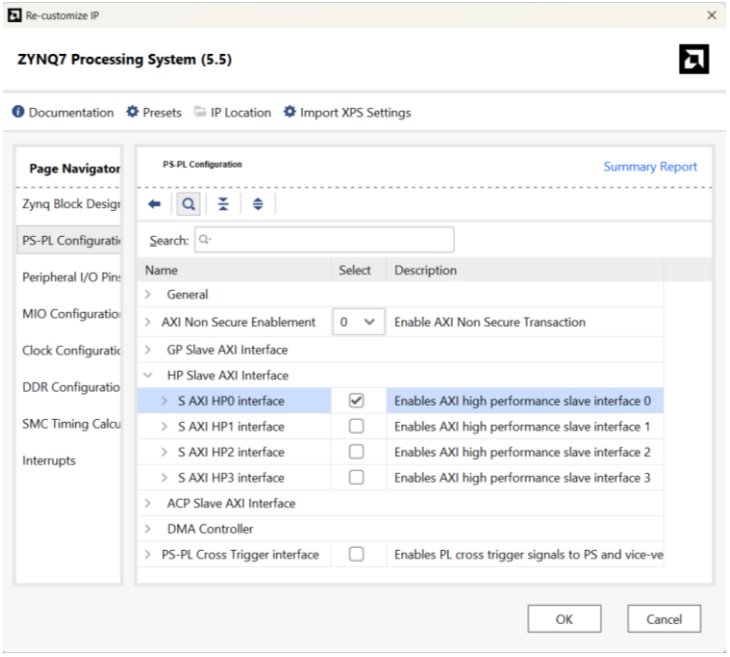


Nota. Vivado reconoce IPs personalizadas solo cuando el repositorio está correctamente declarado.

Después se desarrolla el Block Design, en el cual se establece el Zynq Processing System como nodo fundamental del sistema. La interfaz S_AXI_HP0 se habilita, como se muestra en la Figura 25, lo que posibilita que la IP tenga acceso directo a la memoria DDR del PS. Para los flujos criptográficos que necesitan procesamiento de datos ininterrumpido [37], [38], este enlace proporciona una velocidad de transferencia elevada y una latencia baja.

Figura 25

Configuración del Zynq PS con habilitación de la interfaz S_AXI_HP0 para acceso directo a DDR.



Nota. Una ruta de datos eficiente es crítica para maximizar el rendimiento del acelerador.

Asimismo, desde la plataforma de programación (PL) se permite gestionar interrupciones hacia el procesador, lo que simplifica el aviso al procesador cuando ha terminado de calcular el hash. Este método evita un esquema basado en el polling, mejora la utilización del procesador y hace más fácil que el sistema sea escalable [39].

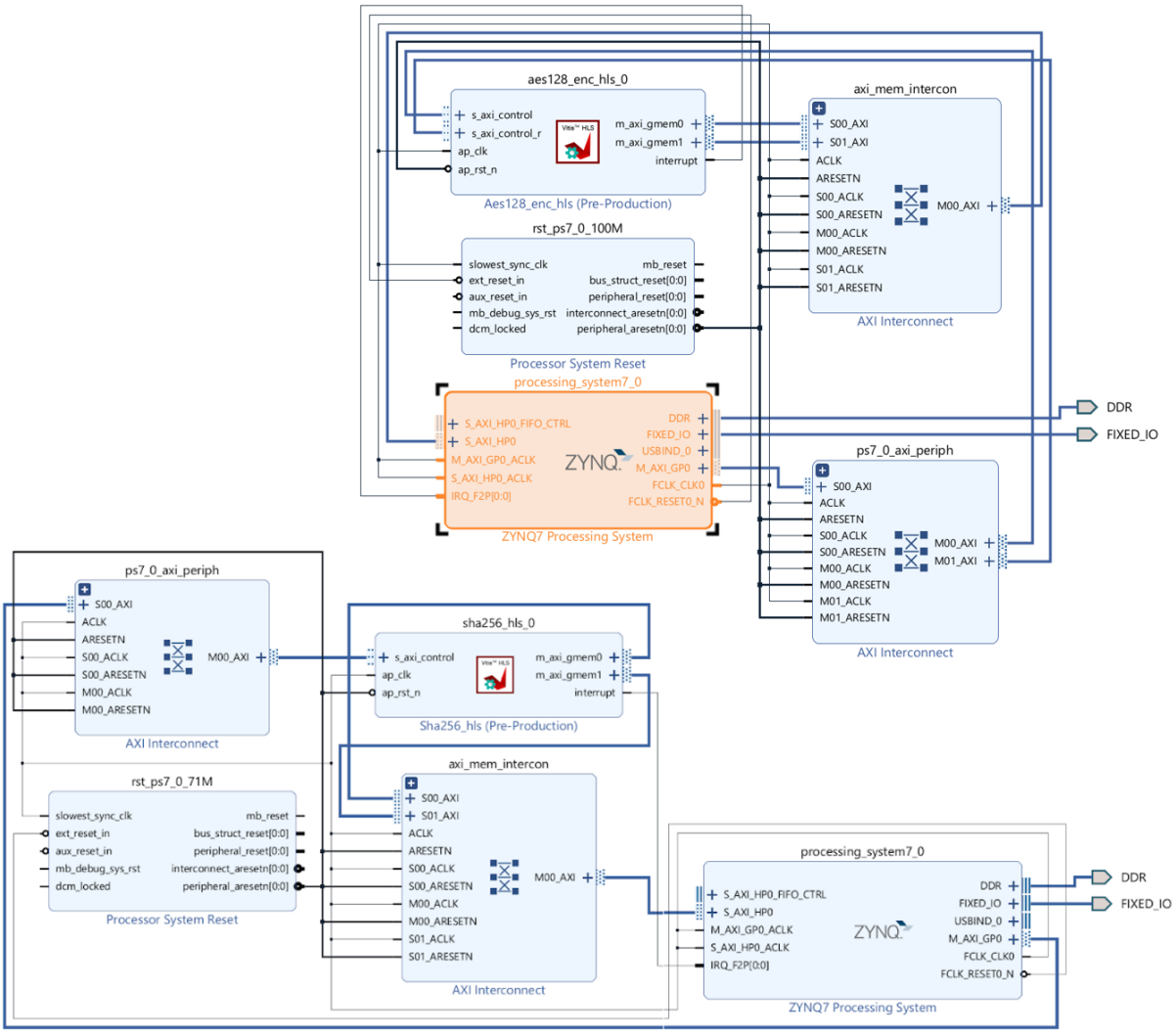
Con el procesamiento configurado, la IP se conecta mediante sus interfaces AXI. El bus AXI4-Lite se enlaza a los registros de control del PS para configurar parámetros como longitud del mensaje e inicio de operación, mientras que el bus AXI4-Master se conecta a la infraestructura de memoria para leer la entrada y escribir el digest generado. Se incorporan bloques auxiliares tales como AXI Interconnect y Processor System Reset con el objetivo de asegurar una distribución adecuada de relojes y señales [34], [35].

Si bien el flujo metodológico se ilustró inicialmente con el algoritmo SHA-256 para describir el proceso de síntesis en HLS, el estándar AES-128 siguió el mismo esquema general de diseño, integración y validación. Sin embargo, pese a que el procedimiento fue equivalente, cada núcleo presentó diferencias internas en su estructura así como profundidad de pipeline y utilización de recursos lógicos, debido a la distinta naturaleza de los algoritmos.

La Figura 26 muestra la inclusión simultánea de los dos aceleradores. Al presentar la información de esta forma, no solo se minimiza la redundancia en la descripción, sino que además se fortalece la contribución técnica del presente proyecto, que es mostrar que la misma arquitectura SoC-FPGA puede integrar diferentes funcionalidades optimizadas criptográficamente en un solo entorno de desarrollo.

Figura 26

Diagrama por bloques de los sistemas con la IP conectada al Zynq PS mediante buses AXI.

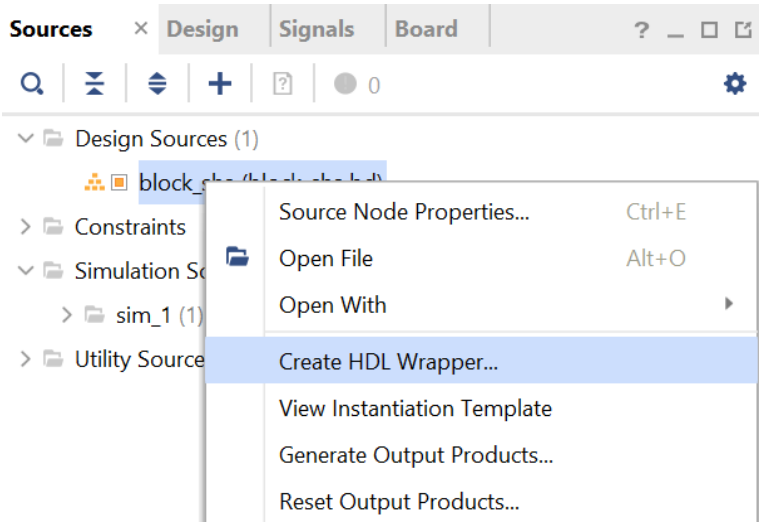


Nota. Ambas arquitecturas muestran la interfaz final exportada desde HLS hacia el entorno de Vivado IP Integrator y su arquitectura facilita el control por software y el acceso eficiente a memoria.

Se lleva a cabo la herramienta Validate Design, que examina si las conexiones son integras, si los anchos de bus son coherentes y si los resets y relojes se propagan correctamente. Una validación exitosa señala que el sistema está listo para la síntesis [34]. El paso posterior es crear el HDL Wrapper, procedimiento mostrado en la Figura 27, que envuelve la estructura gráfica del sistema en una entidad de nivel superior que se puede sintetizar [35].

Figura 27

Creación del HDL Wrapper como módulo superior para síntesis.



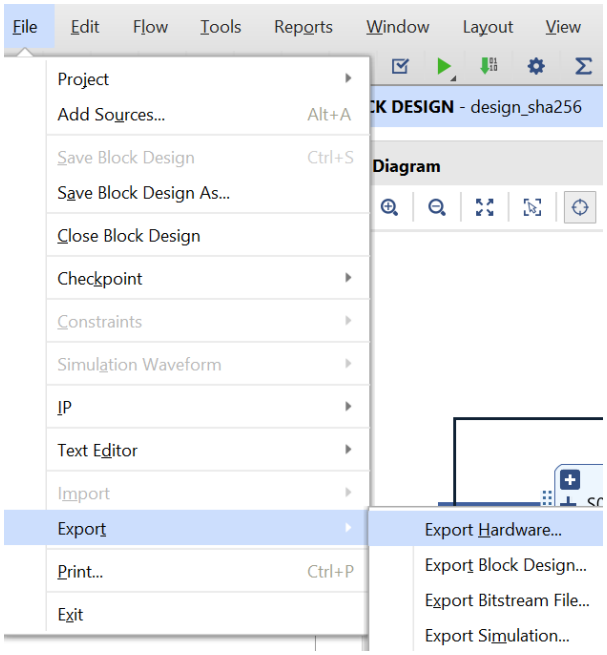
Nota. El wrapper habilita la etapa de implementación para el Block Design.

Finalmente se ejecutan las fases de síntesis e implementación, seguidas de la generación del bitstream, que configura la FPGA con la IP criptográfica y su integración dentro del sistema [37].

Para habilitar la etapa de software, se exporta el hardware como un archivo .xsa, incluyendo el bitstream, proceso que se presenta en la Figura 28. Este archivo contiene la descripción completa del sistema PS-PL y se emplea en la creación de la plataforma embebida para control del acelerador [39].

Figura 28

Exportación del hardware para integración con el flujo de software.



Nota. El archivo .xsa será utilizado en la sección de ejecución embebida del acelerador.

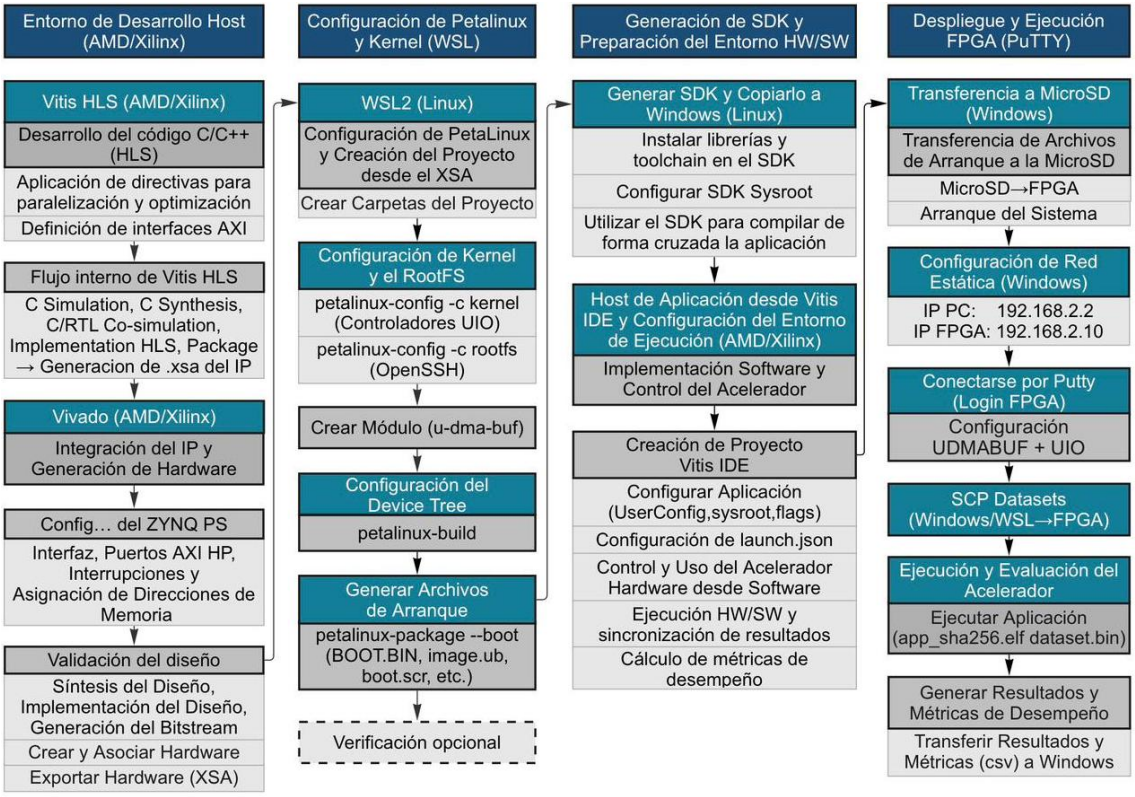
Con la exportación del .xsa, la integración hardware queda completada y el sistema está listo para su programación y validación desde Linux embebido.

Integración del sistema en la plataforma SoC.

La integración consiste en fusionar el diseño del hardware que se ha exportado de Vivado con el sistema operativo Linux embebido y la parte de software de la aplicación. Este procedimiento convierte los bloques IP, que son abstractos, en un sistema con la capacidad de realizar en tiempo real el procesamiento de datos mediante criptografía. La validación consiste en la sincronización del flujo de datos, desde el Host (Windows/WSL) al Target (Zynq-7000), garantizando la consistencia y correcto flujo de los datos. Esta es la razón, por la cual, en esta etapa se valida que la aceleración por hardware opere de manera óptima y controlada por el sistema operativo, como se aprecia en la Figura 29.

Figura 29

Flujo de integración del sistema hardware–software en la plataforma SoC Zynq-7000.



Nota. La figura resume el flujo de integración hardware–software en la plataforma Zynq-7000, desde el entorno Host (Windows/WSL2) hasta el despliegue y ejecución de la aplicación en la FPGA para la evaluación de desempeño.

Configuración del Entorno de Desarrollo Híbrido (Windows–WSL2)

El entorno de desarrollo híbrido empleado en esta investigación integra herramientas que requieren sistemas operativos distintos. Para resolver esta compatibilidad, se utilizó Windows Subsystem for Linux versión 2 (WSL2), una capa de virtualización ligera desarrollada por Microsoft que permite ejecutar un kernel Linux real dentro de Windows sin necesidad de una máquina virtual tradicional. WSL2 te da acceso directo a las herramientas de compilación, utilidades GNU y los entornos necesarios para trabajar con Petalinux y crear sysroot. Y al mismo tiempo, mantiene la conexión con Vivado y Vitis Unified IDE en Windows. Esta forma de trabajar baja el uso de recursos del sistema, mejora la velocidad de los archivos y hace fácil

que los entornos funcionen juntos. Por eso, permite tener un proceso de desarrollo claro para crear y probar el sistema SoC-FPGA [40].

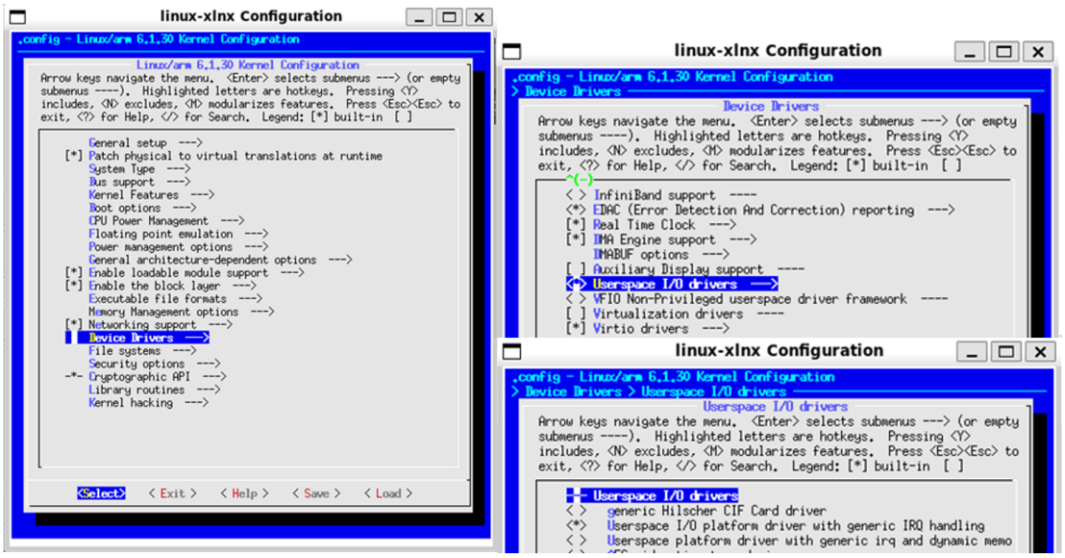
Se establece una estructura de trabajo que integra diferentes espacios para gestionar el flujo del diseño. Para el desarrollo en Windows Vitis IDE, y para la construcción del kernel de Linux, utilizamos el subsistema WSL2. Todo inicia en la etapa de exportación del archivo de descripción de hardware (.xsa) desde Vivado al entorno Linux. La crítica de esta transferencia, es que el sistema operativo debe saber la dirección física de los periféricos. Así, garantizamos que el software reconozca los mapas de memoria que el hardware tiene definidos.

Generación del sistema operativo y artefactos de arranque

El sistema embebido se crea a partir de la herramienta PetaLinux, y su estructura se fundamenta en el archivo .xsa. En esta etapa, sustituimos el servidor Dropbear, que es ligero, por OpenSSH completo y configuramos el RootFS. Asimismo, habilitamos el módulo de memoria contigua (u-dma-buf) y los controladores de espacio del usuario (UIO) en el núcleo, configuración que se muestra en la Figura 30. Para que la aplicación pueda acceder a los registros físicos de la FPGA sin limitaciones, estas configuraciones son imprescindibles. Como resultado, el sistema operativo está listo para administrar transferencias de datos con un alto rendimiento y que sean seguras.

Figura 30

Configuración del kernel para habilitar controladores UIO en PetaLinux



Nota. Activación del soporte de acceso desde espacio de usuario para periféricos de plataforma mediante manejo genérico de IRQ.

Luego de concluir la configuración, se realiza el empaquetado del sistema usando petalinux-package para así crear los archivos que son necesarios para el arranque. En este paso se juntan cosas como el bitstream de la FPGA, el gestor de arranque (FSBL/U-Boot) más el kernel, y todo esto se convierte en unos archivos binarios clave llamados BOOT.BIN, image.ub y boot.scr. Luego, hay que copiar estos archivos en la partición FAT32. Esto permitirá que la placa de la tarjeta microSD se encienda correctamente. Al hacer esto, la tarjeta Zynq inicia con un Linux personalizado con los aceleradores de hardware listos para usarse.

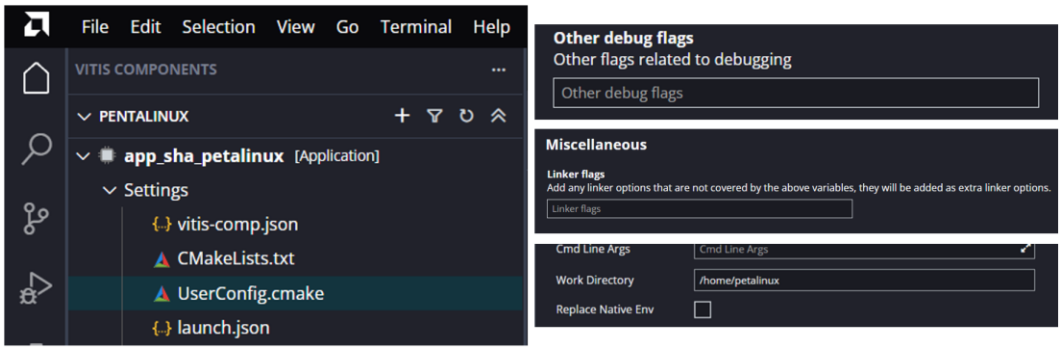
Desarrollo de aplicaciones en vitis ide y sysroot

El desarrollo de la aplicación de control se realiza en Vitis IDE, utilizando el kit de desarrollo de software (SDK) exportado desde PetaLinux. Para asegurar la compatibilidad binaria, vinculamos el sysroot del SDK en la configuración del compilador CMake, como se muestra en la Figura 31. Específicamente, aplicamos banderas como --sysroot y -mfloat-abi=hard para alinear la arquitectura de coma flotante con el procesador ARM Cortex-A9. Este paso es muy importante porque: sincroniza la ejecución, lo que elimina cualquier error debido a la biblioteca cruzada, hace que el binario compilado sea ejecutable en el objetivo.

Finalmente, configuramos nuestro lanzamiento dando una definición explícita del directorio de trabajo remoto en el archivo launch.json. Esta es una configuración importante ya que sirve para alinear el entorno de desarrollo con el sistema de archivos del objetivo. Como resultado, se puede lograr una ejecución remota y nunca surgirán problema con las rutas relativas incorrectas en el sistema destino debido a eso.

Figura 31

Configuración de CMake y directorio de trabajo en Vitis para compilación y ejecución remota



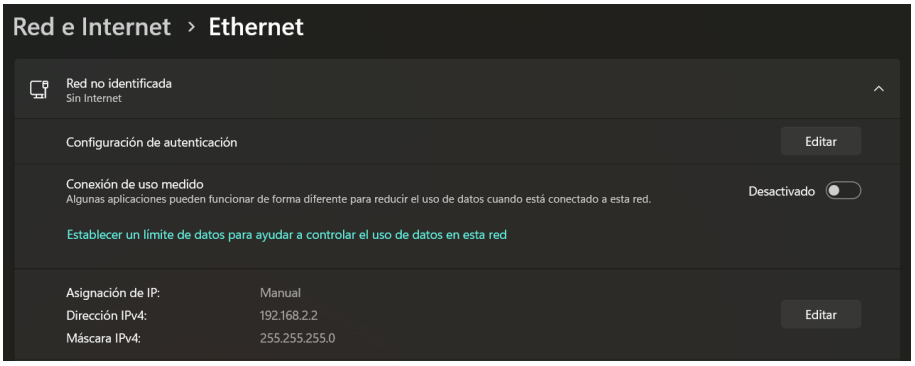
Nota. Parámetros de sysroot y opciones de arquitectura en “Other Debug Flags” y “Miscellaneous”, junto con la definición del directorio de trabajo remoto para la ejecución en el SoC.

Despliegue, conectividad y configuración en tiempo de ejecución

Para su implementación, unimos el SoC con la computadora directamente por medio de Ethernet, estableciendo una configuración de red estática como se muestra en la Figura 32. Con el objetivo de asegurar una conexión permanente tras los reinicios, asignamos direcciones IP fijas a las dos interfaces. Con esta configuración, es posible utilizar los protocolos SSH y SCP para transferir conjuntos de datos y gestionar el sistema a distancia. De esta manera, hacemos que las iteraciones de prueba sean más rápidas y eliminamos la dependencia de infraestructuras externas de red.

Figura 32

Configuración de red estática para comunicación PC–FPGA mediante Ethernet

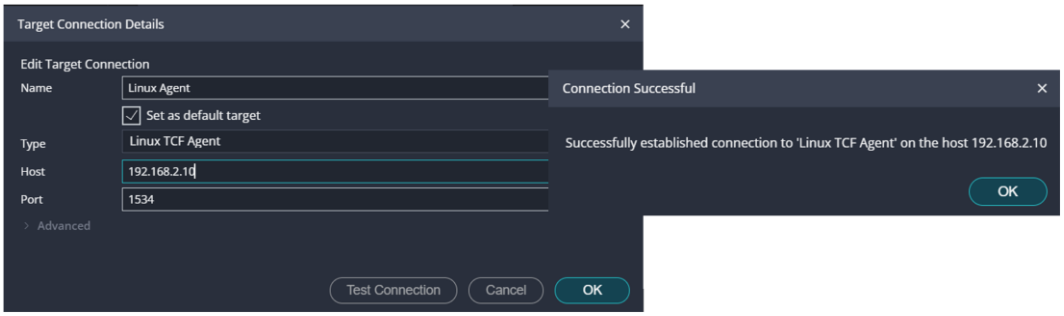


Nota. Asignación de IP en la PC y en la plataforma embebida para habilitar acceso remoto y transferencia de archivos.

Antes de iniciar la aplicación, hay que revisar la conexión con la tarjeta. Vitis mira el "Target" para confirmar que el servicio TCF está funcionando, como sale en la Figura 33. Esto sirve para estar seguros de que el canal de depuración está listo para cargar el archivo .elf. Por lo tanto, evitamos fallos de segmentación derivados de una transferencia incompleta de binarios.

Figura 33

Verificación de conectividad del target en Vitis mediante Target Connection Details



Nota. Confirmación de conexión activa hacia el sistema embebido para habilitar despliegue y ejecución de la aplicación.

Atendiendo a la actividad, es necesaria la reconfiguración "on-the-fly" de los controladores mediante los llamados scripts de inicio. El módulo u-dma-buf se carga y los dispositivos UIO se asignan manualmente a las direcciones base de los aceleradores criptográficos (unmapping y mapping). En este paso se otorgan permisos rw (chmod 666) a la

memoria compartida del espacio de usuario. Ahora, la plataforma está lista para procesar los vectores de prueba y evaluar el rendimiento de la plataforma de hardware bajo condiciones controladas.

Alcance del diseño y contribución técnica

Este trabajo usa la AMD Vitis Security Library, que funciona como un grupo de núcleos criptográficos que ya han sido probados. El fin es que sea más sencillo poner los algoritmos en la lógica programable. Estas librerías traen versiones de AES y SHA que ya funcionan y están listas para usarse con Vitis HLS. De este modo, la librería sirve como una pieza segura que permite dedicarse a integrar el sistema, sin tener que preocuparse por si la matemática del cifrado está bien o mal.

Pero la librería solo funciona para la parte de criptografía. No sirve para diseñar todo el sistema. La librería no explica cómo se hablan el procesador y la FPGA, y tampoco mueve los datos a la memoria externa. Tampoco ayuda con el control de DMA, la configuración de Linux o las formas de hacer pruebas. Todo el trabajo de juntar y organizar estas partes lo tiene que hacer la persona que diseña el sistema.

Lo más importante de este trabajo técnico es un sistema completo de SoC-FPGA que tiene aceleradores de seguridad funcionando en un lugar real. Esta idea junta todo el proceso de inicio a fin y conecta el programa de Python con el hardware. Para que funcione bien, usamos dispositivos UIO y espacios de memoria seguidos dentro del sistema.

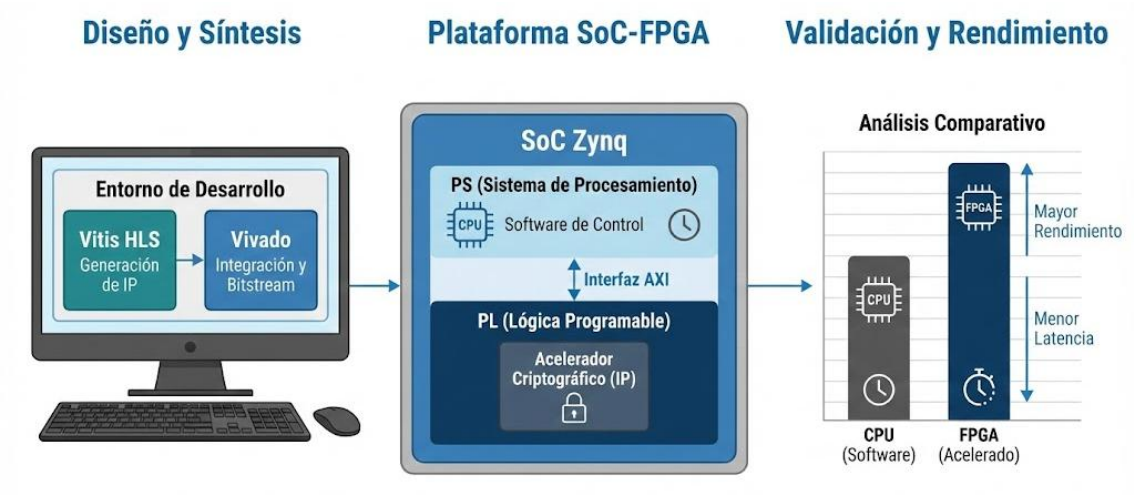
En la parte de las pruebas, el trabajo da una forma de medir los resultados que otras personas pueden volver a hacer. Este procedimiento permite la comparación imparcial de las implementaciones de hardware y software bajo condiciones controladas. Se definieron variables estables, como la frecuencia de operación y el tamaño del bloque. Asimismo, se incorporaron indicadores clave como la latencia y el throughput. Este enfoque comprueba los desarrollos individuales y las bibliotecas en cuanto a las limitaciones reales del dispositivo Zynq-7010.

Capítulo IV: Pruebas de Desempeño

Para garantizar la coherencia experimental a lo largo de este capítulo y el posterior análisis comparativo, todas las pruebas realizadas, tanto en software como en hardware, se rigen por las mismas métricas de rendimiento, siguiendo el flujo metodológico integral resumido en la Figura 34. La variable principal es el Tiempo End-to-End, el cual comprende el ciclo completo desde la carga del archivo de entrada hasta la obtención del resultado final, incluyendo los costos de lectura/escritura (I/O) y los cambios de contexto del sistema operativo. A partir de este valor se calcula el Throughput End-to-End (expresado en MB/s). Asimismo, se miden los tiempos netos de cómputo (Tiempo de CPU o Tiempo de Lógica Programable) para aislar el procesamiento del algoritmo de los factores sistémicos de transferencia y control.

Figura 34

Flujo metodológico integral de la implementación y validación en el SoC Zynq.



Nota. El diagrama resume las etapas del proyecto: Diseño y Síntesis, Integración en la arquitectura híbrida SoC Zynq y Validación final mediante un análisis comparativo.

Configuración Experimental y Métricas de Rendimiento

Las pruebas de desempeño se realizaron bajo un esquema experimental controlado con el fin de garantizar coherencia y reproducibilidad. Se evaluaron dos entornos de ejecución: una implementación puramente software ejecutada en un sistema operativo Linux sobre procesador

multinúcleo, y una implementación acelerada en una plataforma SoC-FPGA Zynq-7000 con Linux embebido. Los dos sistemas usaron exactamente los mismos archivos de entrada. Por eso, las diferencias que vemos son solo por la forma en que funciona cada sistema y no por los archivos.

Para medir la rapidez usamos reglas claras. La regla principal es el Tiempo End-to-End, que es el tiempo total que pasa desde que se carga el archivo hasta que sale el resultado final. Este valor incluye leer y escribir en la memoria, mover los datos de un lugar a otro y usar el código de seguridad. Su finalidad es reflejar el comportamiento real del sistema completo desde el punto de vista del usuario final.

El rendimiento de extremo a extremo consiste en el tiempo de extremo a extremo, convertido a MBps. Esta métrica refleja la capacidad real de procesamiento de datos efectiva del sistema, y consiste en procesamiento continuo, carga y el sistema que calcula el conjunto de datos. El rendimiento, por otro lado, permite medir la capacidad operacional del sistema, que, a diferencia de una simple medición del tiempo, es mucho más importante y útil.

Medir los tiempos netos de cómputo que se menciona a continuación, permite aislar el procesamiento del algoritmo y el resto de los pasos del mismo. En el caso del software, corresponde al tiempo efectivo de CPU dedicado a la función criptográfica. En el caso del hardware, corresponde este al tiempo de ejecución en el interior de la lógica programable. Esta separación permite distinguir entre eficiencia algorítmica, así como sobrecarga del sistema operativo o del subsistema de memoria.

Cada prueba se ejecutó múltiples veces, registrándose el valor promedio a fin de reducir la variabilidad asociada a procesos en segundo plano y efectos transitorios del sistema. En la plataforma SoC, el tiempo total se midió desde espacio de usuario en Linux. Además, el tiempo interno del acelerador se obtuvo mediante mecanismos de temporización disponibles en el sistema. Esta metodología permite poder diferenciar de manera clara entre el rendimiento arquitectónico y los costos de integración.

El uso conjunto de tiempo total, además, throughput da una visión integrada del sistema. El tiempo de cómputo aislado permite estudiar la eficacia estructural del acelerador implementado. En cambio, el tiempo End-to-End reúne el comportamiento real del sistema total. Esta distinción es primordial. Hecha esta distinción, se pueden entender correctamente los resultados que se presentan en las siguientes secciones.

Descripción de los datasets de prueba

El diseño de las pruebas usa grupos de datos fijos para que los resultados se puedan repetir y controlar bien. Elegir estos datos es muy importante, porque el trabajo de los códigos de seguridad depende de qué tipo son y qué tamaño tienen. Por eso, este trabajo usa dos tipos de datos: los creados por computadora y los reales.

Los conjuntos de pruebas más relevantes utilizan datasets sintéticos controlados. Estos conjuntos de pruebas contienen flujos de datos de propósito general que son diseñados específicamente para exponer los límites de la carga de trabajo de procesamiento criptográfico y son carentes de significado. El tamaño de estos archivos va desde kilobytes hasta cientos de megabytes, en potencias de dos. Esto permite estudiar, de forma sistemática, la escalabilidad del sistema a más nivel de carga.

Unos programas hechos en Python crearon estos grupos de datos. Esta forma de trabajar permite controlar el tamaño de los archivos y ayuda a que las pruebas se puedan repetir siempre igual. Este plan hace que no necesitemos datos de otros lugares y evita errores por usar datos con formas raras. También, usar tamaños iguales ayuda a comparar de forma fácil los diferentes sistemas y equipos.

La forma de medir los resultados junta la importancia de los archivos reales con la exactitud de los datos creados por computadora. Los primeros se generan a través de comandos en Python, lo que garantiza un control total sobre las dimensiones exactas de cada archivo. Esta metodología asegura que los experimentos se lleven a cabo de manera rigurosa y que los sesgos estadísticos provenientes de fuentes externas sean eliminados. Asimismo, con

la finalidad de comprobar el sistema en situaciones reales, incluimos documentos PDF e imágenes vinculados con las telecomunicaciones. Usar estos dos tipos de datos enseña que la rapidez del sistema sigue igual. Esto pasa no solo en las pruebas de laboratorio, sino también aunque los datos reales cambien de forma natural [41].

La forma de hacer las pruebas usa los dos tipos de datos por igual para proteger, recuperar y asegurar la información. Tanto el programa como el equipo usan exactamente los mismos archivos. Esto sirve para asegurar que las diferencias de rapidez son solo por cómo está hecho el sistema y cómo funciona, y no por cambios en los archivos.

Ejecución de pruebas con datasets sintéticos y reales

Evaluamos la integración del sistema en condiciones reales y en condiciones de datos sintéticos. Esta fase confirma que el flujo de transferencia, el procesamiento criptográfico y la recuperación de resultados se mantiene estable en varias ejecuciones seguidas. Estas pruebas nos permiten recoger las métricas de rendimiento necesarias para hacer la comparación.

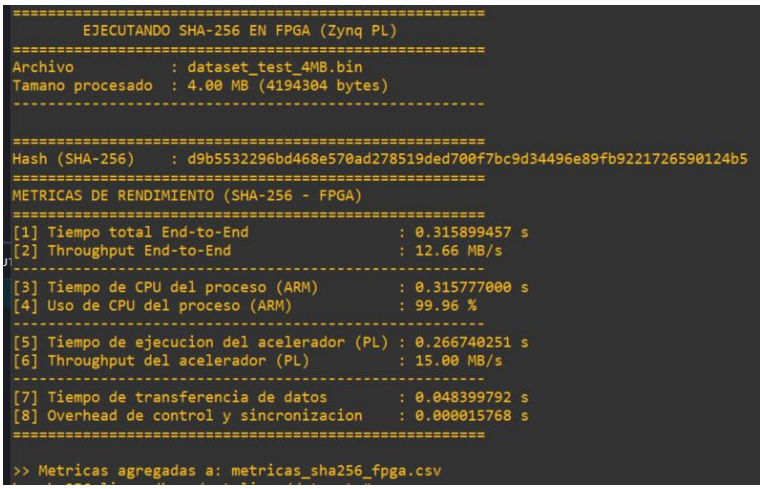
Desde el entorno de desarrollo del computador se organizan los conjuntos de datos y luego se transfieren a la plataforma embebida con herramientas de copia segura. Esta metodología permite automatizar la transferencia de archivos de prueba a una carpeta de trabajo ya determinada en el sistema Linux embebido, lo que ayuda a mantener la rastreabilidad de los tamaños experimentados y fomenta la repetición de las pruebas. La transmisión por medio de la red hace más fácil la obtención de los archivos de salida que el sistema produce, como los informes de métricas y resultados cifrados producidos durante su funcionamiento.

La aplicación de usuario se ejecuta una vez que se han detectado los archivos de entrada en la plataforma y se ha configurado la manera en la que va a trabajar el acelerador de procesamiento. A continuación, el sistema aplica cifrado IP para manipular la información de entrada y analizar cómo funciona el hardware. Aquí se prueban aspectos esenciales del entorno embebido, como el acceso a las áreas de memoria compartida y la interpretación

correcta del conjunto de datos. A su vez, la salida por consola nos muestra qué tan lejos vamos, ya que imprime la aceleración de respuesta. Finalmente, esta documentación continua verifica que todas las pruebas se hayan realizado correctamente y da la información para mejorar y certificar el sistema como se puede ver en la figura 35.

Figura 35

Ejecución de la aplicación y visualización de resultados en consola de la plataforma



Nota. Ejecución de salida que valida la lectura del dataset, el procesamiento criptográfico y la conclusión del flujo.

La funcionalidad del sistema se verifica asegurando que cada resultado se adhiera a cada una de las normas de operación establecidas para cada uno de los algoritmos. En el caso del cifrado AES-128, garantizamos que el archivo de salida conserve la misma estructura y tamaño que el documento original. En el caso de SHA-256, comparamos el digest obtenido con una referencia conocida o con un cálculo de control. De esta manera, se evidencia que la rapidez de nuestro sistema de hardware, además, no compromete la seguridad que las redes de comunicaciones actuales requieren.

Con los datos reales, la prueba fue más allá de comparar archivos como muestra la figura 36 y también hicimos pruebas de uso. Revisamos que los archivos PDF se pudieran abrir bien en programas normales y que las fotos se vieran perfectas, sin ningún fallo. Esta

comprobación confirma que el algoritmo implementado en hardware y software manipula correctamente la carga útil de los archivos sin corromper su formato, demostrando la viabilidad del sistema para proteger tráfico de datos verídico en un entorno de producción.

Figura 36

Verificación de resultados en el entorno de la PC a partir de archivos exportados desde la FPGA – AES 128 ENC

Nota. La verificación del resultado criptográfico se realizará mediante la comparación con una referencia externa.

Usamos el tiempo de las pruebas para guardar datos sobre qué tan rápido funciona el sistema al mismo tiempo que revisamos que tan bien trabaja. Medimos el tiempo que tarda en trabajar, el retraso en cada operación y la cantidad de datos que procesa por segundo para cada tamaño de archivo. El sistema guarda estos números en su propia memoria y luego los manda a la computadora principal para juntarlos todos rápido. Esta lista de datos sirve para comparar con números reales cuánto más rápido es el equipo especial de hardware frente al programa normal de la computadora. Así, garantizamos la repetibilidad de los experimentos manteniendo una separación estricta entre la recolección de métricas y su análisis posterior.

Desempeño de las implementaciones software (Python)

Esta sección evalúa la capacidad de las implementaciones para cifrar y descifrar conjuntos de datos en un escenario de práctica del mundo real. Con este propósito, se ejecutan

operaciones completas en archivos de diferentes tamaños y se miden los tiempos de ejecución, el rendimiento y el uso de CPU. Los resultados se presentan en tablas comparativas que sirven para evaluar el rendimiento del sistema y compararlo con las implementaciones aceleradas en hardware.

Las pruebas de rendimiento iniciales se hacen solo en el procesador. Esto se hace para tener una base y poder comparar después qué tan rápidas son las versiones que usan hardware especial. La computadora de prueba es una laptop muy potente que tiene un procesador Intel i9 de generación 12. Este procesador funciona entre 2.5 y 5 GHz. También, el equipo tiene 32 GB de memoria RAM y un disco SSD de 1 TB para guardar la información.

Para medir el trabajo de forma exacta, los programas de AES-128 y SHA-256 se hicieron en Python y se usaron directamente en el sistema operativo principal. Anotamos los tiempos que explicamos al inicio del capítulo con relojes muy exactos. Esto sirve para saber si el tiempo se gasta en el cálculo del código o en la forma en que la computadora mueve los datos.

La Tabla 1 muestran el rendimiento de AES-128 en la CPU para operaciones de cifrado y descifrado. En los cuatro parámetros medidos, el tiempo de extremo a extremo crece aproximadamente de manera lineal con el aumento del tamaño del archivo. El rendimiento, sin embargo, tiende a aumentar y estabilizarse cuando el costo fijo de inicialización y de E/S se amortiza con cargas de trabajo mayores. El tiempo de CPU también crece con el aumento del tamaño del archivo, y el uso de CPU de archivos grandes aumenta, lo que indica que el procesamiento criptográfico comienza a dominar sobre otras tareas del sistema. En tamaños mucho más pequeños, el rendimiento efectivo es menor, ya que la sobrecarga del sistema operativo y el acceso al disco se llevan una mayor parte del tiempo.

De la misma manera en que medimos los tiempos, también verificamos los datos reales y confirmamos que, después de la encriptación y desencriptación del software, tanto los archivos de prueba como los reales mantuvieron su estructura y contenido exactamente bit por

bit, lo que muestra que los scripts de Python funcionan correctamente antes de hacer la comparación con el hardware.

Tabla 1

Desempeño del algoritmo AES-128 en implementación software (Python).

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
1 MB	27.02	0.037	0.002	0.00
2 MB	44.53	0.044	0.007	0.00
4 MB	79.65	0.050	0.015	31.11
8 MB	140.96	0.056	0.031	55.06
16 MB	188.97	0.084	0.015	18.45
32 MB	329.62	0.097	0.046	48.28
44 MB	235.23	0.187	0.125	66.75
64 MB	606.89	0.105	0.062	59.27
128 MB	755.15	0.169	0.125	73.75
256 MB	824.31	0.310	0.265	85.53

Nota. Las métricas que se muestran a continuación son a través de End-to-End, tiempo de ejecución, uso de CPU, y se presentan para diferentes tamaños de archivo. Los valores presentados corresponden a la operación de cifrado, ya que se omiten los resultados de descifrado al considerar que el algoritmo presentó simetría con una desviación menor al 2% en el rendimiento a los datos presentados.

El rendimiento de SHA-256 en CPUs se muestra en la Tabla 2 utilizando el mismo enfoque de medición. En el tiempo de extremo a extremo, el volumen de datos es directamente proporcional al tiempo tomado, lo que configura la naturaleza iterativa del procesamiento del algoritmo hash que se realiza de manera secuencial dependiendo del estado anterior. En consecuencia, el uso de la CPU tiende a mantenerse alto para tamaños medianos a grandes.

Para tamaños de entrada muy pequeños, algunas mediciones asociadas con el tiempo de CPU y el uso de CPU se acercan a cero, lo que se atribuye a la resolución temporal del sistema de medición y al peso relativo de la sobrecarga de tiempos extremadamente cortos, sin afectar la validez funcional del algoritmo.

Tabla 2

Desempeño de SHA-256 en implementación software (Python).

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
100 B	0.01	0.008	0	0
200 B	0.64	0.0002	0	0
300 B	1.3	0.0002	0	0
400 B	1.76	0.0002	0	0
500 B	1.44	0.0003	0	0
600 B	1.8	0.0003	0	0
700 B	2.03	0.0003	0	0
800 B	3.36	0.0002	0	0
900 B	2.69	0.0003	0	0
1 KB	2.98	0.0003	0	0
8 KB	24.98	0.0003	0	0
16 KB	45.74	0.0003	0	0
32 KB	80.07	0.0003	0	0
64 KB	151.15	0.0004	0	0
128 KB	382.26	0.0003	0	0
252 KB	383.5	0.0006	0	0
256 KB	648.34	0.0003	0	0
512 KB	655.65	0.0007	0	0

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
1 MB	621.66	0.0016	0	0
2 MB	706.86	0.0028	0	0
4 MB	747.86	0.0053	0	0

Nota. Métricas de desempeño para SHA-256 ejecutado en CPU. En archivos muy pequeños, el tiempo de CPU puede aproximarse a cero por resolución del sistema de medición, mientras que el comportamiento general mantiene la tendencia lineal esperada respecto al tamaño de entrada.

Los resultados indican que si bien la CPU logra altas tasas de procesamiento en tareas criptográficas, su desempeño en general se ve limitado por la ejecución de tareas de propósito general, la administración por el sistema operativo y el costo relativo de I/O, en particular en cargas pequeñas.

En esta parte se analiza la eficiencia de las implementaciones hardware para cifrar y descifrar bloques de datos reales en una SoC basada en FPGA. Las pruebas utilizan IPs criptográficas controladas desde el ARM y los resultados se muestran en tablas con la utilización de la CPU, el rendimiento y el tiempo de ejecución para distintos tamaños de entrada. Estos indicadores evalúan cómo el comportamiento que se observa se ve influenciado por la forma en que está diseñado el sistema.

Las pruebas de rendimiento en hardware se llevaron a cabo en un SoC basado en FPGA, donde los algoritmos criptográficos se ejecutan en la lógica programable y son controlados por el procesador embebido. La plataforma utiliza un procesador ARM junto con lógica programable interconectada mediante interfaces AXI. Estas últimas posibilitan que los datos fluyan desde la memoria DDR a los aceleradores. La sincronización de las transferencias y la inicialización de los núcleos introduce un overhead inherente al esquema.

En la tabla 3, examinamos dos enfoques: el uso de núcleos criptográficos proporcionados por las bibliotecas Vitis y las implementaciones del autor (Personalizadas) a través de síntesis de alto nivel (HLS). Para obtener resultados, se desarrolló una aplicación de análisis de consistencia metodológica en Python para garantizar la consistencia con las pruebas de software. En el caso del algoritmo AES-128, la versión basada en la biblioteca Vitis ofrece un rendimiento constante y predecible, y correlaciones casi lineales en el tiempo de ejecución y el tamaño del archivo procesado. El rendimiento tiende a estabilizarse con el aumento de los datos porque los costos del procesador de control son insignificantes en comparación con el tiempo de computación gastado en lógica programable.

Tabla 3

Desempeño de AES-128-Encrypt en implementación hardware basada en librerías Vitis.

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
1 MB	45.253	0.022	0.022	99.77
2 MB	45.913	0.043	0.043	99.94
4 MB	47.399	0.084	0.084	99.97
8 MB	51.206	0.156	0.156	99.98
16 MB	53.323	0.300	0.299	99.95
32 MB	54.379	0.588	0.588	99.95
44 MB	54.699	0.805	0.805	99.97
64 MB	55.023	1.163	1.162	99.97
128 MB	54.529	2.348	2.348	99.97
256 MB	54.073	4.735	4.733	99.97

Nota. La gestión de la transferencia de datos necesaria para la implementación estándar de Vitis conlleva un uso intenso de la CPU. Se omiten los resultados de descifrado al considerar

que el algoritmo presentó simetría con una desviación menor al 1.5% respecto a los valores mostrados.

Por su parte en la tabla 4 y 5, la implementación AES-128 “Custom” desarrollada por el autor exhibe un comportamiento caracterizado por una arquitectura optimizada para el flujo de datos. Al tener una gestión de memoria específica para el caso de estudio, se observa un escalamiento lineal eficiente en el throughput y una tendencia estable en los tiempos de ejecución a medida que aumentan los tamaños de entrada, reflejando una alta utilización de los recursos de la lógica programable.

Tabla 4

Desempeño de AES-128-Encrypt en implementación hardware Custom desarrollada.

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
1 MB	48.136	0.020	0.020	99.82
2 MB	49.769	0.040	0.040	99.93
4 MB	50.433	0.079	0.079	99.81
8 MB	54.899	0.145	0.145	99.95
16 MB	57.326	0.279	0.278	99.95
32 MB	58.583	0.546	0.546	99.97
44 MB	56.469	0.780	0.779	99.98
64 MB	59.256	1.080	1.079	99.96
128 MB	59.533	2.150	2.149	99.97
256 MB	58.717	4.359	4.358	99.97

Nota. Comportamiento y de rendimiento de la implementación de AES-128-Encrypt *Custom*.

Tabla 5

Desempeño de AES-128-Descrypt en implementación hardware Custom.

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
1 MB	45.206	0.022	0.022	99.89
2 MB	46.389	0.0432	0.043	99.94
4 MB	47.233	0.0848	0.084	99.97
8 MB	52.669	0.1520	0.151	99.96
16 MB	54.489	0.293	0.293	99.99
32 MB	56.666	0.564	0.564	99.96
44.05 MB	57.969	0.760	0.759	99.99
64 MB	58.163	1.100	1.100	99.97
128 MB	57.766	2.216	2.215	99.97
256 MB	57.129	4.482	4.481	99.97

Nota. Comportamiento y de rendimiento de la implementación de AES-128-Descrypt Custom.

La tabla 6 muestra el comportamiento del algoritmo SHA-256 en el FPGA que dependiendo de la implementación muestra diferentes perfiles. La versión de la biblioteca de Vitis muestra una limitación en el tamaño máximo de los archivos que pueden ser procesados, logrando una operación estable para tamaños más pequeños del orden de kilobytes. Para tamaños mayores, el sistema detiene la ejecución, un comportamiento relacionado con la memoria interna y las restricciones de control del núcleo IP estándar.

Tabla 6

Desempeño de SHA-256 en implementación hardware basada en librerías Vitis.

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
105 B	0.173	0.0005	0.0005	98.86
210 B	0.343	0.0005	0.0005	98.92
315 B	0.513	0.0005	0.0005	98.99

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
419 B	0.689	0.0005	0.0005	99.06
524 B	0.846	0.0005	0.0005	98.83
629 B	0.976	0.0005	0.0005	98.88
734 B	1.163	0.0005	0.0005	98.95
839 B	1.326	0.0005	0.0005	98.96
944 B	1.466	0.0005	0.0005	98.95
1 KB	1.569	0.0006	0.0006	100.05

Nota. La ejecución estable se limita a tamaños pequeños debido a restricciones internas de la arquitectura.

La implementación Custom de SHA-256, concebida a través de HLS, integra mecanismos explícitos de administración de memoria. Esta versión fue concebida con una limitación arquitectural explícita que permite el procesamiento estable de archivos de hasta 4 MB, cuyos resultados se presentan en la Tabla 7. En el procesamiento de archivos de tamaño considerable, el sistema calcula el hash exclusivamente sobre los primeros cuatro megabytes de datos. Esta conducta corrobora que la restricción se origina a partir de una decisión de diseño en la asignación de memoria local (BRAM), y no de una falla funcional del algoritmo.

Tabla 7

Desempeño de SHA-256 en implementación hardware Custom.

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
8 KB	5.449	0.001	0.001	98.70
16 KB	7.623	0.002	0.002	99.11
32 KB	9.516	0.003	0.003	99.42
64 KB	10.893	0.005	0.005	99.40

Tamaño	Throughput E2E MBs	Tiempo E2E Seg	Tiempo CPU Seg	Uso CPU %
128 KB	11.733	0.010	0.010	99.84
256 KB	12.299	0.020	0.020	99.91
512 KB	12.406	0.040	0.040	99.95
1 MB	12.55	0.079	0.079	99.93
2 MB	12.629	0.158	0.158	99.98
4 MB	12.669	0.315	0.315	99.96

Nota. En la implementación personalizada podemos procesar archivos grandes que se han diseñado con un máximo de 4 MB debido a limitaciones de hardware.

Preservamos la integridad de los datos reales en todas las pruebas, ya que, independientemente de las variaciones en el throughput o el tamaño de bloque, el sistema operativo pudo procesar los archivos recuperados desde la memoria de la FPGA. La aceleración por hardware no cambió la cantidad de trabajo ni los metadatos. Estos resultados demuestran que el diseño integral del sistema especialmente la gestión de memoria y el control del ARM define el rendimiento observable del acelerador. Así, establecemos una base técnica sólida para el análisis comparativo que abordaremos en el siguiente capítulo.

Capítulo V: Análisis de Resultados

Comparación software vs hardware

Esta sección confronta el rendimiento de las implementaciones de software frente a la aceleración por hardware en la ejecución de algoritmos criptográficos intensivos. Usamos las medidas del Capítulo 4 para ver cómo funcionan AES-128 y SHA-256 en tres casos: en el CPU, con Vitis y en nuestra FPGA. Nos fijamos más en el tiempo total y en la velocidad final. Y hacemos esto porque estos datos suman el tiempo de cálculo y el tiempo de mover la información.

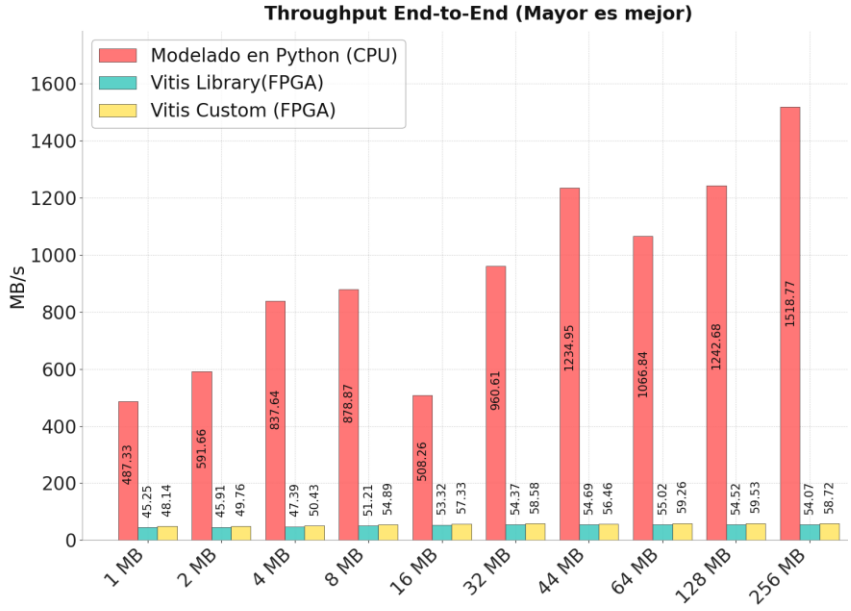
Análisis de AES-128 (Encriptado)

Investigar el cifrado AES-128 muestra el comportamiento típico de arquitecturas de propósito general. Para la implementación de software (CPU), el tiempo de ejecución aumenta linealmente con el aumento del tamaño del archivo. El primer crecimiento del throughput es aparente, la solicitud de estancarse es rápida por la saturación de los recursos del procesador. La explicación de este límite de rendimiento es la naturaleza secuencial de la CPU y la intervención del sistema operativo con interrupciones de la gestión de memoria. Estos factores son los que ocurren en los retrasos. Por lo tanto, la diferencia de la velocidad es evidente en los aumentos de la carga de trabajo.

La implementación en FPGA mejora de forma notable el tiempo de procesamiento, mejora que se acentúa al trabajar con archivos de tamaño medio o grande. En este sentido, conviene resaltar que en el caso que nos ocupa, la arquitectura Custom mejora el throughput que la propuesta de Vitis. Esto valida que un diseño a medida optimiza el aprovechamiento de los recursos lógicos y la sobrecarga de control. A medida que los datos inflacionan, el sistema amortiza el coste fijo de inicialización del acelerador, y permite que la pura velocidad del hardware domine el rendimiento global. Estos datos son consolidados en las Figuras 37 y 38, que señalan que para cargas de trabajo importantes, la limitación de la CPU se torna irrelevante al hacer uso de la aceleración por hardware.

Figura 37

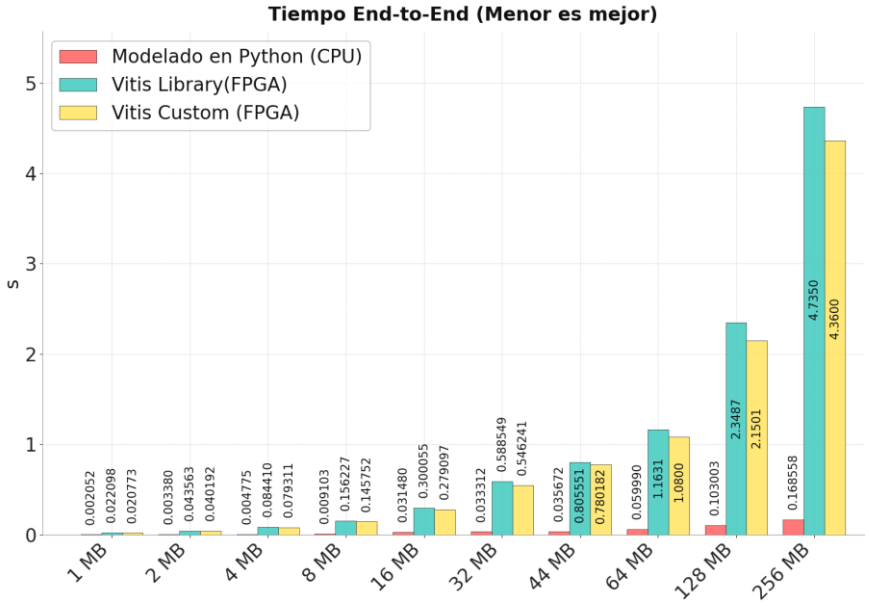
Comparación de throughput end-to-end para AES-128-Encriptado (software vs hardware).



Nota. Comparación del rendimiento global del sistema para AES-128 en modo encriptado, mostrando la diferencia entre la ejecución en CPU y en FPGA.

Figura 38

Comparación del tiempo end-to-end para AES-128-Encriptado (software vs hardware).



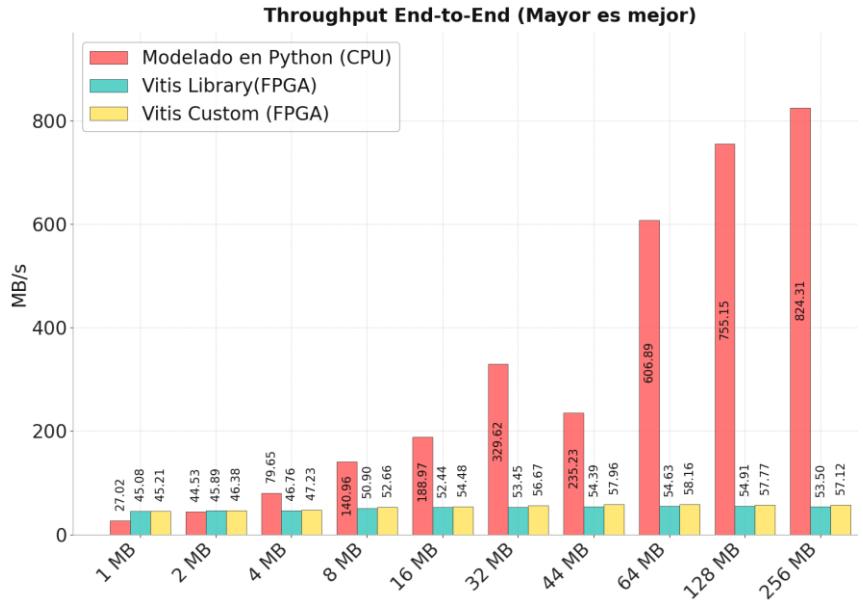
Nota. Tiempo total de ejecución del algoritmo AES-128 en modo encriptado, considerando todo el flujo de procesamiento.

Análisis de AES-128 (Desencriptado)

El modo de descifrado replica el patrón de rendimiento observado durante la encriptación. Mientras la ejecución por software choca con el techo físico impuesto por la frecuencia de la CPU, las implementaciones en FPGA escalan su capacidad de respuesta bajo carga. Sobresale nuevamente nuestra arquitectura Custom, que supera de forma consistente a la versión basada en bibliotecas Vitis. Este resultado de las figuras 39 y 40 validan la hipótesis de diseño: la optimización manual del flujo de datos y una gestión precisa de las interfaces AXI eliminan los ciclos de espera (idle cycles) inherentes a las soluciones genéricas, maximizando así el throughput efectivo.

Figura 39

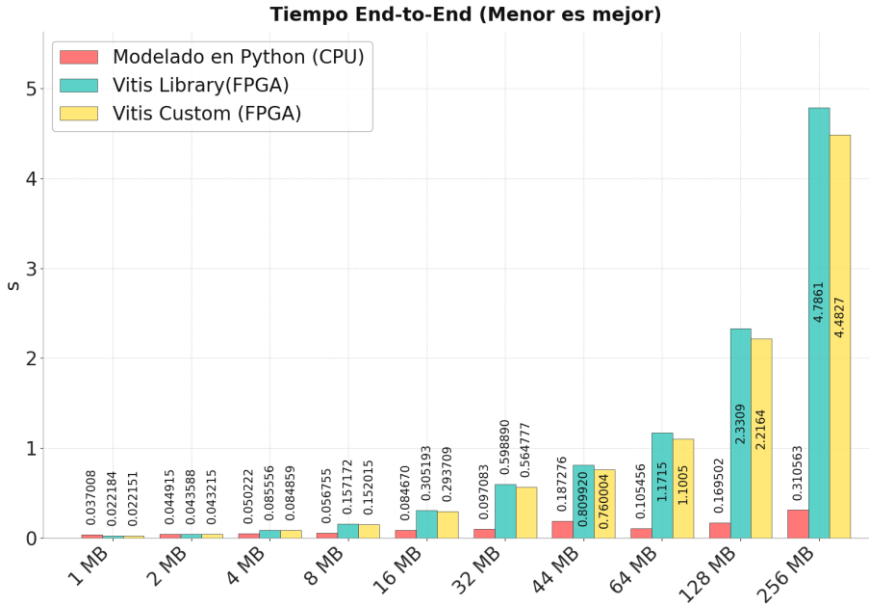
Comparación de throughput end-to-end para AES-128-Decrypt (software vs hardware).



Nota. Rendimiento global del sistema para AES-128 en modo desencriptado, comparando CPU y FPGA.

Figura 40

Comparación del tiempo end-to-end para AES-128-Decrypt (software vs hardware).



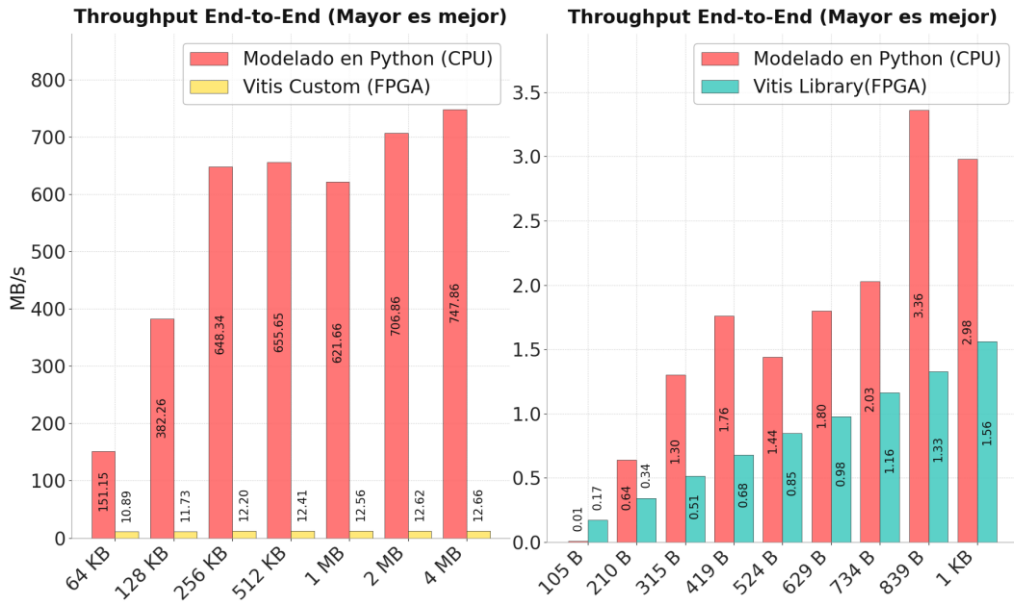
Nota. Tiempo total de ejecución del proceso de desencriptado AES-128 para las distintas plataformas.

Análisis de SHA-256

La validación de un algoritmo SHA-256 permite verificar la exactitud de la implementación matemática de algoritmos en un código. En la evaluación de la versión hardware de la Vitis, se pueden identificar ciertas restricciones. Por ejemplo, la versión Vitis tiene restricciones en el tamaño de archivos que puede manejar, lo que la descualifica para el manejo de cargas de trabajo pesadas.Nuestra implementación Custom evita este tipo de cuellos de botella mediante el uso de técnicas de control de memoria más sofisticadas. Este Diseño es capaz de procesar bloques de datos de hasta 4 MB de manera eficiente y de forma considerablemente menor a la latencia que presenta la CPU. La Aceleración personalizada, según las Figuras 41 y 42, supera la latencia de la biblioteca del Vitis y el coeficiente de la paralelización.

Figura 41

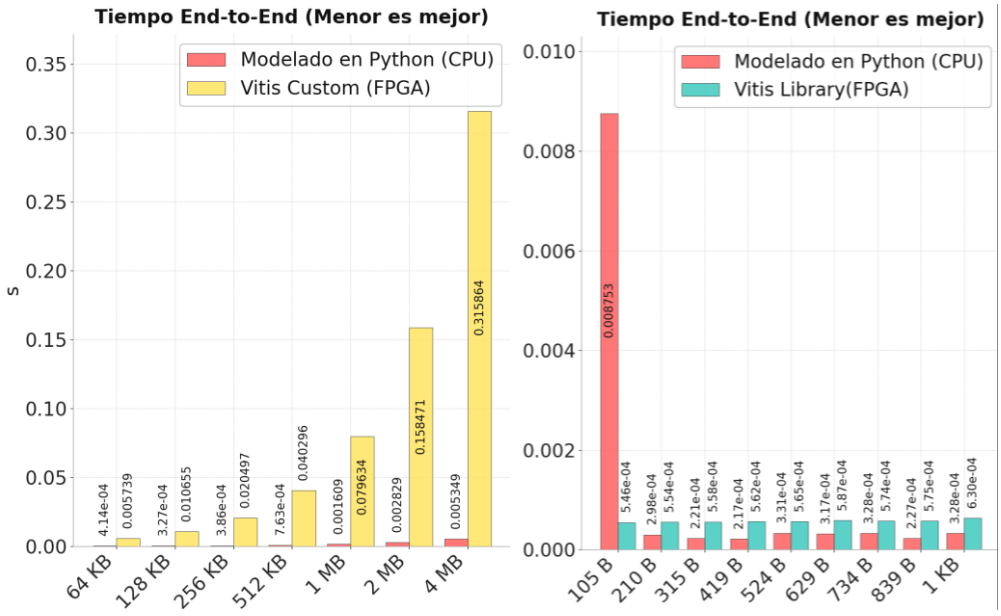
Comparación de throughput end-to-end para SHA-256 (software vs hardware).



Nota. Comparación del rendimiento del algoritmo SHA-256 en CPU y FPGA, destacando las diferencias de las implementaciones.

Figura 42

Comparación del tiempo end-to-end para SHA-256 (software vs hardware).



Nota. Tiempo total de ejecución del algoritmo SHA-256 para las distintas plataformas evaluadas.

El análisis muestra que por el tipo de aceleración que se use el hardware siempre tendrá mejor rendimiento que el software cuando el volumen de datos manejado es mediano o grande. De todas formas, se necesita un diseño eficiente para que el rendimiento bruto de cálculo sea suficiente. Comparamos dos alternativas, el uso de bibliotecas, que dan un desarrollo más rápido y el uso de implementaciones propias. Ajustar recursos lógicos a necesidades específicas permite eliminar sobrecargas de control y latencias asociadas a soluciones genéricas. Por lo tanto, sólo las Arquitecturas Personalizadas optimizan el rendimiento de extremo a extremo.

Análisis de cuellos de botella, sobrecarga computacional y throughput del sistema en FPGA

Como parte del ciclo de ejecución interno en la plataforma SoC-FPGA, se generado una lista de elementos que proporcionan información sobre las variaciones en el rendimiento general. El rendimiento final se ve afectado por la velocidad del núcleo criptográfico y la comunicación entre el procesador ARM y la lógica programable. El flujo de hardware incorpora pasos de control, transferencia de datos y sincronización que las soluciones puramente basadas en software no tienen, lo cual es perjudicial para el rendimiento de extremo a extremo (E2E) de todo el sistema. Las figuras 43 a 46 capturan métricas internas para SHA-256 y AES-128. Esta distinción nos permite separar el tiempo dedicado a la gestión del procesador principal del tiempo dedicado a la lógica programable.

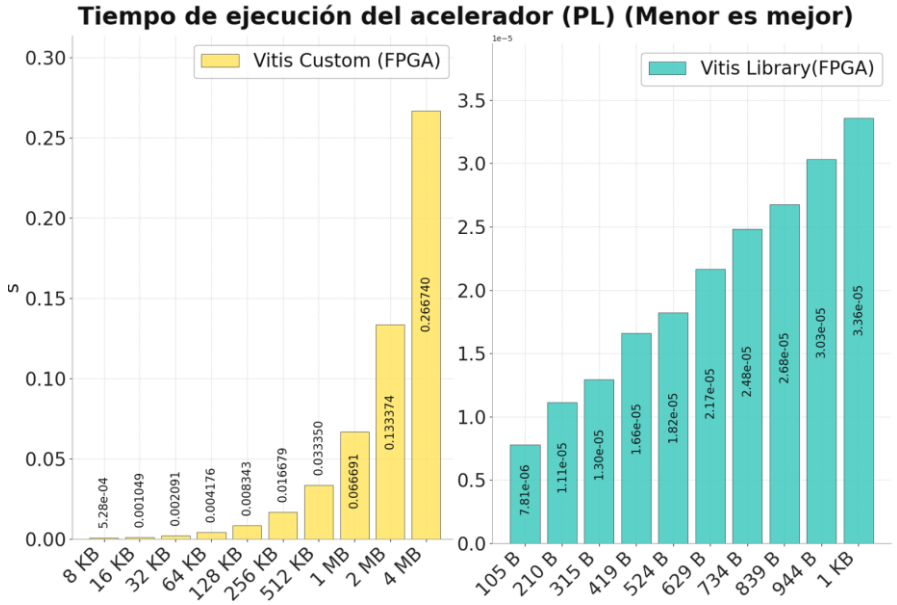
Se comprobó que la métrica interna, que aísla el tiempo de procesamiento puro de las latencias de la CPU, define la potencia máxima del núcleo. Se probó que el hardware realiza las operaciones a una alta velocidad, lo que conlleva a una alta diferencia en el rendimiento. Esta diferencia demuestra que el cuello de botella se encuentra en los mecanismos de comunicación y no en la capacidad de cómputo de la FPGA.

La gestión del procesamiento ARM calcula la eficiencia del ciclo de ejecución por archivos. En trabajos de tamaño reducido, los costes fijos de orquestación y ajustes de la CPU

consumen prácticamente todo el tiempo. En cambio, en el procesamiento de un volumen alto de datos, la lógica de cálculo programable se incrementa de forma proporcional y el cálculo desplaza el overhead de la CPU. Este comportamiento evidencia que el sistema amortiza el control de los procesadores en trabajos de gran tamaño, dejando a la FPGA operar casi a su máxima capacidad.

Figura 43

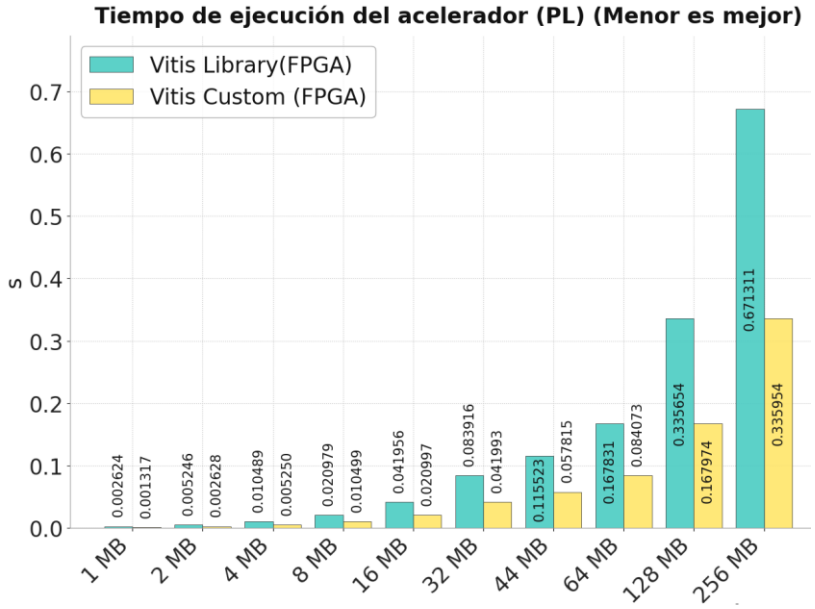
Tiempo de procesamiento puro en la lógica programable (PL) de SHA 256.



Nota. Tiempo de ejecución interno del núcleo de SHA 256 en la FPGA medido desde el inicio hasta la finalización del kernel, excluyendo transferencias.

Figura 44

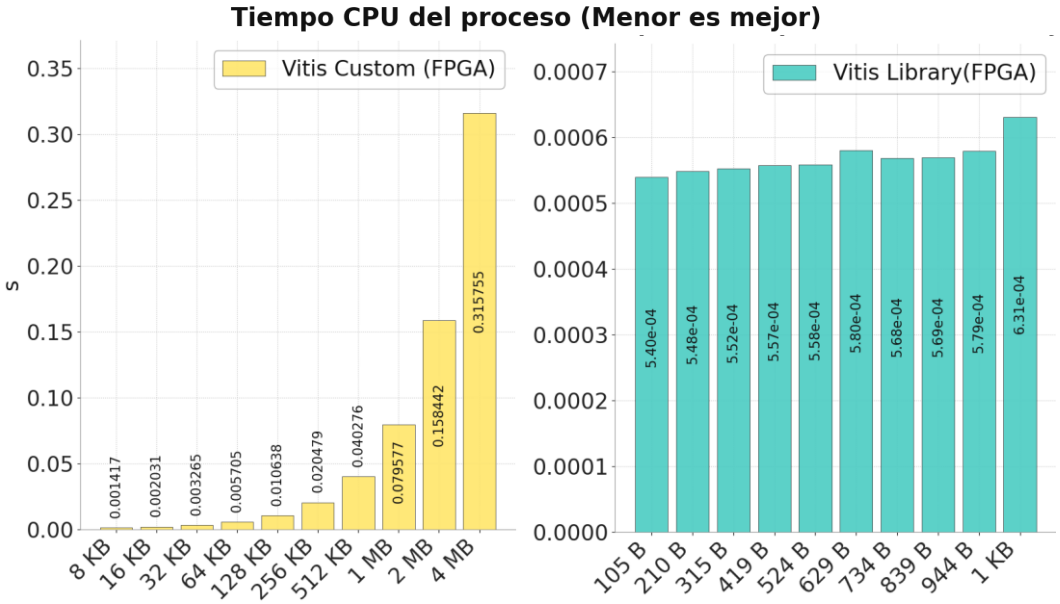
Tiempo de procesamiento puro en la lógica programable (PL) de AES 128.



Nota. Tiempo de ejecución interno del núcleo de AES 128 en la FPGA medido desde el inicio hasta la finalización del kernel, excluyendo transferencias.

Figura 45

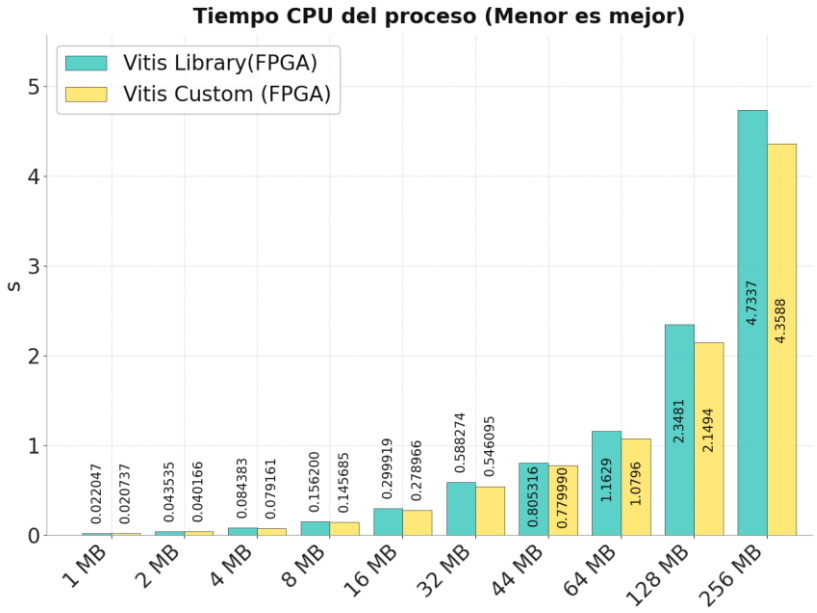
Tiempo de CPU del proceso (ARM) de SHA 256.



Nota. Tiempo durante el cual el procesador ejecuta instrucciones asociadas al proceso de SHA 256, incluyendo control y sincronización.

Figura 46

Tiempo de CPU del proceso (ARM) de AES 128.

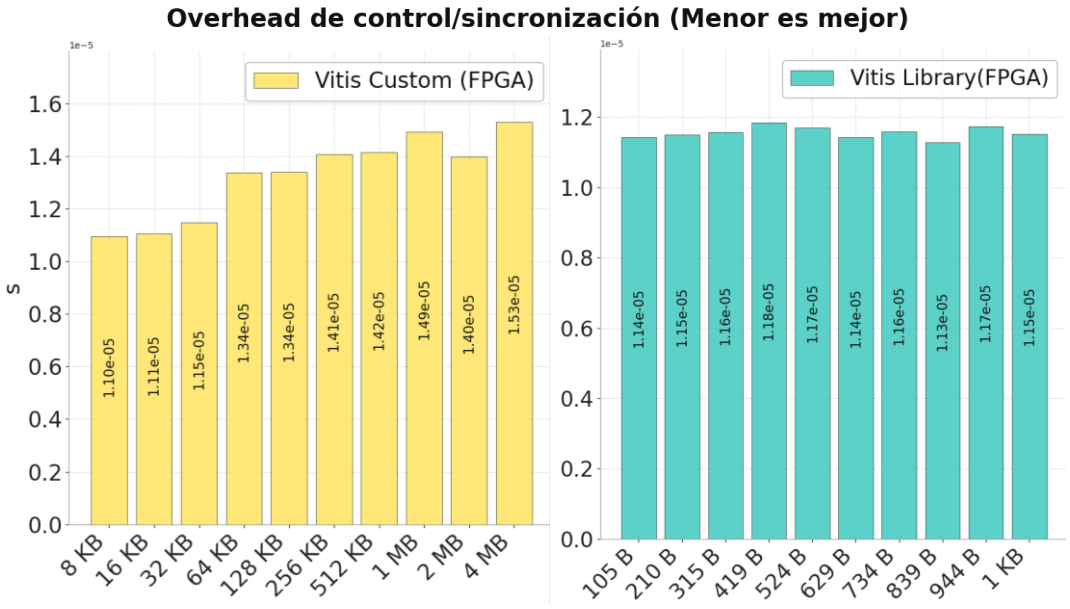


Nota. Tiempo durante el cual el procesador ejecuta instrucciones asociadas al proceso de AES 128, incluyendo control y sincronización.

En cuanto al análisis de sobrecostos para el procesador ARM, dividimos el análisis en dos flujos operativos críticos que corresponden a la transferencia de datos y el control del sistema. En la primera fase, la monopolización del bus debido al movimiento de grandes volúmenes de datos entre la memoria principal y la lógica programable. En la segunda, la gestión de la configuración de los registros, la sincronización con la FPGA y la gestión de interrupciones. La implementación personalizada de los diseños que hemos realizado, tiene como meta reducir estos cuellos de botella. Gracias a la optimización del protocolo de comunicación, hemos reducido de forma drástica la latencia en ambos elementos y, además, hemos liberado ciclos muy valiosos del procesador principal.

Figura 47

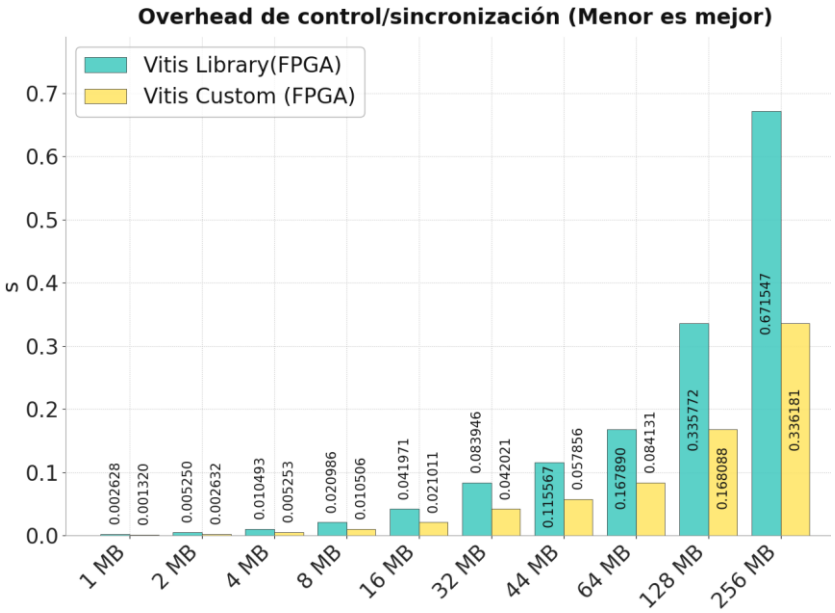
Overhead de control y tiempo de transferencia en FPGA de SHA 256.



Nota. Tiempo no productivo asociado a la configuración del acelerador, movimiento de datos y sincronización del sistema del SHA 256.

Figura 48

Overhead de control y tiempo de transferencia en FPGA de AES 128.



Nota. Tiempo no productivo relacionado con la configuración del acelerador, traslado de datos y la sincronización del sistema del AES 128.

En el análisis realizado, se comprueba que en el sistema SoC-FPGA el límite está en el coste de control y comunicación, y no en el coste de procesamiento del acelerador. Reducir el overhead permite al sistema acercarse al rendimiento teórico del núcleo hardware, lo que pone de manifiesto la mejora que aportan las implementaciones Custom en relación con las de tipo librería.

Análisis de uso de recursos lógicos y su impacto en el desempeño de la FPGA

Complementamos el análisis de rendimiento computacional evaluando el consumo de recursos lógicos, ya que la ocupación efectiva del dispositivo define la viabilidad real del sistema. En el flujo de trabajo con Vitis HLS y Vivado, examinamos dos etapas críticas. La síntesis HLS proyecta una estimación preliminar del costo en hardware. En contraste, la fase de implementación revela el uso exacto de LUTs, Flip-Flops y BRAM tras el mapeo físico en la Zynq-7010. Aunque las tablas siguientes detallan los reportes de síntesis, fundamentamos las conclusiones de desempeño en los valores finales de implementación.

La Tabla 8 muestra la estimación de recursos para el módulo de encriptación de AES-128 en sus versiones librería y Custom. En el resumen, la versión Custom aumenta considerablemente el uso de BRAM en comparación con la implementación de librería. Esto es resultado de una estrategia de diseño que utiliza, durante el procesamiento en modo streaming, buffers internos de mayor tamaño. Esta estimación, a operacionalmente, el área lógica como resultado de priorizar el throughput del acelerador.

Tabla 8

Estimación de recursos y timing para AES-128 Encriptación (Librería vs Custom).

Implementación	Target ns	Estimated ns	BRAM	FF	LUT
FPGA Librería	10	7.882	52	7096	12523
FPGA Custom	10	8.804	144	7470	17039

Nota. La versión Custom incrementa el uso de BRAM para optimizar la transferencia de datos.

Los datos obtenidos en la etapa de implementación muestran diferencias importantes con respecto a las estimaciones de los modelos de síntesis. El módulo de AES-128 personalizado tuvo un consumo real de 7018 LUTs, que supera en 17039 a los estimados, para BRAM se consumieron 74 blocks, lo que valida nuestra arquitectura de la memoria interna. Estos datos valen que la herramienta HLS subestima de manera sistemática el costo lógico. Consideramos que los reportes a posteriori de implementación son los únicos que consideran de manera real el uso físico del dispositivo.

La Tabla 9 muestra los recursos estimados asociados con el algoritmo SHA-256 a nivel de síntesis HLS. Los valores estimados presentan una alta densidad en lógica, particularmente en LUTs, lo que indica una alta presión sobre la lógica combinacional del dispositivo debido a la naturaleza iterativa y secuencial del algoritmo. Este tipo de estimación permite anticipar las restricciones del nivel del sistema de integración lógica y las dificultades adicionales para integrar lógica en más sistemas.

Tabla 9

Estimación de recursos y timing para SHA-256.

Implementación	Target ns	Estimated ns	BRAM	FF	LUT
FPGA Librería	20	14.6	16	8761	14590
FPGA Custom	20	14.6	24	8254	11146

Nota. El alto consumo de LUTs (14,590) evidencia la saturación del dispositivo Zynq-7010.

Las proyecciones iniciales de HLS se reducen drásticamente con la implementación del acelerador SHA-256. El consumo final se ajustó a 5,562 LUTs, 8,528 Flip-Flops, y solo 9 bloques de BRAM. Esta optimización demuestra las capacidades de la herramienta Vivado para la compresión de la lógica en la fase de place & route. No obstante, lo clasificamos como uno de los módulos más densos de todo el sistema. Su elevada demanda de recursos limita la posibilidad de interrumpir aceleradores en paralelo sin afectar el rendimiento del FPGA.

La Tabla 10, en comparación de los módulos de encriptar y de desencriptar de AES-128, en la versión de librería y en la Custom, la síntesis HLS estima un costo lógico mayor para el desencriptado. Esta diferencia se debe a la mayor complejidad de las transformaciones inversas, sobre todo de Inverse MixColumns, que incrementa la profundidad lógica del diseño.

Tabla 10

Estimación de recursos y timing para AES-128 Desencriptación.

Implementación	Target ns	Estimated ns	BRAM	FF	LUT
FPGA Librería	10	7.88	52	9990	21740
FPGA Custom	10	8.80	145	9774	25228

Nota. La lógica inversa incrementa el consumo de recursos frente al cifrado directo.

Puede ser un resultado real de usar el dispositivo, aunque geográficamente usa la asimetría manteniendo una cierta cantidad, y en términos de implementación, el módulo de desencriptación personalizado AES-128 utiliza 8,928 LUTs en comparación con 7,018 para su contraparte de encriptación y también un uso constante de 74 BRAM. Estos resultados demuestran que la desencriptación penaliza el rendimiento general del sistema, no solo por su latencia computacional, sino también por su mayor ocupación lógica que reduce el espacio disponible para el paralelismo adicional en la FPGA.

Por último, el análisis conjunto de síntesis e implementación permite determinar el límite práctico del sistema propuesto. Las estimaciones HLS dan una primera aproximación al costo de la arquitectura, sin embargo, es la implementación la que realmente influye en las prestaciones del FPGA. Para el Zynq-7010, el consumo real de recursos del bloque SHA-256 y las versiones Custom de AES limita la posibilidad de instanciar múltiples núcleos en paralelo. Así, el rendimiento operativo alcanzado está muy cerca del óptimo que se puede ofrecer a esta plataforma y cualquier mejora significativa debe dirigirse a dispositivos que aumenten la lógica para, ondular la explotación del paralelismo.

Capítulo VI: Conclusiones y Trabajos Futuros

Conclusiones

Se realizó un análisis de rendimiento para AES-128 y SHA-256 en software y hardware y se compararon utilizando medidas objetivas de tiempo, arquitectura y uso de recursos. Los resultados experimentales mostrados en las tablas relevantes para el análisis de rendimiento y latencia E2E, muestran que la velocidad del algoritmo no es el único factor importante. El rendimiento final es el resultado del equilibrio físico entre el diseño del acelerador y la capacidad real de movimiento de datos entre ARM y FPGA. La CPU estableció un alto estándar. Fue bueno debido a su alta frecuencia y al uso de bibliotecas optimizadas. Esto hizo posible una comparación justa con el código compilado.

También aprovecha la memoria del procesador ARM. Por otro lado, la plataforma SoC-FPGA que vimos estaba bajo la estructura2 de frecuencia que estaba limitada a 100 MHz como se define en la configuración de diseño y se valida en los informes de implementación. Lo que encontramos es que aunque el kernel de hardware tiene latencias internas reducidas, el rendimiento de extremo a extremo es un factor de elementos externos al cálculo puro. También descubrimos que a la FPGA le va mejor con el paralelismo estructural y la especialización funcional, que no siempre compensan la brecha de frecuencia en algoritmos que tienen elementos secuenciales como es el caso de SHA-256.

El análisis comparativo de métricas de kernel y métricas de extremo a extremo muestra que uno de los principales cuellos de botella no es tanto la capacidad aritmética del acelerador, sino el control y la sobrecarga del movimiento de datos, y en particular, la transferencia DMA, la sincronización ARM y el control de bloques que crean latencia que se acumula en función del tamaño del mensaje y puede superar el tiempo de cálculo del núcleo de hardware. En contraste, la CPU tiene una mejor y más consistente tolerancia a la latencia debido a la

arquitectura, y es mejor en el control y flujo de datos en comparación con la CPU, en el flujo y control de datos.

También se evaluó la hipótesis central de la investigación: El diseño Custom desarrollado, de forma consistente, supera a las bibliotecas estándar en la misma FPGA, como se pueden ver en las tablas de la comparación de rendimiento interno. Al realizar el control holgado y optimizar el control del flujo de datos a través de una arquitectura específica, se logran mejoras en el rendimiento End-to-End, en comparación de implementaciones genéricas. Este resultado logra el objetivo específico de construir un acelerador optimizado que haga uso de los recursos disponibles, sin comprometer las restricciones de temporización y área.

Ciertamente, el hardware de nuestra solución no permite ganar tiempo, ya que su efectividad varía en función del balance entre cómputo, gestión de memoria y transferencia de datos. Los objetivos alcanzados permiten afirmar que un diseño reflexionado sobre estas variables permite aprovechar el potencial de los sistemas reconfigurables, en situaciones de severas restricciones de frecuencia y arquitectura.

Trabajos Futuros

Como primera medida, establecemos la escalabilidad para archivos masivos. El procesamiento de bloques y cuellos de botella actuales confirma que el rendimiento se ve limitado. Proponemos que se rediseñen los mecanismos de buffering y streaming para que se pueda habilitar un flujo de datos continuo y, por lo tanto, se eliminen las constantes reinicializaciones del sistema. En paralelo, iremos evolucionando el núcleo AES para integrar los modos CTR y GCM. Esta evolución permitirá que se puedan ejecutar en hardware de forma directa, aunque se tendrá que confirmar el impacto que la lógica de control adicional tendrá sobre el throughput end to end.

Nuestra propuesta incide en la optimización de la interfaz entre el procesador y la lógica programable que el sistema hace uso de construcciones DMA en avance y colas de hardware dedicadas. Este sistema integrará nuevas formas de valorar la energía consumida, lo que, junto

con el valor de la energía utilizada, menciona la posible versatilidad que el acelerador presenta para su uso en IoT y Edge Computing. Asimismo, se prevé la migración a sistemas Zynq-7020, de mayor densidad, que, junto con la posibilidad de incursionar en múltiples núcleos de procesadores criptográficos a la vez, le permitirá a nuestra arquitectura Custom adaptarse a los amplios requerimientos de seguridad en las nuevas redes.

Referencias

[1] B. Navas, “2020-PIC-015-INV (CRaA: Computación Reconfigurable y Auto-Adaptiva: Aplicada a la Industria de la Defensa-Fase 01)”, ELÉCTRICA Y ELECTRÓNICA. Consultado: el 3 de septiembre de 2025. [En línea]. Disponible en: https://deee.espe.edu.ec/2020-pic-015-inv/

[2] D. Berrazueta-Mena y B. Navas, “AHA: Design and Evaluation of Compute-Intensive Hardware Accelerators for AMD-Xilinx Zynq SoCs Using HLS IP Flow”, *Computers*, vol. 14, núm. 5, p. 189, may 2025, doi: 10.3390/computers14050189.

[3] “Implementation on FPGA • Vitis Libraries • Reader • AMD Technical Information Portal”. Consultado: el 3 de septiembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/2022.2-English/Vitis_Libraries/security/guide_L1/internals/sha2.html_1

[4] C. E. B. Santos, L. M. D. da Silva, M. F. Torquato, S. N. Silva, y M. A. C. Fernandes, “SHA-256 Hardware Proposal for IoT Devices in the Blockchain Context”, *Sensors (Basel)*, vol. 24, núm. 12, p. 3908, jun. 2024, doi: 10.3390/s24123908.

[5] R. Karakchi, R. Stahle-Smith, N. Chinnasami, y T. Yu, “Toward a Lightweight, Scalable, and Parallel Secure Encryption Engine”, el 18 de junio de 2025, *arXiv*: arXiv:2506.15070. doi: 10.48550/arXiv.2506.15070.

[6] O. Al-Khaleel, S. Baktir, M. Al-Khaleel, y A. Küpçü, “Efficient ECC Processor Designs for IoT Using Edwards Curves and Exploiting FPGA Embedded Components”, *IEEE Access*, vol. 12, pp. 167183–167200, 2024, doi: 10.1109/ACCESS.2024.3495995.

[7] Crockett, Louise H.; Elliot, Ross A.; Enderwitz, Martin A.; Stewart, Robert W., *The Zynq Book: Embedded Processing with the ARM® Cortex®-A9 on the Xilinx® Zynq®-7000 All Programmable SoC*, 1st Edition. Glasgow, Scotland, UK: Strathclyde Academic Media, 2014.

[8] “AMD Zynq™ 7000 SoCs”, AMD. Consultado: el 12 de febrero de 2026. [En línea]. Disponible en: https://www.amd.com/de/products/adaptive-socs-and-fpgas/soc/zynq-7000.html

[9] “AMD Vitis™ HLS”, AMD. Consultado: el 11 de diciembre de 2025. [En línea]. Disponible en: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html

[10] “Vitis Libraries • Vitis Libraries • Reader • AMD Technical Information Portal”. Consultado: el 12 de febrero de 2026. [En línea]. Disponible en: https://docs.amd.com/r/en-US/Vitis_Libraries/index.html

[11]“AMD Vitis™ Accelerated Libraries”, AMD. Consultado: el 13 de febrero de 2026. [En línea]. Disponible en: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-libraries.html

[12]“AMD Vitis™ Accelerated Libraries”, AMD. Consultado: el 12 de febrero de 2026. [En línea]. Disponible en: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-libraries.html

[13]“Vitis HLS Design Flow Lab”, High Level Systhesis Design Flow. Consultado: el 11 de diciembre de 2025. [En línea]. Disponible en: https://xilinx.github.io/xup_high_level_synthesis_design_flow/Lab1.html

[14]“Vitis Security Library • Vitis Libraries • Reader • AMD Technical Information Portal”. Consultado: el 12 de febrero de 2026. [En línea]. Disponible en: https://docs.amd.com/r/en-US/Vitis_Libraries/security/index.html

[15]“Vitis Libraries • Vitis Libraries • Reader • AMD Technical Information Portal”. Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/2023.2-English/Vitis_Libraries/index.html

[16]“AMD Vitis HLS Performance Pragma”. Consultado: el 11 de diciembre de 2025. [En línea]. Disponible en: https://www.amd.com/content/dam/amd/en/documents/products/vitis/vitis-hls-performance-pragma.pdf

[17]“Accelerated App Development on Kria KD240 in Vitis 2023.2”, Hackster.io. Consultado: el 11 de diciembre de 2025. [En línea]. Disponible en: https://www.hackster.io/whitney-knitter/accelerated-app-development-on-kria-kd240-in-vitis-2023-2-374fc7

[18]“AMD Technical Information Portal”. Consultado: el 11 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/en-US/ug949-vivado-design-methodology/Vivado-Design-Suite-IP-Catalog

[19]“Vivado What’s New”. Consultado: el 11 de diciembre de 2025. [En línea]. Disponible en: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado/vivado-whats-new.html#tabs-de9b056824-item-d2d3a6dfcc-tab

[20]“Accelerating Data Center Applications with Vitis Software Platform • Data Center Acceleration Using Vitis User Guide (UG1700) • Reader • AMD Technical Information Portal”. Consultado: el 11 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/en-US/ug1700-vitis-accelerated-data-center/Accelerating-Data-Center-Applications-with-Vitis-Software-Platform

[21]“AMD Vitis™ Integrated Design Environment (IDE)”, AMD. Consultado: el 11 de diciembre de 2025. [En línea]. Disponible en: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-ide.html

[22]“Welcome to pyca/cryptography — Cryptography 47.0.0.dev1 documentation”. Consultado: el 11 de diciembre de 2025. [En línea]. Disponible en: https://cryptography.io/en/latest/

[23]National Institute of Standards and Technology (US), “Secure hash standard”, National Institute of Standards and Technology (U.S.), Washington, D.C., NIST FIPS 180-4, 2015. doi: 10.6028/NIST.FIPS.180-4.

[24]N. I. of S. and Technology, “Advanced Encryption Standard (AES)”, U.S. Department of Commerce, Federal Information Processing Standard (FIPS) 197, may 2023. doi: 10.6028/NIST.FIPS.197-upd1.

[25]“TS 133 501 - V18.10.0 - 5G; Security architecture and procedures for 5G System (3GPP TS 33.501 version 18.10.0 Release 18)”.

[26]“NIST SP 800-38D, Recommendationfor Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC”. Consultado: el 12 de diciembre de 2025. [En línea]. Disponible en: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf

[27]E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, Internet Engineering Task Force, Request for Comments RFC 8446, ago. 2018. doi: 10.17487/RFC8446.

[28]“Introduction • Vitis Unified IDE and Common Command-Line Reference Manual (UG1553) • Reader • AMD Technical Information Portal”. Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/en-US/ug1553-vitis-ide/Introduction

[29]“pragma HLS interface • Vitis High-Level Synthesis User Guide (UG1399) • Reader • AMD Technical Information Portal”. Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/en-US/ug1399-vitis-hls/pragma-HLS-interface

[30]“AXI4-Lite Interface • Vitis High-Level Synthesis User Guide (UG1399) • Reader • AMD Technical Information Portal”. Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/2023.2-English/ug1399-vitis-hls/AXI4-Lite-Interface

[31]“S_AXILITE Bundle Rules • Vitis High-Level Synthesis User Guide (UG1399) • Reader • AMD Technical Information Portal”. Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/2023.2-English/ug1399-vitis-hls/S_AXILITE-Bundle-Rules

[32]“Output of C/RTL Co-Simulation • Vitis High-Level Synthesis User Guide (UG1399) • Reader • AMD Technical Information Portal”. Consultado: el 26 de diciembre de 2025. [En

línea]. Disponible en: https://docs.amd.com/r/2023.2-English/ug1399-vitis-hls/Output-of-C/RTL-Co-Simulation

[33] "Running C/RTL Co-Simulation • Vitis High-Level Synthesis User Guide (UG1399) • Reader • AMD Technical Information Portal". Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/2023.2-English/ug1399-vitis-hls/Running-C/RTL-Co-Simulation

[34] "Designing with IP Integrator • Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator (UG994) • Reader • AMD Technical Information Portal". Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/en-US/ug994-vivado-ip-subsystems/Designing-with-IP-Integrator

[35] "Vivado Design Suite User Guide: Embedded Processor Hardware Design (UG898) • Viewer • AMD Technical Information Portal". Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/api/khub/documents/fumFEgy5sU25HHz8NhtzpQ/content?Ft-Calling-App=ft%2Fturnkey-portal&Ft-Calling-App-Version=5.2.19#

[36] "Managing IP Repositories • Vivado Design Suite User Guide: Designing with IP (UG896) • Reader • AMD Technical Information Portal". Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/en-US/ug896-vivado-ip/Managing-IP-Repositories

[37] "Documentation Resources • Zynq 7000 SoC Technical Reference Manual (UG585) • Reader • AMD Technical Information Portal". Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM/Documentation-Resources

[38] "AXI Reference Guide (UG761) • Viewer • AMD Technical Information Portal". Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/api/khub/documents/olK8hi~2WaCHfZ2Ppi2I_w/content?Ft-Calling-App=ft%2Fturnkey-portal&Ft-Calling-App-Version=5.2.19#

[39] "Extensible XSA • Embedded Design Development Using Vitis User Guide (UG1701) • Reader • AMD Technical Information Portal". Consultado: el 26 de diciembre de 2025. [En línea]. Disponible en: https://docs.amd.com/r/en-US/ug1701-vitis-accelerated-embedded/Extensible-XSA

[40] GrantMeStrength, "Install WSL". Consultado: el 13 de febrero de 2026. [En línea]. Disponible en: https://learn.microsoft.com/en-us/windows/wsl/install

[41]“Ships in Satellite Imagery”. Consultado: el 9 de febrero de 2026. [En línea]. Disponible en: https://www.kaggle.com/datasets/rhammell/ships-in-satellite-imagery