



Universidad de Sevilla
Escuela Politécnica Superior de Sevilla



TRABAJO FIN DE GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL

Implementación de algoritmos criptográficos lightweight sobre sistemas empotrados

Autor: Sergio Sauro del Valle

Tutores: Carlos J. Jiménez Fernández

Macarena C. Martínez Rodríguez

Curso: 2018-2019

Trabajo de Fin de Grado
Grado en Ingeniería Electrónica Industrial

Implementación de algoritmos criptográficos lightweight sobre sistemas empotrados

Autor:

Sergio Sauro del Valle

Tutores:

Carlos J. Jiménez Fernández

Macarena C. Martínez Rodríguez

Departamento de Tecnología Electrónica

Escuela Politécnica Superior de Sevilla

Universidad de Sevilla

Sevilla, 2019

Trabajo de Fin de Grado:

Implementación de algoritmos criptográficos lightweight sobre sistemas empuotrados.

Autor: Sergio Sauro del Valle

Tutores: Carlos J. Jiménez Fernández

Macarena C. Martínez Rodríguez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle una calificación de:

Sevilla, 2019

El secretario del tribunal

Agradecimientos

Con la realización de este trabajo, se cierra una etapa muy importante en mi vida, por lo que es el mejor momento para agradecer a todas las personas que me han ayudado a superar las dificultades encontradas en mi camino hasta ahora. Empiezo con mi familia, que ha estado apoyándome desde que comencé la Universidad y especialmente en este proyecto. También quiero agradecer a una persona especial que ha estado dándome apoyo moral desde que la conozco y me ha demostrado que no es necesario estudiar la misma carrera para hacerte saber que está ahí para ayudarte en lo que necesites.

Agradecer a Carlos, mi tutor, haberme dado la oportunidad de realizar mi proyecto sobre un tema tan interesante que desconocía por completo.

Por último, quiero agradecer a Macarena, mi tutora, y especialmente a Santiago, por haberme permitido estar el curso entero con ellos aprendiendo día tras día en el instituto de microelectrónica. Sin ellos, la realización de este proyecto habría sido imposible. Muchísimas gracias por vuestra buena labor.

Sergio Sauro del Valle

Grado en Ingeniería Electrónica Industrial

Sevilla, 2019

Resumen

Este trabajo presenta un flujo de diseño que facilita el desarrollo y evaluación de distintas soluciones para la implementación hardware de cifradores autenticados y su incorporación como periféricos aceleradores en sistemas empuetrados para distintos casos de aplicación.

Por un lado, la competición CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) ha establecido un catálogo de algoritmos que permitan cifrar y autenticar mensajes de forma simultánea, que resulten robustos desde la perspectiva de la seguridad que puedan proporcionar, y eficientes en cuanto al rendimiento de sus implementaciones hardware y software para distintas categorías. Por otro lado, los actuales dispositivos programables constituyen una plataforma ideal de desarrollo y test, tanto para las propuestas asociadas a la categoría “Lightweight”, como para las de “alto-rendimiento”.

De acuerdo con las consideraciones anteriores, el principal objetivo de este proyecto ha sido implementar, mediante dispositivos SoC, algunos de los algoritmos finalistas en la competición CAESAR con idea de disponer de implementaciones eficientes de estos algoritmos para su uso en sistemas empuetrados con recursos de cómputo y memoria limitado y comparar distintas realizaciones hardware, software y/o híbridas de estos algoritmos sobre la misma plataforma de desarrollo.

Abstract

This work presents a design flow that eases the development and evaluation of different solutions for the hardware implementation of authenticated ciphers and their incorporation as accelerating peripherals in embedded systems for different application cases.

On the one hand, the CAESAR competition (Competition for Authenticated Encryption: Security, Applicability, and Robustness) has set a portfolio of algorithms that encrypt and authenticate messages simultaneously, which are robust from the security perspective and efficient in terms of the performance of their hardware and software implementations for the different categories. On the other hand, the current programmable devices set an ideal platform for the development and testing, both for the proposals associated with the "Lightweight" category, as well as for those focused on "High-performance".

According to the previous considerations, the main goal of this project is to implement in SoC devices, some of the finalist algorithms of the CAESAR competition with the objective of obtain efficient implementations of these algorithms in order to use them in embedded systems with limited computing resources and memory, and also to compare different hardware, software and / or hybrid realizations of these algorithms on the same development platform.

Índice

1.	Introducción	2
1.1.	Criptología.....	2
1.2.	Criptografía. Tipos de cifradores y protocolos	3
1.3.	El cifrador autenticado.....	7
2.	Competición CAESAR y proyecto ATHENa	8
2.1.	Competición CAESAR	8
2.1.1.	Requerimientos para la participación	8
2.1.2.	Requerimientos para las propuestas.	9
2.1.3.	Resultados de la competición:	11
2.2.	PROYECTO ATHENa.....	11
2.2.1.	GMU Hardware API	13
2.2.1.	Fuentes RTL del módulo cifrador.....	13
2.2.2.	Protocolo de comunicación.....	18
2.2.3.	Generación de vectores de test.	20
3.	Cifradores autenticados	22
3.1.	AEGIS.....	22
3.1.1.	Introducción.....	22
3.1.2.	Especificaciones.....	23
3.1.3.	AEGIS-128	25
3.2.	ACORN.....	28
3.2.1.	Introducción.....	28
3.2.2.	Especificaciones.....	28
3.2.3.	ACORN-128	29
4.	Implementación de cifradores empotrados en SoC.....	34
4.1.	Entorno de trabajo	34
4.1.1.	Placa de desarrollo.	34
4.1.2.	Entorno de diseño Hardware.	38
4.1.3.	Entorno de diseño Software.....	40
4.2.	Desarrollo del hardware del sistema empotrado	41
4.2.1.	Implementación de cifrador.	41
4.2.2.	Generación del módulo IP del cifrador.	45
4.2.3.	Conexión del módulo IP al procesador.....	51
4.3.	Desarrollo del software del sistema empotrado.....	59

4.3.1. Adaptación del software de ATHENA a las limitaciones del sistema empotrado.	59
4.3.2. Implementación software del cifrador en el sistema empotrado.	61
4.3.3. Software de verificación del módulo IP (AXI4-Lite y AXI4).	62
4.3.4. Software del módulo IP (AXI DMA)	65
5 Resultados obtenidos.....	68
5.1. Implementación hardware del cifrador	68
5.2. Incorporación del cifrador en un sistema empotrado.....	69
5.2.1. Conexión del módulo IP mediante FIFOs	69
5.2.2. Conexión del módulo IP usando DMA.	73
5.3. Verificación de los sistemas empotrados	74
5.3.1. Implementación del cifrador mediante software empotrado.	75
5.3.2. Sistemas empotrados con FIFOs (AXI4-Lite y AXI4).	76
.....	80
5.3.3. Sistema empotrado conectado mediante DMA.	80
5.3.4. Comparación de los distintos esquemas de conexión.....	82
5.4. Implementación de otros cifradores.	84
5.4.1. Implementación del cifrador ACORN.....	85
5.4.2. Generación del módulo IP del cifrador.	87
5.4.3. Inclusión del cifrador ACORN en un sistema empotrado.....	89
5.4.4. Comparación de los recursos utilizados entre AEGIS y ACORN.....	92
6 Conclusiones y trabajo futuro	94
Bibliografía	96

Tabla de ilustraciones

Figura 1. Criptología y sus principales ramas.....	2
Figura 2. Función de transformación SubBytes().....	5
Figura 3. Función de transformación ShiftRows.....	5
Figura 4. Función de transformación MixColumns().....	6
Figura 5. Función de transformación AddRoundKey().....	6
Figura 6. Diagrama de bloques de alto nivel de la interfaz para cifradores dirigidos a aplicaciones de alto rendimiento ("High-Performance").....	14
Figura 7. Entrada de datos públicos y secretos en el Pre-Procesador.....	15
Figura 8. Salida de datos del Post-Procesador.....	17
Figura 9. Entrada y salida de un cifrador autenticado.....	18
Figura 10. Formato de sdi para a) Cargar clave principal, b) Cargar secuencia de rondas de clave.	18
Figura 11. Ejemplo del formato de instrucción para SDI.....	19
Figura 12. Formato de datos públicos de entrada para los casos a) un único segmento de cada tipo de datos, b) múltiples segmentos de mensajes y datos asociados.....	19
Figura 13. Ejemplo del formato de instrucción para PDI.....	20
Figura 14. Función de actualización de estado de AEGIS-128.	25
Figura 15. Concatenación de los 6 LFSRs. fi indica el bit general de realimentación para el paso i; mi indica el bit de mensaje para el paso i.	29
Figura 16. Placa de desarrollo Zybo.	35
Figura 17. Esquema interno del Cortex-A9.....	36
Figura 18. Interfaz gráfica del entorno de diseño Xilinx Vivado 2018.2.....	38
Figura 19. Repositorio IP de Vivado.....	39
Figura 20. Entorno de desarrollo Xilinx SDK.	40
Figura 21. Formato de instrucción modificada del fichero pdi.txt.	41
Figura 22. Modificación de la anchura de los buses de datos públicos.	43
Figura 23. Fuentes de AEGIS-32, a) Fuentes de diseño, b) Fuentes de simulación. ...	43
Figura 24. Simulación de AEGIS-32.	44
Figura 25. Esquema del módulo AEAD_IP.	45
Figura 26. Fuentes del módulo AEGIS-IP, a) Fuentes de diseño, b) Fuentes de simulación.....	46
Figura 27. Asistente de creación de un nuevo paquete IP.....	47
Figura 28. Simulación del comportamiento de AEGIS-IP.....	47
Figura 29. Salida del periférico cifrador.	48
Figura 30. Configuración de la señal de reloj con los parámetros adecuados.....	49
Figura 31. Diagrama de bloque cipher usando "aegis".	49
Figura 32. Creación del envoltorio del diseño.....	50
Figura 33. Fuentes de simulación de AEGIS-BD.....	50
Figura 34. Proyecto completo AEGIS-BD.	51
Figura 35. Simulación de AEGIS-BD.	51
Figura 36. Configuración del módulo "AXI-Stream FIFO" para conectarlo al procesador a través del bus AXI4-Lite.	53

Figura 37. Configuración del módulo "AXI-Stream FIFO" para conectarlo al procesador a través del bus AXI4-Lite.	53
Figura 38. Interfaz de configuración del procesador ARM.	54
Figura 39. Conexión de los bloques del sistema empujado usando AXI4-Lite.	54
Figura 40. Configuración "AXI Interconnect".....	55
Figura 41. Conexión de los bloques del sistema empujado.....	55
Figura 42. Configuración del AXI DMA.	56
Figura 43. Configuración del bus AXI SMART-CONNECT.	57
Figura 44. Configuración del componente Concat.....	57
Figura 45. Diagrama de bloques del sistema empujado.	58
Figura 46. Jerarquía del diseño ARM con acceso directo a memoria.	58
Figura 47. Detalle del fichero de cabecera "data.h".....	60
Figura 48. Rutina de medida de tiempo creada para las aplicaciones software de verificación.....	60
Figura 49. Jerarquía de las aplicaciones de verificación de la implementación software del cifrador en el sistema empujado. En A) IP conectado con FIFOs a través de buses AXI4-Lite, en B) con buses AXI4 y en C) IP conectado mediante acceso directo a memoria.	61
Figura 50. Fuentes de la aplicación basada en polling para los sistemas empujados que utilizan buses AXI4-Lite en A) y AXI4 en B).	63
Figura 51. Fuentes de la aplicación basada en interrupciones para los sistemas empujados que emplean buses AXI4-Lite en (A) y con buses AXI4 en (B).	64
Figura 52. Fuentes de la aplicación de verificación basada en polling para configuraciones del sistema empujado que emplea buses AXI-DMA.....	66
Figura 53. Jerarquía de la aplicación de verificación mediante interrupciones para la configuración del sistema empujado que emplea AXI-DMA.....	67
Figura 54. Utilización detallada de los recursos de la FPGA tras la implementación de AEGIS-32.....	68
Figura 55. Utilización detallada porcentual de los recursos de la FPGA tras la implementación de AEGIS-32.	69
Figura 56. Utilización de la FPGA tras la implementación del sistema empujado utilizando FIFOs conectadas mediante AXI4-Lite.	69
Figura 57. Utilización de recursos de la FPGA tras la implementación del sistema empujado utilizando FIFOs conectadas mediante buses AXI4-Lite.	70
Figura 58. Utilización porcentual de la FPGA tras la implementación del sistema empujado utilizando FIFOs conectadas mediante buses AXI4-Lite.	70
Figura 59. Potencia consumida por la FPGA para el sistema empujado que utiliza FIFOs conectadas mediante buses AXI4-Lite.	71
Figura 60. Utilización de la FPGA tras la implementación del sistema empujado utilizando una FIFO conectada mediante bus AXI4.....	71
Figura 61. Utilización de los recursos de la FPGA tras la implementación del sistema empujado utilizando una FIFO conectada al bus AXI4.....	72
Figura 62. Potencia consumida por la FPGA para el sistema empujado que emplea una FIFO conectada mediante bus AXI4.	72
Figura 63. Utilización de la FPGA tras la implementación del sistema empujado que utiliza DMA.	73
Figura 64. Utilización de los recursos de la FPGA tras la implementación del sistema empujado que utiliza DMA.....	73

Figura 65. Utilización porcentual de los recursos de la FPGA tras la implementación del sistema empujado que utiliza DMA.	74
Figura 66. Potencia consumida por la FPGA tras implementar el sistema que emplea el módulo AXI DMA.	74
Figura 67. Tiempo de ejecución de la implementación software del cifrador con las cachés de datos e instrucciones habilitadas.....	75
Figura 68. Tiempo de ejecución de la implementación software del cifrador sin la caché de datos ni la de instrucciones habilitadas.....	75
Figura 69. Tiempo de ejecución de la implementación software del cifrador sin la caché de datos habilitada.	76
Figura 70. Ejecución de la aplicación de verificación mediante polling con el modo de depuración Debug 0 en el sistema con FIFOs y AXI4-Lite.....	76
Figura 71. Ejecución de la verificación mediante polling con el modo de depuración Debug 1 en el sistema con FIFOs y AXI4-Lite.....	77
Figura 72. Ejecución de la aplicación de verificación por interrupciones en modo Debug 0 en el sistema con FIFOs y AXI4-Lite.	77
Figura 73. Ejecución de la aplicación de verificación por interrupciones en modo Debug 1 en el sistema con FIFOs y AXI4-Lite.	78
Figura 74. Ejecución de la aplicación de verificación mediante polling en modo Debug 0 en el sistema que usa una FIFO conectada al bus AXI4.....	78
Figura 75. Ejecución de la aplicación de verificación mediante polling en modo Debug 1 en el sistema que usa una FIFO conectada al bus AXI4.....	79
Figura 76. Ejecución de la aplicación de verificación por interrupciones en modo Debug 0 en el sistema que usa una FIFO conectada al bus AXI4.....	79
Figura 77. Ejecución de la aplicación de verificación por interrupciones en modo Debug 1 en el sistema que usa una FIFO conectada al bus AXI4.....	80
Figura 78. Ejecución de la aplicación de verificación mediante polling en modo Debug 0 en el sistema que emplea el módulo DMA.	80
Figura 79. Ejecución de la aplicación de verificación mediante polling en modo Debug 1 en el sistema que emplea el módulo DMA.	81
Figura 80. Ejecución de la aplicación de verificación por interrupciones en modo Debug 0 en el sistema que emplea el módulo DMA.	81
Figura 81. Ejecución de la aplicación de verificación por interrupciones en modo Debug 1 en el sistema que emplea el módulo DMA.	82
Figura 82. Resultado de simulación del comportamiento del cifrador autenticado ACORN.	86
Figura 83. Utilización de recursos de la FPGA para el cifrador autenticado ACORN... ..	86
Figura 84. Fuentes del módulo IP del cifrador autenticado ACORN.....	87
Figura 85. Esquemático del módulo IP del cifrador ACORN.....	88
Figura 86. Simulación del comportamiento del módulo IP del cifrador ACORN.	88
Figura 87. Fuentes y diagrama de bloques del periférico AXI del módulo IP del cifrador ACORN.	89
Figura 88. Resultado de la simulación del comportamiento del periférico del módulo IP de ACORN.....	89
Figura 89. Sistema empujado con el módulo IP del cifrador ACORN.....	90
Figura 90. Consumo de recursos de la FPGA para el sistema empujado que utiliza el cifrador ACORN.....	91
Figura 91. Tiempos de ejecución en Software del cifrador ACORN con distintas configuraciones de memoria caché.....	91

Figura 92. Verificación del funcionamiento del diseño a través de la herramienta de depuración proporcionada por el "Hardware Manager".	92
---	----

GUÍA DE TABLAS

Tabla 1. Componentes de la placa de desarrollo Zybo..... 35

Tabla 2 Comparación de todas las estrategias implementadas para la creación de cipher-32..... 45

Tabla 3. Comparación del consumo de recursos de la FPGA para cada uno de los sistemas empotrados..... 82

Tabla 4. Comparación de tiempos de ejecución para los distintos sistemas empotrados y diferentes opciones de verificación. 83

Tabla 5. Comparación de los recursos utilizados por los cifradores AEGIS y ACORN.92

Motivación y objetivos

En una Sociedad cada vez más “digitalizada”, en la que las Tecnologías de la Información y las Comunicaciones facilitan el intercambio de datos necesario para el desarrollo de muchas actividades cotidianas de carácter laboral, económico o social, los distintos aspectos relacionados con la seguridad en la transmisión de dichos datos cobran cada día mayor importancia.

Esta circunstancia, junto a otras motivaciones de naturaleza estratégica o defensiva, ha provocado un interés creciente por parte de estados, empresas y organizaciones internacionales en el desarrollo y estandarización de protocolos criptográficos seguros, que les permitan salvaguardar sus intereses y/o potenciar el desarrollo de sus áreas de actividad.

Frente a un desarrollo tradicionalmente “secreto” de los procedimientos y dispositivos criptográficos, en los últimos años se ha convertido en práctica habitual la celebración de competiciones abiertas, enfocadas a la selección de las propuestas que se adapten mejor a las necesidades y condiciones establecidas en el certamen. La competición CAESAR, fue iniciada en 2013 con el objetivo concreto de buscar algoritmos que permitan cifrar y autenticar mensajes de forma simultánea, que resulten robustos desde la perspectiva de la seguridad que puedan proporcionar, y eficientes en cuanto al rendimiento de sus implementaciones hardware y software para distintas categorías o casos de aplicación.

Una de estas categorías, la denominada “Lightweight”, estrechamente relacionada con los nuevos retos que plantea la proliferación de dispositivos con tamaño y consumo de potencia reducidos, pero elevadas capacidad de cómputo y comunicación, surgidos en el contexto de lo que ha venido en llamarse “la Internet de las cosas” o IoT, contempla el desarrollo de cifradores autenticados que proporcionen una buena relación coste/rendimiento.

Los actuales dispositivos programables constituyen una plataforma ideal de desarrollo y test, tanto para las propuestas asociadas a este caso de uso, como para las enfocadas a otro más convencional etiquetado como “High-Performance”. En particular, los dispositivos de la familia Zynq-7000, que combinan un sistema de procesado basado en un ARM Cortex-A9 de doble núcleo con lógica programable de las FPGAs de la Serie 7 de Xilinx, resultan especialmente atractivos para el diseño de “Sistemas en un Chip” (SoC) que faciliten el desarrollo y la comparación de implementaciones hardware y software de los distintos cifradores propuestos.

De acuerdo con las consideraciones anteriores, el principal objetivo de este proyecto es implementar, mediante dispositivos SoC de la familia Zynq-7000 de Xilinx, algunos de los algoritmos finalistas en la competición CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) con idea de:

- Disponer de implementaciones eficientes de estos algoritmos para su uso en sistemas empujados con recursos de cómputo y memoria limitados.

- Comparar distintas realizaciones hardware, software y/o híbridas de estos algoritmos sobre la misma plataforma de desarrollo.

Esta memoria recoge las principales tareas llevadas a cabo para cumplir los objetivos anteriores, así como los resultados obtenidos que pueden resultar más significativos. Su estructura es la siguiente:

Para contextualizar el campo en el que se desarrolla el trabajo, en el Capítulo 1 se introducen una serie de conceptos básicos relacionados con la Criptología, sus distintas vertientes y los diferentes tipos de cifradores y protocolos criptográficos. Se incluye asimismo una descripción en profundidad del cifrador AES, ya que, además de ser una referencia obligada por su uso en muchos de los protocolos criptográficos actuales, es la base de algunos de los cifradores autenticados que se contemplan en este trabajo.

En el Capítulo 2 se explica con detalle la competición CAESAR, llevada a cabo para determinar los mejores cifradores autenticados para diferentes casos de uso y de acuerdo con los requerimientos de diseño hardware y software que se establecen en dicha competición. Se describen asimismo las distintas herramientas proporcionadas por el proyecto ATHENA para facilitar el desarrollo y la comparación de las implementaciones hardware de las propuestas presentadas a la competición.

En el Capítulo 3 se detallan las características más importantes, arquitectura, componentes, nivel de seguridad, etc. de dos de los cifradores ganadores de la competición, escogidos para llevar a cabo este trabajo. Concretamente, el cifrador AEGIS, uno de los dos algoritmos seleccionados en la categoría “High Performance”, y el cifrador ACORN, segunda de las opciones elegidas en la categoría “Lightweight”.

El Capítulo 4 reúne información sobre la plataforma de desarrollo y el flujo y las herramientas de diseño utilizados para realizar el trabajo. Se detalla también las distintas etapas de síntesis y verificación involucradas en el ciclo de desarrollo, que contemplan la implementación del cifrador AEGIS, su conversión en un módulo IP y su incorporación en un sistema empotrado utilizando diferentes componentes y esquemas de interconexión que permiten establecer distintos compromisos entre el coste de las soluciones y las prestaciones que proporcionan.

Por último, con idea de facilitar el análisis comparativo de los mismos, en el Capítulo 5 se agrupan los resultados obtenidos en las etapas de implementación y verificación de las diferentes soluciones desarrolladas para el cifrador autenticado AEGIS. Dichos resultados corresponden básicamente, a medidas de consumo de los recursos empleado en la FPGA, que proporcionan un parámetro de “coste” de las distintas soluciones, y medidas de los tiempos empleados en realizar la operación asignada, que permiten cuantificar el “rendimiento” de las diferentes alternativas. El capítulo incluye, finalmente, un resumen de los resultados de aplicar el mismo flujo de diseño y test a otro de los cifradores de CAESAR, el cifrador ACORN, con el objetivo de poner de manifiesto el elevado nivel de generalización de la metodología propuesta en el trabajo.

Capítulo 1

Introducción

En este capítulo, en primer lugar se va a describir el contexto que enmarca el estudio que se ha llevado a cabo, comenzando por la disciplina que lo estudia y centrándose en uno de sus campos. Y, en segundo lugar, el capítulo se centrará en el origen de los cifradores autenticados.

1.1. Criptología.

Se define la Criptología como “la disciplina que se dedica al estudio de la escritura secreta, es decir, estudia los mensajes que, procesados de cierta manera, se convierten en difíciles o imposibles de leer por entidades no autorizadas” [1]. Las distintas ramas que componen la criptología se observan en la Figura 1 [2]. El estudio que se va a llevar a cabo en los capítulos posteriores se va a centrar en la criptografía y, más concretamente, en los cifradores simétricos. No obstante es necesario hacer una descripción de cada uno de los campos de la criptología para comprender las diferencias que existen entre ellos. [2]

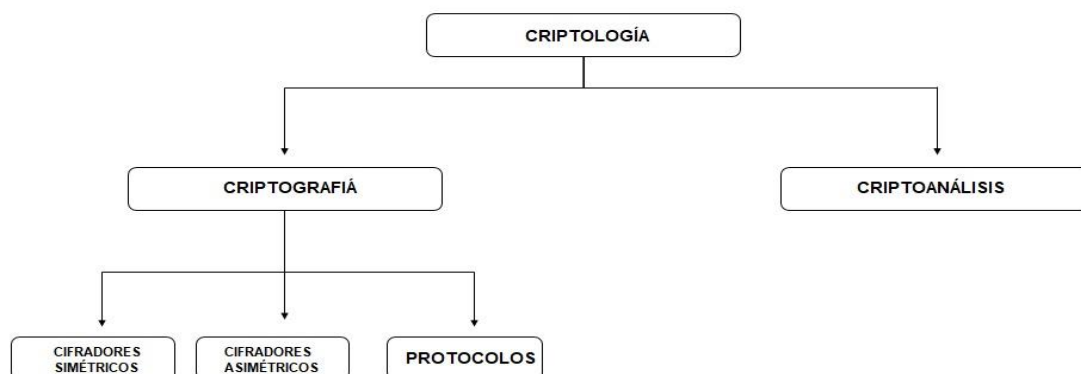


Figura 1. Criptología y sus principales ramas.

En primer lugar, se va a establecer la diferencia entre las dos ciencias que componen la criptología, las cuales son la criptografía y el criptoanálisis. La primera de ellas, la criptografía, se ocupa del estudio de los algoritmos, protocolos y sistemas que se utilizan para proteger la información cuya meta principal consiste en esconder el significado de un mensaje, es decir, cifrar un mensaje. Por otra parte, el criptoanálisis

es la ciencia que se encarga de desmontar cripto-sistemas. En una primera impresión puede parecer que solo se utiliza con fines delictivos, sin embargo es una ciencia usada habitualmente por investigadores de todo el mundo para comprobar si verdaderamente un sistema es seguro o no. Entre los principales tipos de ataques a los cripto-sistemas se encuentran el criptoanálisis clásico, los ataques de implementación y los ataques de ingeniería social. Los dos primeros tipos de ataque son aquellos que los propios diseñadores de los cifradores están encargados de explotar al máximo para conocer cada una de las debilidades que pueda tener su propuesta.

El criptoanálisis clásico es un ataque directo al algoritmo en el que se basa el cifrador tratando de obtener la clave secreta o el propio texto que se está cifrando, ya sea por fuerza bruta (tipo de ataque básico) o por medio de ataques analíticos. Por su parte, el ataque de implementación, consiste en aprovechar canales laterales que se obtienen de la implementación del cifrador para así obtener información útil con la cual se pueda recuperar lo que se denomina “el estado”. Este método tiene que ser estudiado a fondo porque al ser el cifrador de código público (cualquiera puede ver la estructura interna del cifrador, de esta forma otros diseñadores pueden identificar las debilidades del cifrador y ayudar a mejorarlo) en el momento que se detecte una anomalía en el consumo eléctrico del procesador, por ejemplo, como el código es conocido es posible obtener el estado o la clave. Algunos de los canales laterales, además del comentado, pueden ser la radiación electromagnética o los tiempos invertidos en la ejecución del propio algoritmo del cifrador.

El tercer tipo de ataque, basado en ingeniería social, carece de relevancia para el trabajo descrito en esta memoria puesto que no es usado por los diseñadores para la mejora de los cifradores y además se usa con fines delictivos en la mayoría de los casos (sobornos, chantaje, extorsión,...).

1.2. Criptografía. Tipos de cifradores y protocolos

En esta sección se van a introducir brevemente los dos tipos de cifradores que se pueden encontrar, así como los protocolos criptográficos.

A. Cifradores Simétricos.

En primer lugar se encuentran los cifradores simétricos, que consisten en que dos partes tienen un método de cifrado y descifrado y, además, comparten una clave secreta. Hasta el año 1976 la Criptografía se basaba íntegramente en este método. Los cifrados simétricos se siguen usando en la actualidad para el cifrado de datos y comprobación de mensajes. El proceso es el siguiente: se toma un mensaje en forma de bits, este es cifrado por el emisor por medio de un cifrador simétrico, posteriormente es enviado a través de un canal no seguro y finalmente llega al receptor el cual lleva a cabo el proceso de descifrado (proceso inverso). Si se ha utilizado un algoritmo de seguridad elevada el mensaje cifrado en el canal se verá como una sucesión de bits aleatorios carentes de sentido para un tercero que intente interceptar el mensaje. El texto original antes de ser cifrado se denomina “texto plano”, y una vez es cifrado se llama “texto cifrado”. Se puede tener una clave o un conjunto con todas las claves posibles y válidas para descifrar el texto cifrado.

Entre los tipos de cifradores simétricos, se puede a su vez distinguir entre cifradores de flujo y cifradores de bloque. El primer tipo, el cifrador de flujo, cifra los bits individualmente. Esto se consigue añadiendo un bit del *flujo de clave* (en inglés conocido como *KeyStream*) al bit de texto plano. Se pueden encontrar cifradores de flujo síncronos, dependientes únicamente de la clave, y asíncronos, dependientes tanto de la clave como del texto cifrado. Este tipo de cifrador es la base para uno de los algoritmos de estudio en capítulos posteriores, ACORN, que además de cifrar también autentica, es decir, que no solamente cifra el mensaje sino que también acredita o da fe de la autenticidad del mensaje.

El segundo tipo de cifrador simétrico, el cifrador de bloques, cifra un bloque completo de bits del texto plano utilizando la misma clave. En la práctica, la gran mayoría de los cifradores de bloque tienen un tamaño de bloque de 64 o 128 bits. De hecho el algoritmo de cifrado escogido como estándar FIPS en EEUU en 1976, Data Encryption Standard (DES) es un cifrador de bloques de 64 bits [3] y su sustituto, Advanced Encryption Standard (AES) es un cifrador de 128 bits [4].

A continuación se describe brevemente el funcionamiento del AES, ya que otro de los cifradores autenticados escogidos para este trabajo, el algoritmo AEGIS, se basa en las rondas del AES. Cada ronda del AES es una función que se repite en este algoritmo. Estas rondas que componen el AES y el AEGIS usan funciones de transformación de los bytes para cifrar y descifrar. En AES-128, la entrada es un bloque de 128 bits y dicho bloque es copiado a una matriz, denominada array de estado, la cual va a ser modificada en cada una de las transformaciones que se mencionan a continuación.

Por orden de ejecución del cifrado, estas transformaciones son [5]:

- *SubBytes()*: Se denomina SubBytes a la función de sustitución directa de bytes, llevada a cabo durante el proceso de cifrado. Esta función consiste en una simple consulta a una tabla, donde AES define una matriz de 16x16 de valores de un byte denominada S-BOX que alberga una permutación de los 256 valores de 8 bits. Cada byte del estado es mapeado de forma individual a un nuevo byte de la siguiente forma: los 4 bits más a la izquierda del byte serán usados como valores de fila, y los 4 bits de más a la derecha serán usados como valores de columnas. Estos valores de filas y columnas van a servir como índice de la S-BOX para proporcionar una única salida de 8 bits como se muestra en la Figura 2.
- *ShiftRows()*: Una vez realizada la primera transformación, se toma la matriz 4x4 y se lleva a cabo el proceso de desplazamiento de filas. La primera fila, la fila de estado, permanece inalterada mientras que sobre la segunda fila se realiza un desplazamiento de 1 byte hacia la izquierda. Por otra parte, tanto la tercera como la cuarta fila también sufren un desplazamiento hacia la izquierda de valor 2 bytes (para la tercera fila) y 3 bytes (para la cuarta fila) tal como se representa en la Figura 3.

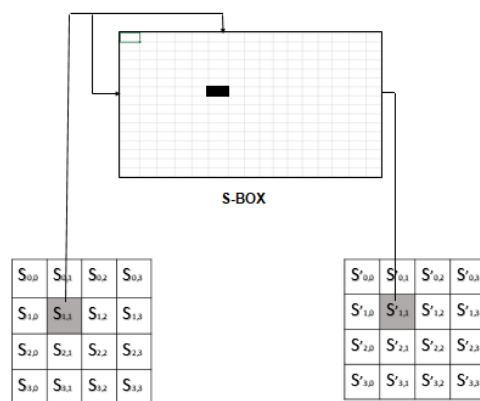


Figura 2. Función de transformación SubBytes().

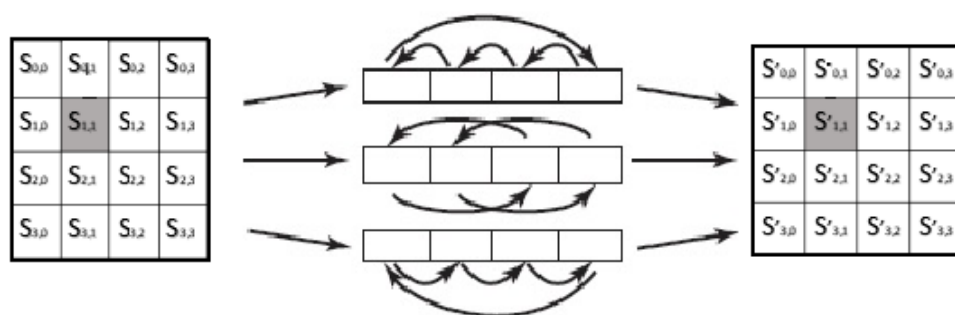


Figura 3. Función de transformación ShiftRows

- *MixColumns()*: Esta función de transformación opera sobre cada columna de la nueva matriz de forma independiente, cada byte de la columna es mapeado a un nuevo valor el cual es función de los 4 bytes de la columna como se ilustra en la Figura 4. Cada elemento en la matriz producto que se muestra en la Figura 4, es la suma de los productos de los elementos de una fila y una columna. La multiplicación y la suma se desarrollan como operaciones de campos finitos de Galois (GF) y estas últimas quedan reducidas a operaciones XOR [2].
- *AddRoundKey()*: Hace una operación XOR bit a bit de los 128 bits de la clave de ronda. En esta función, se realiza una operación columna a columna entre los 4 bytes de la columna de estado y una palabra de la clave de ronda como se aprecia en la Figura 5.

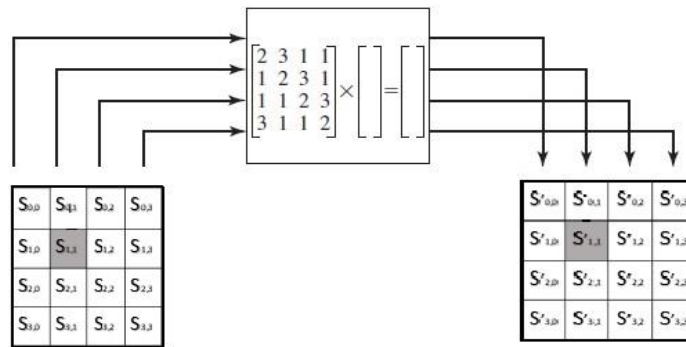


Figura 4. Función de transformación MixColumns().

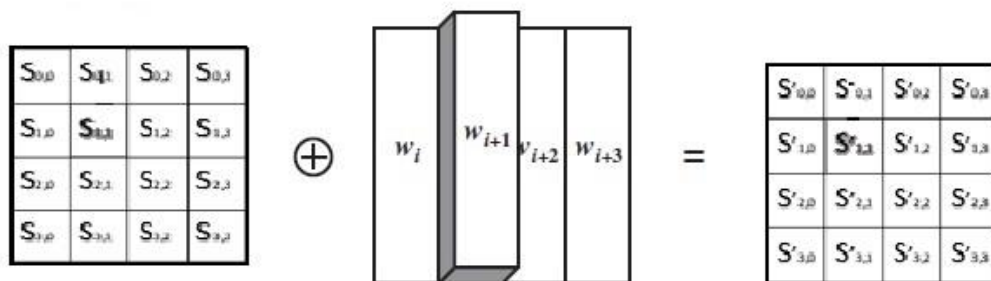


Figura 5. Función de transformación AddRoundKey().

Todas las transformaciones anteriores se usan de forma inversa para el descifrado del texto. Además, existe una ronda preliminar que solo lleva a cabo la transformación AddRoundKey() tal como viene explicado en el documento referenciado de AES [4].

B. Cifradores asimétricos.

En 1976 se introdujo un nuevo método de cifrado por parte de Whitfield Diffie, Martin Hellman and Ralph Merkle. En este método el usuario posee una clave secreta, clave privada, como en el cifrado simétrico y una clave pública. Los algoritmos asimétricos se usan para aplicaciones como la firma electrónica y también para el cifrado clásico de datos. En este último caso, solamente el que posee la clave privada puede cifrar los mensajes, pero cualquiera con la clave pública puede descifrarlos. Esto garantiza que el autor del mensaje cifrado sea quien dice ser y no que cualquiera que tenga la clave compartida como es el caso de los cifradores simétricos pueda cifrar un mensaje sin tener que demostrar su autoría sobre el mismo.

C. Protocolos criptográficos.

Estrictamente hablando, los protocolos criptográficos se encargan de las aplicaciones de los algoritmos descritos. De esta forma, los algoritmos simétricos y asimétricos se pueden ver como bloques de construcción para aplicaciones como la comunicación segura en internet, entre otras.

1.3. El cifrador autenticado

A lo largo de la historia, siempre ha existido la necesidad de proteger mensajes utilizando los medios de los que se disponía, desde simples jeroglíficos en el antiguo Egipto hasta la máquina Enigma usada en la segunda guerra mundial. Pero no fue hasta 1974 cuando el NIST (National Institute of Standards and Technology) publicó una primera petición para establecer un algoritmo de cifrado como estándar [2]: La idea era tener un algoritmo seguro para diversas aplicaciones, puesto que se pretendía usar para la seguridad del país pero también en otras aplicaciones como la banca. Fue así que en 1976 se nombra el algoritmo DES como estándar.

No obstante, la tecnología avanza a toda prisa y se confirmó posteriormente que el DES ya no era seguro (principalmente por el corto tamaño de clave), por lo que era necesario comenzar a buscar nuevas alternativas entre las cuales surgió usar por ejemplo el “triple DES” que consiste en usar el algoritmo tres veces de forma secuencial. Finalmente en el año 2001 el NIST publicó el nuevo estándar avanzado de cifrado, AES, que se puede consultar en “FIPS 197” [4].

Por el momento, AES sigue siendo el estándar, puesto que es un cifrador de bloque bastante seguro y además está disponible para distintos tamaños de bloque. Sin embargo, surgen nuevas inquietudes y se intenta añadir un plus en seguridad, por lo que en 2013 se anunció en Luxemburgo la búsqueda de nuevos algoritmos de cifrado que además puedan autenticar los mensajes. De aquí nace la competición CAESAR, donde todos los diseñadores que quieran, tienen la oportunidad de proponer algoritmos de cifrado y autenticación para medirlos frente a sus oponentes y lograr un listado de algoritmos ganadores. En esta competición, los participantes no solo tendrán que presentar su algoritmo, sino que deberán proporcionar tanto una implementación Software como Hardware completamente funcional.

Es necesario remarcar la gran ayuda que ha proporcionado el proyecto “ATHENA” a esta competición, pues ha proporcionado una serie de utilidades para guiar el desarrollo y test de las diferentes propuestas y ha servido como repositorio donde guardar todas las versiones de implementación hardware de los participantes así como toda la documentación referente a CAESAR. Tanto la competición como el proyecto ATHENA serán explicados en profundidad en el siguiente capítulo.

De los resultados de dicha competición, se puede adelantar que hay varios algoritmos ganadores, debido a que se han evaluado con distintos criterios. La pregunta ahora es, ¿se alzaría alguno de estos algoritmos como un nuevo estándar en el futuro o se seguirán buscando otras alternativas? Actualmente se siguen evaluando los algoritmos pues no ha sido hasta 2019 cuando se han anunciado los ganadores un año después de anunciar los finalistas. Por ello es bastante pronto para especular sobre si alguno puede o no ser un nuevo estándar.

Capítulo 2

Competición CAESAR y proyecto ATHENa

En este capítulo se va a describir, en primer lugar, el desarrollo y funcionamiento de la competición de algoritmos de cifrado y autenticado denominada CAESAR y, en segundo lugar, el proyecto ATHENa el cual ha tenido un papel fundamental en la competición conteniendo información sobre todas las implementaciones VHDL presentadas y facilitando su comparación bajo criterios objetivos. El proyecto ATHENa se va a estudiar como repositorio en primer lugar, y como herramienta en segundo lugar.

2.1. Competición CAESAR

En el año 2013 se anunció en la “Early Symmetric Crypto”, en Mondorf-les-Bains (Luxemburgo), la creación de una nueva competición de cifradores donde se buscaban los algoritmos que además de cifrar y descifrar, pudieran autenticar los mensajes: CAESAR. [6]

La competición constó de varias rondas cada una de las cuales tuvieron unos requerimientos para seguir avanzando en la lid. Los requerimientos principales son la seguridad, aplicabilidad y robustez de los cifradores, y que además, ofrecieran ventajas frente al AES-GCM y fueran apropiados para una adopción generalizada. También cabe destacar que los algoritmos debían ser adecuados para ser utilizados en redes de comunicaciones, protegiendo los paquetes pero manteniendo las cabeceras (datos asociados) de dicho paquete sin cifrar. Algunos de los algoritmos finalistas de esta competición han sido escogidos como ejemplos en este trabajo por lo que van a ser explicados detalladamente en el próximo capítulo.

2.1.1. Requerimientos para la participación

Se presentan a continuación una serie de requisitos que la competición impuso a los participantes los cuales pueden ser consultados en su totalidad en la página web de la competición referenciada al comienzo de este capítulo.

En esta competición, se definió un cifrador autenticado como una función con cinco entradas de cadenas de bytes y una salida de cadena de bytes.

Las cinco entradas son: texto plano con tamaño variable, datos asociados con tamaño variable, número de mensaje secreto de tamaño fijo, número de mensaje público de tamaño fijo, clave de tamaño fijo. La salida es el texto cifrado de tamaño variable.

Las cuatro primeras entradas tienen distintos propósitos de seguridad, todas deben tener integridad mientras que los datos asociados y los mensajes públicos no tienen que ser confidenciales como sí tienen que serlo el resto.

Algunos de los requerimientos funcionales fueron:

- Debe ser posible recuperar el texto plano y el número de mensaje secreto a partir del texto cifrado, datos asociados, clave y mensaje público.
- Los bytes deben ser enteros entre 0 y 255 ambos inclusive. Si los cifradores presentan otro formato, se debe mostrar claramente la relación que tiene dicho formato con las cadenas de bytes.
- Se permite que el cifrador establezca el tamaño máximo de texto plano y datos asociados, pero ninguno de los dos puede ser menor de 65536 bytes. Tanto el tamaño del mensaje como el de los datos asociados puede ser cero.
- Los cifradores deben aceptar como clave, número de mensaje, datos asociados y texto plano toda cadena de bytes que cumpla con los tamaños especificados de la clave y los mensajes, así como los tamaños máximos especificados de los datos asociados y texto plano.

En cuanto a los requerimientos Software, cada propuesta debía estar acompañada por una implementación software de referencia portable a distintas plataformas (CPUs, Oss) para la comprensión pública del cifrador, un criptoanálisis de la propuesta, una verificación de las consecuentes implementaciones, etc... Esta implementación software debe cubrir todos los parámetros recomendados establecidos, y debe llevar a cabo con exactitud la función descrita en la propuesta. Esta implementación se espera optimizada para una mayor claridad, no para rendimiento. Además, se pueden incluir otras implementaciones software que optimicen el rendimiento en cuanto a velocidad.

Los criterios de evaluación oficiales de CAESAR determinaron cuáles eran las implementaciones más rápidas en cada plataforma, de entre todas las propuestas por cada algoritmo, descartando así las implementaciones que se vieron superadas en velocidad por otras implementaciones del mismo algoritmo en la misma plataforma.

En cuanto a los requerimientos Hardware, cada propuesta seleccionada para la segunda ronda y siguientes, debía también presentar una implementación hardware de referencia ya sea en lenguaje VHDL o Verilog.

2.1.2. Requerimientos para las propuestas.

A continuación se detalla la estructura que debía cumplir el documento de cada una de las propuestas participantes. En el próximo capítulo se va a seguir un orden similar para describir los algoritmos seleccionados como ejemplo de este trabajo.

Se establecen una serie de secciones que debieron incluir cada uno de los documentos presentados por parte de los participantes con cada una de las propuestas. Algunas de estas secciones son las siguientes:

- Especificaciones.

Definición completa de la familia de cifradores autenticados. Esta definición debe incluir el espacio de parámetros completo que define la familia, y debe incorporar una lista de un máximo de 10 conjuntos de parámetros recomendados. La definición del cifrador debe ser autosuficiente, conteniendo toda la información necesaria para ser implementado desde cero.

- Metas de seguridad.

Tabla cuantificando para cada parámetro recomendado establecido, el número previsto de bits de seguridad para diversas categorías como son la confidencialidad del texto plano, integridad del texto plano, integridad de los datos asociados, integridad de los mensajes públicos y/o privados que no sean de tamaño 0, etc...

- Análisis de seguridad.

Una explicación de por qué el diseñador esperaba que la familia de cifradores que proponía, con cada uno de los conjuntos de parámetros recomendados, cumplía con los objetivos de seguridad y robustez establecidos en la competición.

- Función:

Lista detallada con las funciones del cifrador sobre las que el diseñador quisiera hacer énfasis. En este apartado también se esperaba que se proporcionaran futuras aplicaciones, además que se especificaran ventajas sobre cifradores previos, etc...

- Justificación del diseño:

Desglose de todas y cada una de las decisiones tomadas en el desarrollo del diseño. Se esperaba además, que se incluyeran los mecanismos necesarios para ocultar lo máximo posible las debilidades del cifrador.

Además, la propuesta se debía acompañar con una portada con el nombre de la misma, los datos de los creadores, y su versión. Se debía acompañar también con un documento sobre la propiedad intelectual de la propuesta y un apartado en el que los participantes prestaban su consentimiento para la participación en la competición.

2.1.3. Resultados de la competición:

La cartera de finalistas de CAESAR ha sido definitivamente propuesta el 20 de Febrero de 2019, y viene clasificada según tres diferentes casos de uso:

1- Aplicaciones Lightweight (para entornos con recursos limitados).

Para esta aplicación, se establecieron dos propuestas finalistas donde un diseño prevalece sobre el otro:

- 1º Ascon, diseñado por Christoph Dobraunig, Maria Eichlseder, Florian Mendel, Martin Schläffer. [7]
- 2º ACORN, diseñado por Hongjun Wu. [8].

2- Aplicaciones “High-Performance” o de alto rendimiento.

Para esta aplicación, se seleccionaron dos propuestas donde ninguna prevalece sobre la otra, por orden alfabético son:

- AEGIS-128, diseñado por Hongjun Wu, Bart Preneel [9].
- OCB, diseñado por Ted Krovetz, Phillip Rogaway [10].

3- Defensa en profundidad.

Para esta aplicación, hubo también dos propuestas finalistas donde un diseño prevalece sobre otro:

- 1º Deoxys-II, diseñado por Jérémy Jean, Ivica Nikolić, Thomas Peyrin, Yannick Seurin [11].
- 2º COLM [12]. Diseñado por Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, Kan Yasuda.

2.2. PROYECTO ATHENa

Como se ha comentado anteriormente, de forma paralela a la competición CAESAR (y a otras competiciones anteriores) se puso en marcha un proyecto que ha dado soporte para el desarrollo hardware de cada una de las propuestas, así como la posibilidad de compararlas con criterios objetivos y consultar los rankings provisionales de cada una de las etapas de la competición. Este proyecto se denomina ATHENa [13] y fue creado en la universidad George Mason (estado de Virginia, EEUU) como una herramienta para la evaluación automatizada de núcleos criptográficos de hardware

dirigidos a FPGAs, SOC's y ASICs. La evaluación que se pretende hacer es lo más justa y completa posible. La página web del proyecto contiene un repositorio con toda la información necesaria acerca de CAESAR y otras competiciones.

El diseño de ATHENa y su metodología para realizar las evaluaciones permite llevar a cabo la comparativa de distintos algoritmos de diferentes competiciones como se mencionó anteriormente y, además, de los propios algoritmos frente a sus versiones anteriores, y de sus implementaciones sobre distintas plataformas hardware. Un claro ejemplo de esto es la comparativa que realizan entre las plataformas hardware Xilinx Virtex 7 y Altera Stratix V. Sin embargo, en el contexto del trabajo descrito en esta memoria, el principal motivo de mención y estudio de este proyecto surge de la implicación del proyecto con la competición CAESAR, en la que se va a centrar este apartado.

En primer lugar, el repositorio web proporciona el “GMU hardware API v1.2”, donde se encuentra toda la documentación y códigos necesarios para llevar a cabo la implementación en VHDL de una interfaz común para todos los cifradores presentados en la competición. A continuación se estudiará este documento en profundidad.

En segundo lugar, el repositorio web proporciona documentación y código HDL tanto de AES como de keccak [14] (de vital importancia en la otra competición descrita en ATHENa, galardonada como nuevo standard SHA-3 [14]). Principalmente suministrando sus diagramas de bloques y cartas ASM.

En otra sección se encuentran los códigos fuente suministrados por los participantes en las rondas segunda y tercera de la competición. Para la segunda ronda se presentan las fuentes RTL (VHDL o Verilog) para la aplicación de tipo 1 (Lightweight), mientras que para la tercera ronda se presentan, además de las anteriores, las fuentes para la aplicación de tipo 2 (High-Performance) para flujos de diseño RTL y/o basados en herramientas de síntesis de alto nivel (HLS).

Por otra parte, se encuentra la sección de evaluación y comparación entre los candidatos de la competición CAESAR. En esta sección se localiza un enlace directo a la página web de la competición, seguido de distintas presentaciones con los diseños y resultados de la tercera ronda así como documentos de interés descritos en el apéndice. También se pueden consultar las tablas que muestran todos los candidatos de la tercera ronda y la base de datos donde se recogen los resultados.

Y por último y no menos importante, en este proyecto se ha desarrollado una herramienta, denominada herramienta ATHENa, encargada de la generación automática de vectores de test para todos los candidatos propuestos en la competición.

La herramienta ATHENa como tal, se encuentra disponible en su web para distintos sistemas operativos donde la versión más reciente es la 0.6.5, publicada en 2014. Incluye, a parte del programa, un tutorial de uso de dicha versión. Por último, se puede consultar en su web los cambios que se han ido produciendo en las distintas versiones de la herramienta, pues han sido muchos desde las primeras versiones las cuales únicamente soportaban FPGAs de Xilinx y Altera con funciones básicas.

2.2.1. GMU Hardware API

En este apartado se va a explicar el contenido del paquete que ofrece ATHENa “GMU Hardware API v1.2” [15]. El paquete se divide principalmente en dos partes, un sub-paquete Hardware y otro Software. No obstante, cabe remarcar que para las implementaciones Hardware y la generación de vectores de test se ha usado la siguiente versión del paquete, “CAESAR_HW_DEVPKG-2.0” [13].

En el paquete Hardware se encuentran los códigos fuente VHDL de los elementos Hardware que dan soporte a la implementación de cifradores autenticados para aplicaciones “Lightweight” y para aplicaciones “High-Performance”. Este paquete ofrece en primer lugar las fuentes RTL que describen las entidades, *PreProcessor* y *PostProcessor*, las cuales se conectan con la entrada y la salida del cifrador. También incluye las plantillas *CipherCore*, *AEAD_Core* y *AEAD_Core_Ent*, donde los diseñadores tienen que introducir su cifrador (*CipherCore*), montar el bloque “pre-cifrador-post” (*AEAD_core_Ent*) y por último encapsular la implementación completa “pre-cifrador-post” (*AEAD_Core*).

Tras las fuentes RTL, se encuentran las fuentes de simulación que consisten en un único test-bench mostrando el comportamiento que debe tener cada una de las señales y salidas de las distintas entidades descritas anteriormente.

En el paquete Software se encuentra el programa AETVgen, que requiere el código C de los distintos algoritmos candidatos, así como sus correspondientes librerías compliadas en el sistema operativo utilizado. Este programa está escrito en Python y permite generar automáticamente vectores de test. Para ello, se encarga de crear los archivos Public Data Input (pdi), Secret Data Input (sdi) y Data Output (do) los cuales serán tratados en el capítulo 4.

En la versión 1.2 del API, el paquete Software también incluye una serie de scripts (bajo el directorio “VivadoBatch”) que permiten automatizar la generación de resultados en Vivado usando el modo batch y una pequeña guía rápida de cómo usarlos.

2.2.1. Fuentes RTL del módulo cifrador.

En esta sección se va a explicar cómo está diseñada la interfaz del módulo de cifrado proporcionada por ATHENa. El esquema global con todas las entradas y salidas se muestra en la Figura 6 y es importante para entender el comportamiento de las entradas y salidas de cada una de las entidades que se van a explicar a continuación: Pre-Procesador, Cifrador y Post-Procesador (conocidas en su conjunto como AEAD Core) [15]. El mecanismo previsto para pasar los datos de entrada al Pre-Procesador y recoger los datos de salida del Post-Procesador, se basa en una serie de FIFOs que facilitan el envío y recepción de los datos.

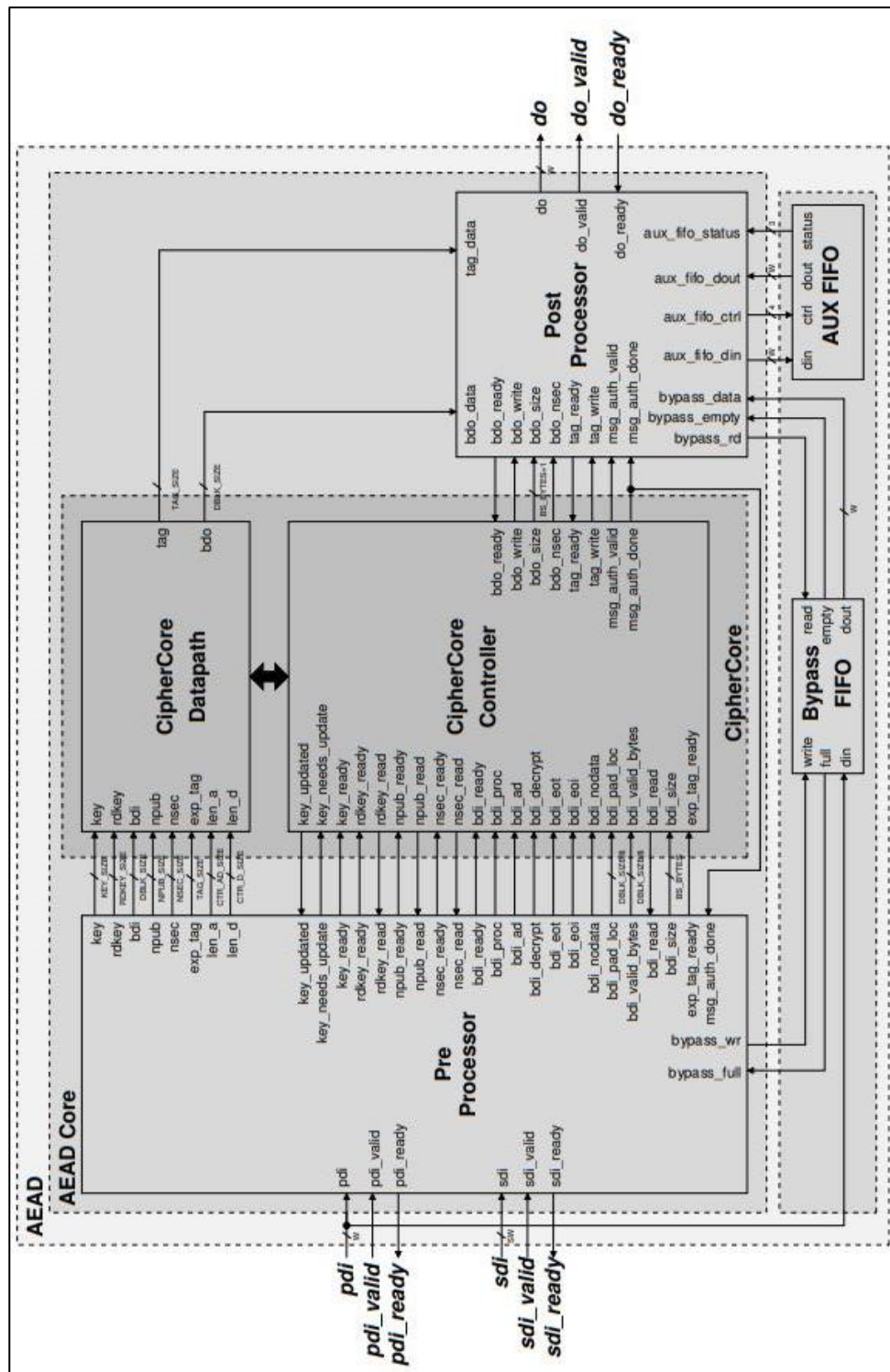


Figura 6. Diagrama de bloques de alto nivel de la interfaz para cifradores dirigidos a aplicaciones de alto rendimiento ("High-Performance").

A. Pre-Procesador.

El Pre-Procesador es la entidad responsable de la ejecución de una serie de tareas comunes para la mayoría de candidatos de CAESAR. Entre estas tareas encontramos: Análisis de las cabeceras de los segmentos de datos que le llega, carga y activación de las claves, conversión serie-paralelo de los bloques de datos de entrada, relleno de los bloques de entrada y control del número de bytes de datos que quedan por procesar. El Pre-Procesador tiene únicamente seis entradas, tres entradas para los datos públicos y otras tres para los datos privados tal y como se muestra en la Figura 7.

Estas entradas son:

-pdi: Bloque de datos públicos de entrada que van a pasar al cifrador.

-pdi_valid y pdi_ready: Señales de protocolo para controlar el envío de la información por medio de PDI FIFO. La secuencia es la siguiente: La FIFO, que actúa en este caso como emisor, comunica si está vacía o llena de datos públicos. Si está vacía, la entrada *pdi_valid* permanece a 0 y no se envía nada; pero si contiene datos para enviar al cifrador, esta señal sube a 1 y si el receptor está listo (tiene activa la señal *pdi_ready*) se produce la transmisión de los datos.

-sdi: Bloque de datos privados de entrada que van a pasar al cifrador.

-sdi_valid y sdi_ready: Señales de protocolo para controlar el envío de la información por SDI FIFO. La secuencia de operación es idéntica a la del caso anterior.

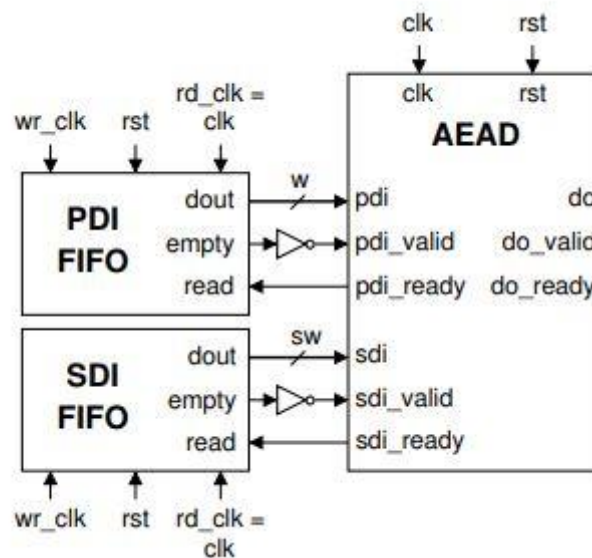


Figura 7. Entrada de datos públicos y secretos en el Pre-Procesador.

El núcleo del Pre-Procesador es muy parametrizable, esto se consigue usando definiciones de parámetros tipo "Generic" en VHDL. Estos parámetros se usan para modificar, por ejemplo, la anchura de los puertos pdi y sdi, el relleno de los datos asociados y el mensaje, etc.

B. Cifrador.

Esta entidad, *CipherCore*, es la que los participantes de la competición deben desarrollar manteniendo una serie de normas que se van a explicar a continuación. Este bloque está diseñado de tal forma para que todos los cifradores que cumplan dichas normas vayan a tener una perfecta compatibilidad con las entidades Pre-Procesador y Post-Procesador. Los diseñadores tendrán que desarrollar el *CipherCore Datapath* y el *CipherCore Controller*. Para llevar a cabo el desarrollo del *Datapath*, cada diseñador puede utilizar la metodología que prefiera, la más sencilla consiste en definir un diagrama de bloques y posteriormente trasladarlo a un lenguaje de descripción Hardware, VHDL o Verilog. Por su parte, el *CipherCore Controller* es descrito usando una carta ASM o un diagrama de estados. Dicha carta ASM suele tener los siguientes estados: primer estado de carga y activación de la clave, segundo estado de procesado de los datos asociados, tercer estado de procesado del texto a cifrar o descifrar y por último el estado de generación de etiqueta de autenticación.

En la primera etapa, la de carga y activación de la clave, se opera de la siguiente manera: Tras un reset inicial, las variables *bandera key_needs_update* y *key_ready* están desactivadas y entonces en cualquier momento una clave puede ser cargada desde el Pre-Procesador. Una vez introducida la clave por el puerto de entrada *sdi* se pondrá a 1 la bandera *key_ready*. Por otra parte, si se recibe la instrucción *ACTIVATE_KEY* por el puerto *pdi*, entonces *key_needs_update* se activa. Estos dos procesos pueden ocurrir en un orden arbitrario.

Después de que *key_needs_update* y *key_ready* estén a 1 y el cifrador se encuentre en el periodo entre el reset y la primera entrada, o bien, en el periodo entre dos entradas consecutivas, el cifrador debería poder leer la clave. Una vez leída, *key_updated* se pone a 1 (deberá ponerse a 0 una vez haya finalizado el procesado de la entrada actual). Si el usuario quiere usar la misma clave para la siguiente secuencia de datos de entrada, se puede omitir la instrucción *ACTIVATE_KEY* y el procesamiento de nuevos datos se iniciará tan pronto como se decodifique una instrucción que describa la forma de procesar una nueva entrada (que se indica con *bdi_proc* a 1). Si no se produce ninguno de los casos descritos, el cifrador permanece en el mismo estado.

En la segunda etapa, la etapa de procesado de los datos asociados, el cifrador se mantiene a la espera de un nuevo bloque de datos asociados hasta que *bdi_eot* se ponga a 1. Esta señal es la indicadora de que el bloque actual es el último bloque de datos asociados recibido. La máquina de estados necesita procesar este bloque para seguir adelante con el resto de los estados, responsables del cifrado y descifrado de los datos. Si el primer bloque leído por el cifrador no es del tipo de datos asociados, se asume que el bloque de datos asociados está vacío (*bdi_ad* = 0). También puede ocurrir que el bloque de datos asociados sea el último bloque recibido de entrada, entonces el mensaje se asume que está vacío.

En la tercera etapa, procesado del texto a cifrar o descifrar, se opera de la misma forma que en la segunda etapa, hasta que el último bloque de texto se haya recibido y procesado. En esta etapa, el puerto *bdi_ad* (*block_data_input_Associated_data*) estará desactivado para no recibir datos asociados. Una vez terminado el procesado, un bloque de datos debe ser enviado al Post-Procesador por medio del puerto *bdo* (*block_data_output*) acompañado de la bandera de salida *bdo_write* activa.

En la última etapa, la de generación de una etiqueta de verificación, el *CipherCore* genera una etiqueta de autenticación nueva, en el caso que el cifrador autenticado esté

cifrando la envía al Post-Procesador a través del puerto *tag* activando la bandera *tag_write*. En el caso que el cifrador esté descifrando un texto, se activa la bandera *msg_auth_done* y el puerto *msg_auth_valid* se activa si la etiqueta generada coincide con la que se ha proporcionado como entrada. La descripción de cada una de las entradas y salidas está explicadas en profundidad en el apéndice de [15].

C. Post-Procesador.

El Post-Procesador es responsable de las siguientes tareas: eliminación de cualquier porción de bloque de salida que no pertenezca ni al texto cifrado ni al texto plano, conversión paralelo-serie de los bloques de salida en palabras de datos, transformación de dichas palabras de datos en segmentos de datos, almacenamiento de los bloques descifrados en un FIFO auxiliar hasta que el resultado de la autenticación sea conocido y por último, generación del bloque de estado con el resultado de la autenticación.

El Post-Procesador tiene únicamente 3 salidas, como se muestra en la Figura 8 y se detallan a continuación:

- *do*: *data output*, el puerto de salida por donde van saliendo los datos procesados.
- *do_ready* y *do_valid*: Señales de protocolo para controlar la recepción de la información por DO FIFO. La FIFO, que actúa en este caso como receptor, comunica si está vacía o llena mediante la señal *do_ready*. El bloque que contiene al cifrador, espera a que esta señal suba a 1 y, si *do_valid* está también activa, inicia la transmisión de datos.

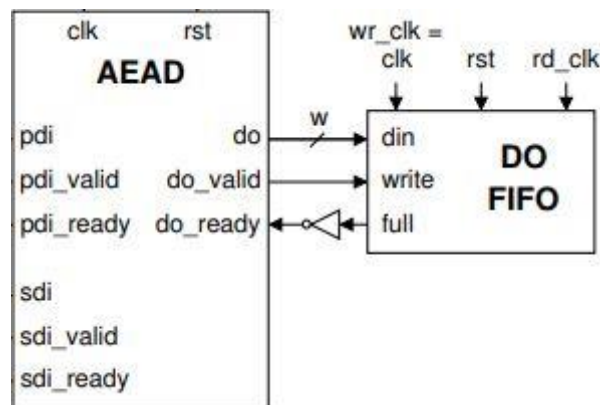


Figura 8. Salida de datos del Post-Procesador

Al igual que el Pre-Procesador, el núcleo del Post-Procesador es parametrizable usando los parámetros del tipo “Generics” en VHDL. De esta forma se puede modificar por ejemplo el tamaño del bloque de texto cifrado y la etiqueta de verificación, así como también ordenar los segmentos de datos de forma determinada para algunos cifradores en particular.

2.2.2. Protocolo de comunicación.

En esta sección se va a detallar qué campos tiene cada una de las instrucciones que llegan al Pre-Procesador, tanto de datos públicos como de datos privados. El apartado explica también cómo se generan los ficheros *pdi*, *sdi* y *do*.

En la Figura 9 [15], se muestran las posibles entradas y salidas de un cifrador autenticado que utiliza la interfaz proporcionada por ATHENa. La entrada *Npub* indica el número de mensaje público, como el nonce o vector de inicialización. *Nsec* denota el número de mensaje secreto, el cual fue introducido recientemente en algunos cifradores autenticados. Se supone que tanto *Npub* como *Nsec* son únicos para cada mensaje cifrado utilizando una clave determinada. La diferencia es que *Npub* es enviado en su forma original mientras que *Nsec* es enviado cifrado. Todas las entradas que no sean la clave son opcionales, y por tanto se pueden omitir. Si esto ocurre, dicha entrada se considera una cadena vacía.

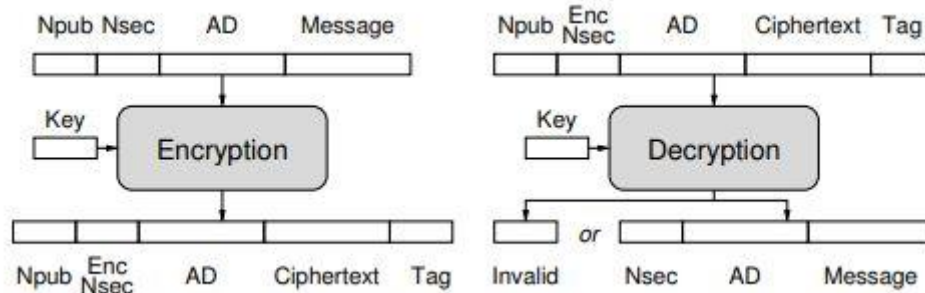


Figura 9. Entrada y salida de un cifrador autenticado.

El formato propuesto para los datos secretos de entrada (*sdi*) se muestra a continuación en la Figura 10:



Figura 10. Formato de *sdi* para a) Cargar clave principal, b) Cargar secuencia de rondas de clave.

La entrada comienza con una instrucción, LDKEY (cargar la clave) y LDRKEY (carga de ronda de clave). La instrucción es seguida de segmentos, cada segmento tiene una cabecera (*header*) describiendo su tipo y tamaño. En el caso de *sdi* solo se permiten dos tipos de segmentos, *Key* y *Round Key*, que habrán sido computados previamente en Software, en el caso de Round Key se comienza con la ronda de índice

más bajo. La Figura 11 ha sido extraída directamente del documento “sdi.txt” generado a través del software AETVgen que se explicará más adelante:

```
#### MsgID= 1, KeyID= 1
# Instruction: Opcode=Load Key
INS = 40000000
# Info :                      Key, EOI=1 EOT=1, Last=1, Length=16 bytes
HDR = C7000010
DAT = 55565758595A5B5C5D5E5F6061626364
```

Figura 11. Ejemplo del formato de instrucción para SDI.

El formato propuesto para los datos públicos de entrada (*pdi*) se muestra a continuación en la Figura 12.

:

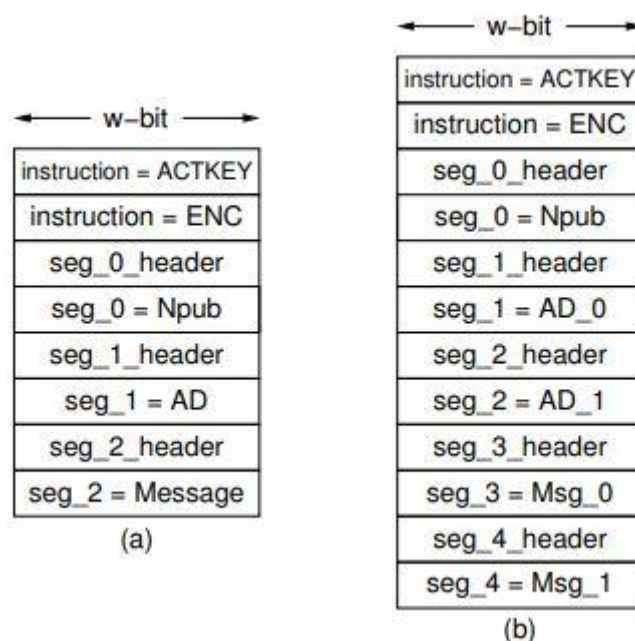


Figura 12. Formato de datos públicos de entrada para los casos a) un único segmento de cada tipo de datos, b) múltiples segmentos de mensajes y datos asociados.

Las instrucciones permitidas para este formato son: ACTKEY (activación de la clave), ENC (Cifrado autenticado) y DEC (Descifrado autenticado). La instrucción de activación de la clave normalmente precede a las otras dos instrucciones. Tras las instrucciones se encuentran una serie de segmentos, que pueden ser del tipo *Npub*, *Nsec*, *AD* (datos asociados) y mensaje. Durante el cifrado también se permite el tipo Enc Nsec (Número de mensaje secreto cifrado), texto cifrado y etiqueta de autenticación.

La principal razón de separar mensaje y datos asociados en segmentos distintos es que a priori no se tiene por qué conocer el tamaño exacto de cada uno de ellos. Un ejemplo de este tipo de formato es el ilustrado, a continuación, en la Figura 13:


```

#### Authenticated Encryption
#### MsgID= 1, KeyID= 1 Ad Size = 0, Pt Size = 0
# Instruction: Opcode=Activate Key
INS = 7000000000000000000000000000000000000000000000000000000000000000
# Instruction: Opcode=Authenticated Encryption
INS = 2000000000000000000000000000000000000000000000000000000000000000
# Info :                               Npub, EOI=1 EOT=1, Last=0, Length=16 bytes
HDR = D600000100000000000000000000000000000000000000000000000000000000
DAT = B0B1B2B3B4B5B6B7B8B9BABBBBCBDBEBF00000000000000000000000000000000
# Info :                               Associated Data, EOI=0 EOT=1, Last=0, Length=0 bytes
HDR = 1200000000000000000000000000000000000000000000000000000000000000
# Info :                               Plaintext, EOI=0 EOT=1, Last=1, Length=0 bytes
HDR = 4300000000000000000000000000000000000000000000000000000000000000

```

Figura 13. Ejemplo del formato de instrucción para *PDI*.

En el documento referenciado en esta sección se puede consultar más a fondo cómo es el formato para crear tanto las instrucciones (INS) como las cabeceras (HDR).

2.2.3. Generación de vectores de test.

Esta sección va a explicar la metodología con la cual se generan los ficheros *pdi*, *sdi* y *do* mencionados repetidas veces para llevar a cabo una serie de tests y así verificar que el cifrador autenticado funciona correctamente.

Concretamente se puede llevar a cabo el proceso desde un sistema operativo Windows o un sistema operativo Linux con la herramienta “Cygwin”. Tras instalar todos los componentes necesarios, se deben seguir una serie de pasos de ejecución del programa contenido en “*aeadtngen*” (disponible en el paquete de “*CAESAR_HW_DEVPKG-2.0*”) mediante los cuales se van a poder configurar distintos tamaños de ficheros generados y, además, distintos modos de creación: aleatorio, pseudo-aleatorio, personalizado, etc... Todos los pasos a seguir para llevar a cabo la creación de los ficheros se encuentran en [15, p. APENDIX E].

Capítulo 3

Cifradores autenticados

En este capítulo se van a estudiar en profundidad algunos de los algoritmos finalistas de la competición CAESAR, el algoritmo AEGIS y el algoritmo ACORN.

En primer lugar, el capítulo se centrará en el algoritmo AEGIS el cual ha finalizado la competición compartiendo el primer puesto junto a otro algoritmo, el OCB, para aplicaciones de alto rendimiento. La versión de AEGIS ganadora es la versión AEGIS-128, aunque sus diseñadores han presentado junto a esta las versiones AEGIS-128L y AEGIS-256.

En segundo lugar, el capítulo se centrará en el algoritmo ACORN, el cual ha finalizado la competición en segunda posición por detrás de ASCON para la aplicación lightweight.

3.1. AEGIS

3.1.1. Introducción

El algoritmo AEGIS lo propusieron Hongjun Wu y Bart Preneel. La propuesta definitiva, después de las distintas rondas de la competición CAESAR, la presentaron en la memoria AEGIS v1.1 [9], que recopila todos los documentos requeridos por la competición.

Para mantener la confidencialidad y autenticidad del mensaje los algoritmos se pueden enfocar de dos formas distintas. El primer enfoque consiste en cifrar y autenticar separadamente, esto es, cifrando el texto plano bien con un cifrado de bloques o con uno de flujo y después autenticando el texto cifrado con un algoritmo MAC (Message Authentication Code).

El segundo enfoque consiste en aplicar un único algoritmo de cifrado y autenticación del mensaje. En el caso de AEGIS, los autores se decantan por este segundo método.

El algoritmo presentado está basado en una función que actualiza un estado y que devuelve el texto cifrado tras una serie de actualizaciones del estado y después de otra serie de actualizaciones del estado más, devuelve la etiqueta de autenticado. Esto hace que las dos operaciones del algoritmo compartan la carga computacional. El algoritmo AEGIS toma como función base para la actualización del estado las rondas del AES.

Una de las condiciones para que el algoritmo AEGIS proporcione un gran nivel de seguridad es que el nonce¹ no se re-use.

El algoritmo AEGIS lo propusieron en tres versiones diferentes: AEGIS-128, AEGIS-128L y AEGIS-256, según la versión se emplea un número de rondas de AES distinto: AEGIS-128L requiere de 8 rondas de AES para procesar un bloque de mensaje de 32 bytes, por otra parte, AEGIS-128 usa 5 rondas de AES para procesar un bloque de mensaje de 16 bytes; y por último, AEGIS-256 usa 6 rondas de AES. El coste computacional de AEGIS es aproximadamente la mitad que AES (en lo que a velocidad se refiere).

3.1.2. Especificaciones

A. Parámetros recomendados.

Para AEGIS-128L, se establece usar una clave de 128 bits, un nonce de 128 bits, un estado de 1024 bits y finalmente, una etiqueta de autenticación de 128 bits. Es la versión más rápida de todos los algoritmos de AEGIS, por lo tanto los autores la propusieron para aplicaciones de alto rendimiento.

Para AEGIS-128, se establece usar una clave de 128 bits, un nonce de 128 bits, un estado de 640 bits y una etiqueta de autenticación de 128 bits. Esta versión también la propusieron para aplicaciones de alto rendimiento.

Para AEGIS-256, se establece una clave de 256 bits, un nonce de 256 bits, un estado de 768 bits y una etiqueta de autenticación de 128 bits.

B. Operaciones.

Las operaciones básicas que utiliza el algoritmo AEGIS son las siguientes:

- Exclusive OR bit-a-bit.
- AND bit-a-bit.
- Concatenación.
- $[x]$ Función límite, donde $[x]$ es el entero más pequeño no inferior a x .

C. Variables y constantes.

Para describir el algoritmo se definen las siguientes variables y constantes:

- ad : Dato asociado (este dato es procesado por el algoritmo, pero no va a ser cifrado-descifrado).
- adi : Bloque de 16-bit de datos asociados (el último bloque puede ser un bloque parcial).

¹ Se conoce nonce como número aleatorio creado para la verificación del proceso de cifrado usado una sola vez.

- adlen: número de bits de los datos asociados, $[0 \leq \text{adlen} \leq 2^{64}]$.
- c: Texto Cifrado
- ci: Bloque de 16-bit de texto cifrado (el último bloque puede ser un bloque parcial).
- const: constante de 32-bit en formato hexadecimal; const es igual a la sucesión de Fibonacci en módulo 256.
- const0: primeros 16-bits de const.
- const1: últimos 16-bit de const.
- IV128: vector de 128-bits de inicialización del AEGIS-128.
- IV256: vector de 256-bits de inicialización del AEGIS-256.
- IV256,0: Primeros 128 bits de IV256.
- IV256,1: Segunda mitad de IV256.
- K128: Clave de 128-bits del AEGIS-128.
- K256: Clave de 256-bits del AEGIS-256.
- K256,0: Primera mitad de K256.
- K256,1: última mitad de K256.
- msglen: longitud en bits del texto plano o del texto cifrado. $[0 \leq \text{msglen} \leq 2^{64}]$.
- mi: Bloque de datos de 16-bit.
- P: texto plano.
- pi: bloque de texto plano de 16-bits
- S_i : Estado en el inicio del paso i .
- $S_{i,j}$: Elemento j de 16-bits del estado S_i . Para el AEGIS-128, $0 \leq j \leq 4$; para AEGIS-256, $0 \leq j \leq 5$ y para AEGIS-128L, $0 \leq j \leq 7$.
- T: Etiqueta de autenticación.
- t: tamaño de la etiqueta de autenticación.
- u: $u = \text{adlen}/128$.
- v: $v = \text{msglen}/128$.
- uL: $uL = \text{adlen}/256$.
- vL: $vL = \text{msglen}/256$.

D. Funciones.

Como se explicó anteriormente, AEGIS utiliza las rondas de AES siguiendo la siguiente nomenclatura:

AESRound(A,B): siendo A el estado de 16 bits y; B la clave de 16 bits para la ronda.

Esta función mapea dos entradas y dos salidas de 16-bits cada una. Para implementaciones software en procesadores x86 se pueden implementar con la instrucción: **AES_m128_aesenc_si128(A, B)**.

3.1.3. AEGIS-128

A continuación, se van a explicar los aspectos más importantes de la versión AEGIS-128 ya que ha sido la seleccionada como finalista para la aplicación de alto rendimiento.

A. Función de actualización de estado.

Esta función es la función principal de este algoritmo. La función $\text{StateUpdate128}(S_i, m_i)$ actualiza el estado S_i de 80 bytes con un bloque de mensaje m_i de 16 bytes tal como se describe y se muestra en la Figura 14.

Función $S_{i+1} = \text{StateUpdate128}(S_i, m_i)$:

$S_{i+1,0} = \text{AESRound}(S_{i,4}, S_{i,0} \text{ XOR } m_i)$;

$S_{i+1,1} = \text{AESRound}(S_{i,0}, S_{i,1})$;

$S_{i+1,2} = \text{AESRound}(S_{i,1}, S_{i,2})$;

$S_{i+1,3} = \text{AESRound}(S_{i,2}, S_{i,3})$;

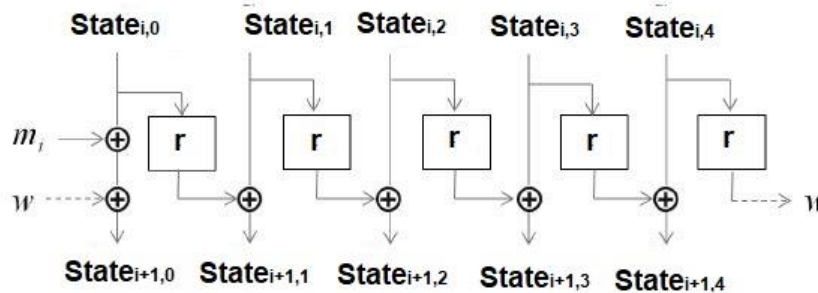


Figura 14. Función de actualización de estado de AEGIS-128.

B. Inicialización de AEGIS-128.

Consiste en cargar la clave y el vector de inicialización en el estado inicial, ejecutar la función de actualización diez pasos usando la clave y el vector de inicialización como mensaje. Esto descrito sería:

Se inicializa el estado usando la clave, el vector de inicialización y la constante const de la siguiente forma:

- 1- $S_{-10,0} = K_{128} \text{ XOR } IV_{128};$
 $S_{-10,1} = \text{const}_1;$
 $S_{-10,2} = \text{const}_0;$
 $S_{-10,3} = K_{128} \text{ XOR } \text{const}_0;$
 $S_{-10,4} = K_{128} \text{ XOR } \text{const}_1;$
2. Desde $i = -5$ hasta $i = -1$, el mensaje se define como:
 $m_{2i} = K_{128}; m_{2i+1} = K_{128} \text{ XOR } IV_{128};$
3. Desde $i = -10$ hasta $i = -1$, el estado se actualiza:
 $S_{i+1} = \text{StateUpdate128}(S_i, m_i).$

C. Procesado de los datos asociados.

Tras la inicialización, los datos asociados AD se usan para la actualización del estado como se describe a continuación:

1. Si el último bloque de datos asociados es incompleto, se usan bits a 0 hasta llegar a 128 y este bloque completo se usa para actualizar el estado. Si $\text{adlen}=0$, no hay datos asociados que procesar, por lo que no se actualiza el estado.
2. Desde $i=0$ hasta $i= \text{adlen}/128 - 1$ se actualiza el estado:
 $S_{i+1} = \text{StateUpdate128}(S_i, \text{AD}_i);$

D. Proceso de cifrado del cifrador.

Tras el procesado de los datos asociados AD, se procesa el texto plano dividido en bloques de 16 bytes. En cada paso del cifrado, un bloque de texto plano de 16 bytes, P_i , se usa para actualizar el estado, y P_i es cifrado y pasa a ser C_i .

1. Si el último bloque de texto plano no está completo, se usan bits a 0 hasta completar los 128 bits y que serán usados para actualizar el estado. En el caso del cifrado del último bloque, solo el texto plano original va a ser cifrado (no se cifran los ceros añadidos para la actualización del estado). En el caso de que sólo haya datos asociados, y no mensaje que cifrar, esto es, $\text{msglen} = 0$, no habrá ni actualización del estado ni cifrado.
2. Se toma $u = \text{adlen}/128$ y $v = \text{msglen}/128$.

Desde $i = 0$ hasta $i = v-1$ se lleva a cabo la actualización del estado y el cifrado de la siguiente forma:

$$C_i = P_i \text{ XOR } S_{u+i,1} \text{ XOR } S_{u+i,4} \text{ XOR } (S_{u+i,2} \text{ AND } S_{u+i,3});$$

$$S_{u+i+1} = \text{StateUpdate128}(S_{u+i}, P_i);$$

E. Finalización de AEGIS-128.

Tras haber cifrado todos los bloques de texto plano, se genera una etiqueta de autenticación realizando siete pasos adicionales. En estos pasos adicionales, primero se establece una variable que usa la longitud del mensaje y de los datos asociados y después se realizan seis actualizaciones de estado y por último se genera la etiqueta de autenticación.

1. Se impone $tmp = S_{u+v,3} \text{ XOR } (adlen \parallel msglen)$, donde $adlen$ y $msglen$ están representados como enteros de 64 bits.
2. Desde $i = u + v$ hasta $i = u + v + 6$, se actualiza el estado de la siguiente forma:

$$S_{i+1} = \text{StateUpdate128}(S_i, tmp);$$

3. Se genera una etiqueta de autenticación a partir del estado S_{u+v+7} de la siguiente forma:

$$T' = S_{u+v+7,0} \text{ XOR } S_{u+v+7,1} \text{ XOR } S_{u+v+7,2} \text{ XOR } S_{u+v+7,3} \text{ XOR } S_{u+v+7,4}.$$

La etiqueta de autenticación T se compone de los t primeros bits de T' .

F. Descifrado y autenticación de AEGIS-128.

El descifrado comienza con la inicialización y procesamiento de los datos autenticados. Tras esto, el texto cifrado es descifrado de la siguiente forma:

1. Si el último bloque cifrado no es un bloque completo, sólo se descifra la parte que contiene los datos cifrados. El bloque parcial de texto plano es rellenado a 0 y el bloque completo de texto plano se usa para actualizar el estado.
2. Desde $i = 0$ hasta $i = v-1$, se lleva a cabo el descifrado y actualización del estado:

$$P_i = C_i \text{ XOR } S_{u+i,1} \text{ XOR } S_{u+i,4} \text{ XOR } (S_{u+i,2} \text{ AND } S_{u+i,3}).$$

$$S_{u+i+1} = \text{StateUpdate128}(S_{u+i}, P_i).$$

Como se observa, los procesos de cifrado y descifrado son exactamente igual (P_i y C_i se pueden intercambiar). Es importante destacar que si la verificación falla, el texto cifrado y la etiqueta de autenticación creada no deben proporcionarse como salida, de lo contrario el estado de AEGIS-128 es vulnerable a ataques de texto plano conocido y ataques de parte del texto cifrado elegido (usando un vector de inicialización fijo).

3.2. ACORN

3.2.1. Introducción

En esta sección se van a describir las características más importantes del algoritmo ACORN que fue propuesto por Hongjun Wu para la competición CAESAR y terminó como finalista para aplicaciones “Lightweight”. Los documentos requeridos por la competición se recopilan en la memoria denominada Acorn v3 [8] después de pasar todas las rondas de la misma. Como principal diferencia con respecto a AEGIS se puede destacar que este algoritmo es un cifrador de flujo mientras que AEGIS lo es de bloque.

3.2.2. Especificaciones

A. Parámetros recomendados.

Las recomendaciones por parte de los diseñadores para el cifrador son: Tomar una clave de 128 bits, tomar un nonce de 128 bits y una etiqueta de verificación también de 128 bits.

B. Operaciones.

Las operaciones que utiliza son:

- Exclusive OR bit-a-bit.
- NOT bit-a-bit.
- AND bit-a-bit.
- Concatenación.

C. Variables y constantes.

Este algoritmo comparte las siguientes variables y constantes con AEGIS: AD, adi, adlen, C, IV₁₂₈, K₁₂₈, K_{128,i}, mi, P, Si, Si,j, T y t.

Y además añade las que se detallan a continuación:

- ci: Bit i del texto cifrado.
- cai: Bit de control del paso i. Se usa para separar el procesado de los datos asociados, procesado del texto plano a cifrar y de la etiqueta de autenticación generada.
- cbi: Otro bit de control del paso i. Se usa para permitir al flujo de claves usarse como realimentación en la inicialización, procesado de los datos asociados o en la generación de la etiqueta de verificación.
- ksi: Bit del flujo del clave generado en el paso i.

- p_{len} : Tamaño en bits del texto plano o del texto cifrado comprendido entre 0 y 2^{64} .
- m_i : Bit de datos.
- p_i : Bit i del texto plano.

D. Funciones.

El algoritmo ACORN usa dos funciones booleanas, “maj” y “ch”:

$$\text{maj}(x,y,z) = (x \text{ AND } y) \text{ XOR } (x \text{ AND } z) \text{ XOR } (y \text{ AND } z).$$

$$\text{ch}(x,y,z) = (x \text{ AND } y) \text{ XOR } ((\text{NOT } x) \text{ AND } z).$$

3.2.3. ACORN-128

A. Estado del ACORN-128.

El tamaño del estado del ACORN-128 es 293 bits. Hay seis LFSRs concatenados que se mostrarán en la Figura 15 [8].

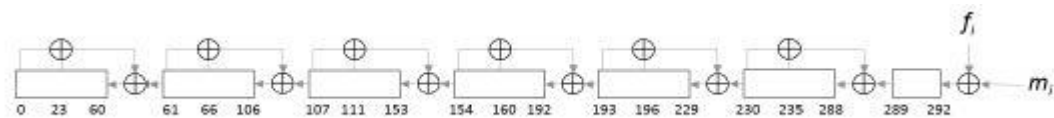


Figura 15. Concatenación de los 6 LFSRs. f_i indica el bit general de realimentación para el paso i ; m_i indica el bit de mensaje para el paso i .

B. Funciones de ACORN-128.

Hay tres funciones en el ACORN-128: La función de generación del flujo de clave a partir del estado, la función para computar el bit de realimentación global, y la función de actualización del estado.

- Generación del bit del flujo de clave: En cada paso, el bit del flujo de clave es computado usando la función $ksi = \text{KSG128}(S_i)$.

$$ksi = S_{i,12} \text{ XOR } S_{i,154} \text{ XOR } \text{maj}(S_{i,235}, S_{i,61}, S_{i,193}) \text{ XOR } \text{ch}(S_{i,230}, S_{i,111}, S_{i,66}).$$

- Cálculo del bit de realimentación: En cada paso, el bit de realimentación es calculado usando la función $f_i = \text{FBK128}(S_i, cai, cbi)$.

$$f_i = S_{i,0} \text{ XOR } (\sim S_{i,107}) \text{ XOR } \text{maj}(S_{i,244}, S_{i,23}, S_{i,160}) \text{ XOR } (cai \& S_{i,196}) \text{ XOR } (cbi \& ksi).$$

- Función de actualización del estado: En cada paso, el pseudo-código para la función de actualización de estado $S_{i+1} = \text{StateUpdate128}(S_i, m_i, c_{ai}, c_{bi})$ es el siguiente:
 - Paso 1: Usando los seis LFSRs:

$$S_{i,289} = S_{i,289} \text{ XOR } S_{i,235} \text{ XOR } S_{i,230};$$

$$S_{i,230} = S_{i,230} \text{ XOR } S_{i,196} \text{ XOR } S_{i,193};$$

$$S_{i,193} = S_{i,193} \text{ XOR } S_{i,160} \text{ XOR } S_{i,154};$$

$$S_{i,154} = S_{i,154} \text{ XOR } S_{i,111} \text{ XOR } S_{i,107};$$

$$S_{i,107} = S_{i,107} \text{ XOR } S_{i,66} \text{ XOR } S_{i,61};$$

$$S_{i,61} = S_{i,61} \text{ XOR } S_{i,23} \text{ XOR } S_{i,0};$$
 - Paso 2: Generando el bit del flujo de clave:

$$K_{Si} = \text{KSG128}(S_i).$$
 - Paso 3: Generando el bit de realimentación no linear:

$$f_i = \text{FBK128}(S_i, c_{ai}, c_{bi}).$$
 - Paso 4: Desplazamiento del registro de 293 bits con el bit de realimentación f_i :

Desde $j = 0$ hasta $j = 291$ hacer $S_{i+1,j} = S_{i,j+1}$;

$$S_{i+1,292} = f_i \text{ XOR } m_i .$$

C. Inicialización de ACORN-128.

La inicialización del ACORN-128 consiste en cargar la clave y el vector de inicialización en el estado, y ejecutando el cifrado 1792 pasos.

1. Se inicializa el estado S_{-1792} a 0.
2. Se establece:

$$m_{-1792+1} = K_{128,i} \text{ desde } i = 0 \text{ hasta } i = 127;$$

$$m_{-1792+128+i} = IV_{128} \text{ desde } i=0 \text{ hasta } i = 127;$$

$$m_{-1792+256} = K_{128,i \bmod 128} \text{ XOR } 1. \text{ Para } i = 0;$$

$$m_{-1792+256+i} = K_{128,i \bmod 128} . \text{ Desde } i = 1 \text{ hasta } i = 1535;$$

3. Se inicializa:

$$c_{a-1792+i} = 1. \text{ Para } i = 0 \text{ hasta } i = 1791;$$

$$c_{b-1792+i} = 1. \text{ Para } i = 0 \text{ hasta } i = 1791;$$

4. Desde $i = 1792$ hasta $i = -1$,
 $S_{i+1} = \text{StateUpdate128}(S_i, m_i, cai, cbi);$

En la inicialización, el bit del flujo de clave es usado para actualizar el estado ya que $cbi = 1$.

D. Procesado de los datos asociados.

Tras la inicialización, los datos asociados AD se usan para actualizar el estado.

1. $m_i = adi$, para $i = 0$ hasta $i = adlen - 1$;
 $m_{adlen} = 1$;
 $m_{adlen+i} = 0$, para $i = 1$ hasta $i = 255$;
2. $cai = 1$ para $i = 0$ hasta $i = adlen + 127$;
 $cai = 0$ para $i = adlen + 128$ hasta $i = adlen + 255$;
 $cbi = 1$ para $i = 0$ hasta $i = adlen + 255$;
3. Para $i = 0$ hasta $i = adlen + 255$,
 $S_{i+1} = \text{StateUpdate128}(S_i, m_i, cai, cbi);$

Incluso cuando no hay datos asociados, se necesita ejecutar el cifrado 256 pasos. Cuando se procesan los datos asociados, el bit del flujo de clave se usa para actualizar el estado ya que $cbi = 1$

E. Cifrado.

Tras procesar los datos asociados, en cada paso del cifrado, un bit del texto plano, pi , es usado para actualizar el estado, y se cifra pasando a llamarse ci .

1. $m_{adlen+256+i} = pi$ desde $i = 0$ hasta $i = pcrlen - 1$.
 $m_{adlen+256+pcrlen} = 1$.
 $m_{adlen+256+pcrlen+i} = 0$ para $i = 1$ hasta $i = 255$.
2. $cai = 1$ para $i = adlen + 256$ hasta $i = adlen + pcrlen + 383$.
 $cai = 0$ para $i = adlen + pcrlen + 383$ hasta $i = adlen + pcrlen + 511$.
 $cbi = 0$ para $i = adlen + 256$ hasta $i = adlen + pcrlen + 511$.
3. Desde:
 $i = adlen + 256$ hasta $i = adlen + pcrlen + 511$,
 $S_{i+1} = \text{StateUpdate128}(S_i, m_i, cai, cbi);$
 $C_{i-adlen-256} = p_{i-adlen-256} \text{ XOR } ks_i;$

Incluso cuando no hay texto plano, se necesita ejecutar el cifrado 256 pasos. Cuando se procesa el texto plano, el bit del flujo de clave no se usa para actualizar el estado ya que $c_{bi} = 0$.

F. Finalización.

Tras procesar todos los bits del texto plano, se genera una etiqueta de autenticación T .

1. $m_{adlen+pclen+512+i} = 0$ para $i = 0$ hasta $i = 767$.
2. $ca_i = 1$ para $i = adlen + pclen + 512$ hasta $i = adlen + pclen + 1279$.
 $c_{bi} = 1$ para $i = adlen + pclen + 512$ hasta $i = adlen + pclen + 1279$.
3. Desde:
 $i = adlen + pclen + 512$ hasta $i = adlen + pclen + 1279$,
 $S_{i+1} = \text{StateUpdate128}(S_i, m_i, ca_i, c_{bi});$

La etiqueta de autenticación T corresponde a los t últimos bits del flujo de clave.

G. Descifrado y verificado.

El descifrado y verificado son muy similares al cifrado y generación de la etiqueta de autenticación. Se hace énfasis en que si la verificación falla, el texto cifrado y la última etiqueta de autenticación generada no deben darse como salida tal y como ocurría en el algoritmo AEGIS, de lo contrario, el estado de ACORN-128 es vulnerable ante los ataques de “texto plano conocido” o “ataque sobre texto cifrado elegido” usando un vector de inicialización fijo.

CAPÍTULO 4

Implementación de cifradores empotrados en SoC

Este capítulo va a describir la plataforma de desarrollo, el flujo y las herramientas de diseño utilizados para realizar el trabajo. Todo ello con el fin de obtener, como ya se definió entre los objetivos del trabajo, implementaciones eficientes de estos cifradores para su uso en sistemas empotrados para aplicaciones reales de comunicación.

También se va a analizar la metodología que se ha usado para poner en marcha el flujo de diseño que permite hacer este estudio factible, el System on Chip (SoC) que se ha usado, el entorno de desarrollo con el cual se han simulado e implementado los cifradores autenticados y las variaciones y adaptaciones que han sido necesarias. Se va a prestar especial dedicación a describir el diseño híbrido Software/Hardware que se ha llevado a cabo con distintas configuraciones de buses de comunicación AXI4 y AXI-Stream.

4.1. Entorno de trabajo

4.1.1. Placa de desarrollo.

La placa de desarrollo con la cual se han llevado a cabo las implementaciones es la Zybo [16]. La placa ZYBO (ZYNq BOard) es una plataforma de desarrollo de circuitos digitales y software embebido de nivel básico, lista para usar y con muchas funciones, creada en torno al miembro más pequeño de la familia Xilinx Zynq-7000, el Z-7010. El Z-7010 se basa en la arquitectura Xilinx All Programmable System-on-Chip (AP SoC), que integra un procesador ARM Cortex-A9 [17] de doble núcleo con lógica programable de las familias de FPGAs (Field Programmable Gate Array) de la serie 7 de Xilinx. La placa de desarrollo se puede visualizar en la Figura 16 mientras sus componentes se detallan en la Tabla 1.

La placa Zybo incorpora un dispositivo Zynq XC7Z010-1GFL400C con las siguientes características:

- Doble procesador Cortex-A9 de 650MHz.
- Controlador de memoria DDR3 con 8 canales DMA.
- Lógica programable equivalente a la de las FPGAs de la familia Artix-7.
 - 4400 slices, cada uno de ellos con 4 LUTs de 6 entradas y 8 flip-flops.
 - 240 KB de RAM.
 - 80 DSPs (para procesamiento digital de señal).
 - Convertidor analógico/digital On-chip (XADC).

También incluye 512MB de memoria DDR3 con 1050 Mbps de ancho de banda, así como un conjunto de botones, conmutadores y leds, y diferentes puertos de entrada y salida.

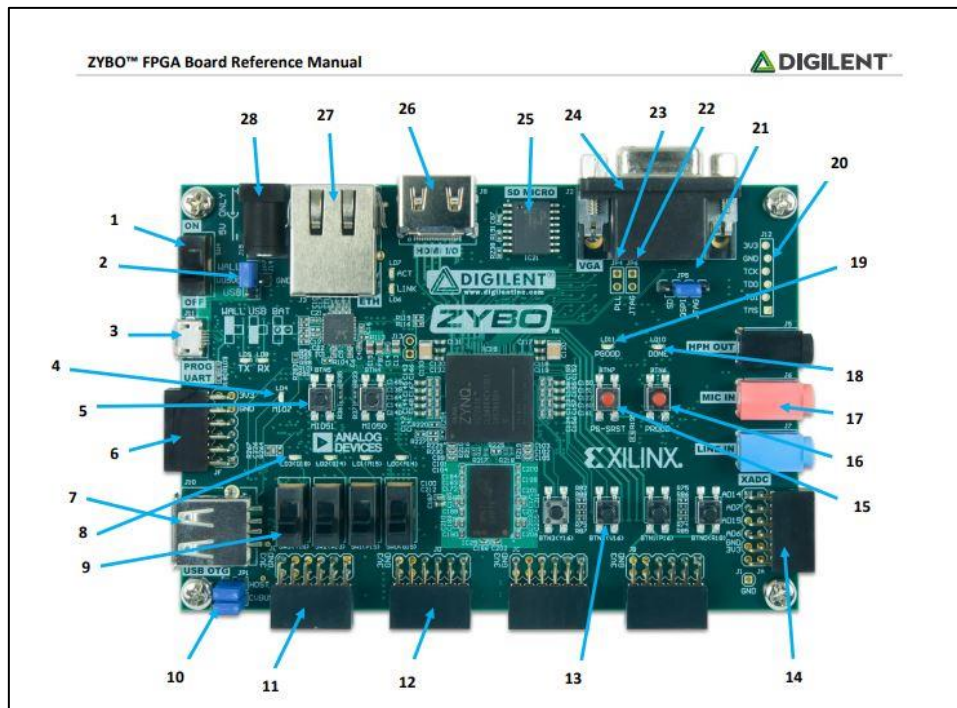


Figura 16. Placa de desarrollo Zybo.

Número	Descripción del componente	Número	Descripción del Componente
1	Interruptor de alimentación	15	Pulsador de Reset del Procesador
2	"Jumper" de selección de potencia	16	Pulsador de Reset de configuraciones lógicas
3	Puerto USB JTAG/UART compartido	17	Conectores Codec de audio
4	MIO LED	18	LED de configuración completada
5	Pulsadores MIO (2)	19	LED de energía de la Placa
6	MIO Pmod	20	Puerto JTAG para cable externo opcional
7	Conectores USB OTG	21	"Jumper" de modo programación
8	Leds (4)	22	"Jumper" de habilitación de modo JTAG independiente
9	Conmutadores lógicos (4)	23	"Jumper" de Bypass PLL
10	"Jumpers" de selección de USB OTG ANFITRIÓN/DISPOSITIVO	24	Conector VGA
11	Pmod Standard	25	Conector tarjeta microSD (parte trasera)
12	Pmods de gran velocidad (4)	26	Conector HDMI
13	Pulsadores lógicos (4)	27	Conector RJ45 Ethernet
14	Pmod XADC	28	Enchufe de alimentación

Tabla 1. Componentes de la placa de desarrollo Zybo.

Para los objetivos del trabajo es particularmente interesante que el dispositivo Zynq contenga un doble core del procesador ARM porque esto nos permite realizar el estudio de los algoritmos de cifrado implementados en software e implementados en hardware, y así comprobar si verdaderamente es posible acelerar mediante hardware los tiempos de ejecución.

- Procesador ARM Cortex-A9

El Cortex-A9 [17] es un procesador de alto rendimiento y bajo consumo que proporciona capacidad de memoria virtual completa. Este procesador implementa la arquitectura ARMv7-A y corre instrucciones ARM de 32 bits. El Cortex-A9 puede ser usado en modo uniprocador y en modo multiprocador. En este último modo se disponen de hasta cuatro procesadores Cortex-A9 en un clúster que comparte la caché de segundo nivel, la placa Zybo usa dos de ellos. Un esquema del multi-core se muestra en la Figura 17.

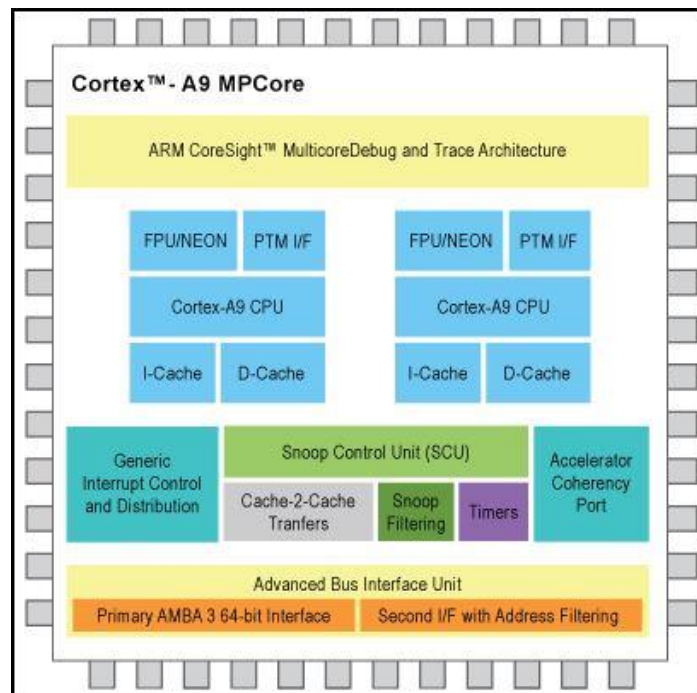


Figura 17. Esquema interno del Cortex-A9.

Como se observa en la Figura 17, el sistema multiprocador tiene, aparte de los dos procesadores, una unidad denominada Snoop Control Unit (SCU) que es responsable, entre otras funciones, de mantener la coherencia entre las cachés de datos L1. También tiene un controlador de interrupciones (IC) con soporte para interrupciones ARM, un reloj global, un timer y un "watchdog" dedicado para cada procesador, y diferentes puertos de comunicación basados en el estándar AXI4.

Los buses de comunicación que incorpora usan el "AXI4-Protocol" [18], y soportan diseños de alto rendimiento y sistemas de alta frecuencia. Proporcionan operaciones de alta frecuencia sin usar puentes complejos y reconoce los requerimientos de interfaz de un gran número de componentes. Otra de las principales características es que separan los canales de escritura y lectura, y por tanto pueden proporcionar un acceso directo a memoria (DMA) de bajo coste tal. Los distintos buses disponibles en los dispositivos zynq se describen a continuación:

A) Buses AXI Stream.

El protocolo AXI4-Stream [19] se utiliza como interfaz estándar para conectar componentes que desean intercambiar datos. La interfaz se puede utilizar para conectar un único maestro, que genera datos, a un solo esclavo, que recibe datos. El protocolo también se puede utilizar cuando se conectan numerosos maestros y/o esclavos utilizando canales independientes. El protocolo soporta múltiples flujos de datos usando el mismo conjunto de cables compartidos, lo que permite construir una interconexión genérica que puede realizar operaciones de aumento y reducción de tamaño y enrutamiento.

Una de las principales diferencias entre AXI4 y AXI4-Stream es que éste último no tiene definido un número máximo de ráfagas de envío o de tamaño de paquete. En adición, AXI4-Stream permite que el tamaño de datos sea cualquier número entero de bytes de datos.

B) Buses AXI4 y AXI4-Lite.

La interfaz AXI4 se usa para requerimientos de gran rendimiento mientras que AXI4-Lite [20] se usa para una comunicación simple, con registros en lugar de espacios de memoria (por ejemplo, hacia y desde los registros de control y de estado). AXI4 y AXI4-Lite contienen 5 canales:

- Canal de direcciones de lectura.
- Canal de direcciones de escritura.
- Canal de lectura de datos.
- Canal de escritura de datos.
- Canal de lectura de respuesta.

Los datos pueden moverse en ambas direcciones entre el maestro y el esclavo simultáneamente y el tamaño de dichas transferencias puede variar. AXI4 permite la transferencia en ráfagas de hasta 256 datos, mientras que AXI4-lite permite únicamente transferir 1 dato por transacción.

En los sistemas híbridos Software/Hardware desarrollados en este trabajo la comunicación entre el procesador ARM y la implementación hardware de cada uno de los cifradores se lleva a cabo, mediante buses de comunicación AXI4 y AXI-Stream. Para ello es necesario configurar la interfaz proporcionada en el API de ATHENA como un módulo-IP compatible con dichos buses (como se explican posteriormente). Dado que dicha interfaz es común para todos los cifradores presentados en la competición CAESAR, esta estrategia nos permite disponer de un sistema genérico de comunicación para cualquiera de los cifradores que cumplan con los requisitos de la competición.

4.1.2. Entorno de diseño Hardware.

En este apartado se va a describir el entorno de diseño utilizado para llevar a cabo las implementaciones de los cifradores. La herramienta usada ha sido Xilinx Vivado [21]. En un primer momento se ha trabajado con la versión 2015.4 para conocer la herramienta y sus diferencias fundamentales con Xilinx ISE. Una vez conocido el entorno de trabajo, se han llevado a cabo todas las pruebas e implementaciones con dicha versión y con la actualizada 2018.2.

Xilinx Vivado es un paquete de programas desarrollado por Xilinx para la síntesis y e implementación de sistemas digitales sobre FPGAs. Reemplaza a las herramientas de Xilinx ISE e incorpora funciones adicionales para el desarrollo de SoCs y la síntesis a alto nivel. Vivado representa una re-escritura desde 0 y un replanteamiento del flujo de diseño. En la Figura 18 se muestra la interfaz gráfica del entorno Vivado.

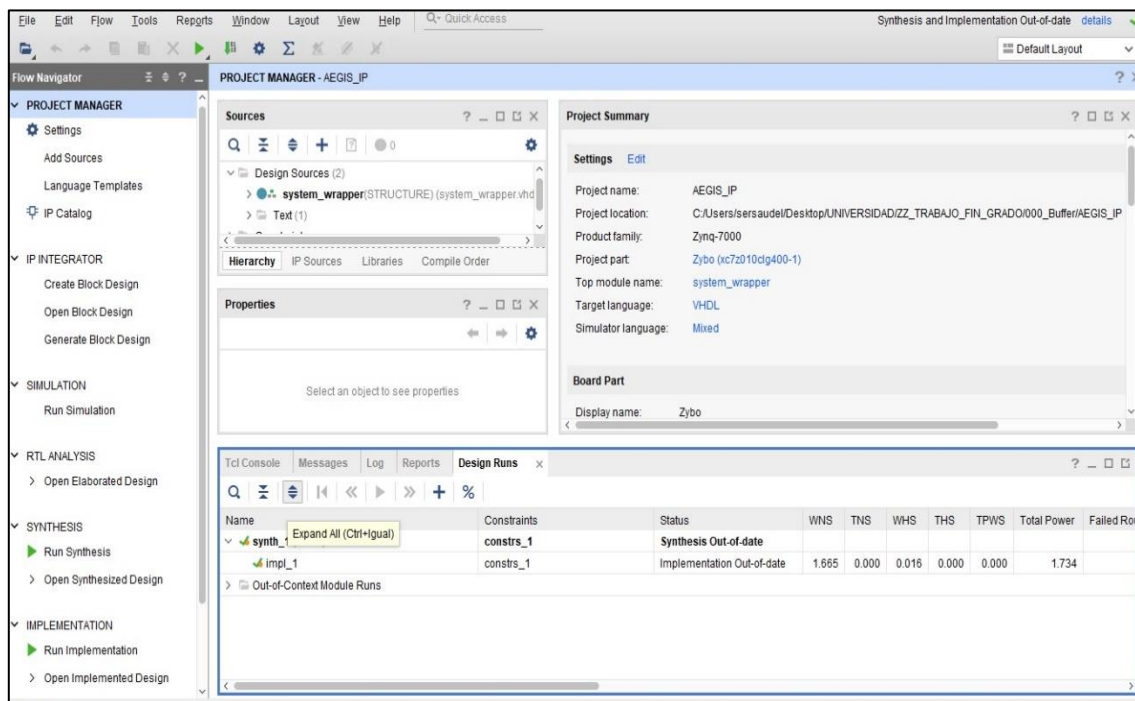


Figura 18. Interfaz gráfica del entorno de diseño Xilinx Vivado 2018.2

Desde 2018, Xilinx recomienda usar Vivado Design Suite para nuevos diseños con Ultrascale, Ultrascale+, Virtex-7, Kintex-7, Artix-7 y con el SoC que se va a utilizar en este estudio, zynq-7000. Los usuarios del entorno de trabajo pueden llevar a cabo sus diseños siguiendo flujos de diseño basados en HDL (VHDL o Verilog) y utilizar la funcionalidad que ofrece "IP integrator" para combinar sistema de procesamiento y módulos IP para construir un System on Chip (SoC). Esta herramienta de Vivado permite usar tanto los distintos IPs disponibles en la librería de Xilinx como los IPs creados por los propios usuarios. Esto se observa en la Figura 19 que muestra la herramienta *IP Catalog* del flujo de trabajo, en la que aparecen separados los IPs creados por el usuario y el repositorio que ofrece Vivado con IPs genéricos de uso común.

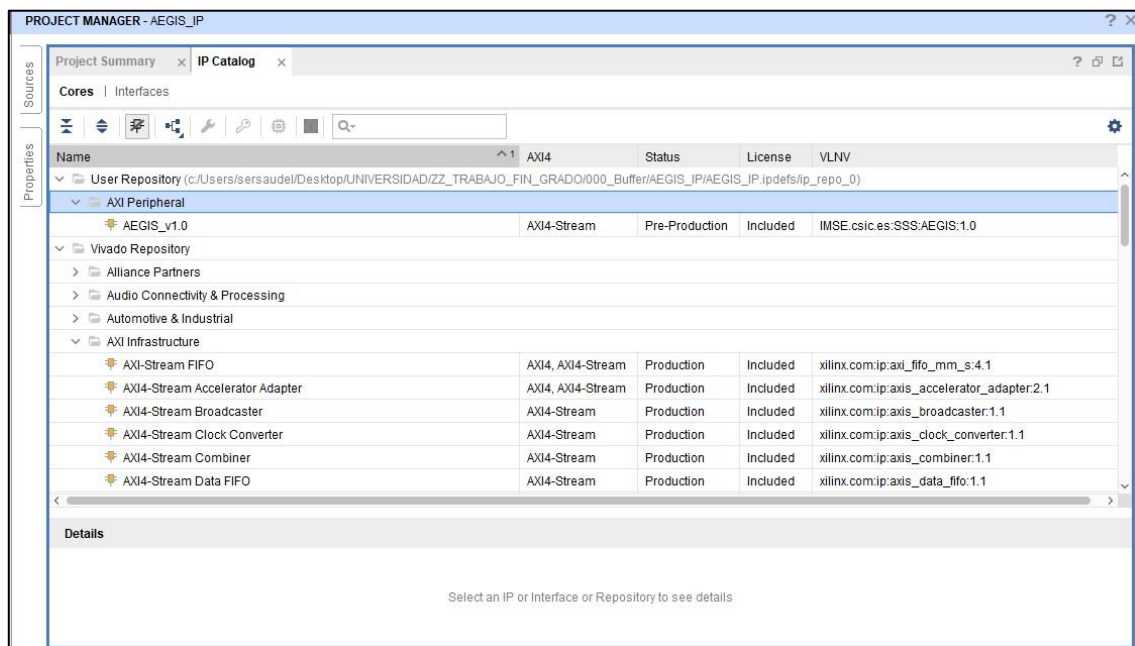


Figura 19. Repositorio IP de Vivado.

El navegador de flujo “Flow Navigator” que se muestra en la Figura 18, permite realizar muchas funciones dentro del entorno de Vivado. En la parte superior del navegador, se encuentran las opciones de configuración del proyecto que se está desarrollando, como la adición de fuentes, adición de un módulo IP del “*IP Catalog*”, etc... La siguiente herramienta del navegador es el *IP Integrator*, que permite realizar diseños de sistemas empotrados desde cero de forma gráfica utilizando IPs disponibles en el repositorio de Vivado.

El navegador de flujo incluye también la sección de simulación, que permite ejecutar la simulación de un en las distintas etapas del flujo de desarrollo: simulación funcional, simulación post-síntesis y simulación post-implementación, así como las secciones de síntesis e implementación del diseño con distintas configuraciones.

Para programar un dispositivo con un diseño realizado, se usa la sección “program and debug”, que permite programar el dispositivo tras haber implementado correctamente el diseño. Para llevar esto a cabo, es necesario generar el *bitstream* del diseño sintetizado e implementado y posteriormente abrir el “hardware manager”, el cual se encuentra en esta sección. El “hardware manager” mediante la opción de “Refresh Device” permite buscar el dispositivo que se desea programar (que debe estar conectado y alimentado) y con la opción “Program Device”, permite realizar una conexión local con el mismo y programarlo.

4.1.3. Entorno de diseño Software.

La herramienta incluida en Vivado para llevar a cabo la programación software es Xilinx Software Development Kit (XSDK) [22, 20]. Esta herramienta es un entorno de diseño integrado (IDE) basado en Eclipse que facilita la creación de aplicaciones integradas en los procesadores de Xilinx (ARM y MicroBlaze). SDK ofrece un diseño multiprocesador, depuración y análisis de rendimiento. Entre las funcionalidades se incluyen: soporte para los SoC mencionados anteriormente, IDE que conecta directamente con el entorno de desarrollo hardware de Vivado, soporte para diseño software y flujo de depuración así como la capacidad de hacer una co-depuración multiprocesador Hardware/Software, etc... En la Figura 20 se muestra la interfaz gráfica del entorno de desarrollo Xilinx SDK.

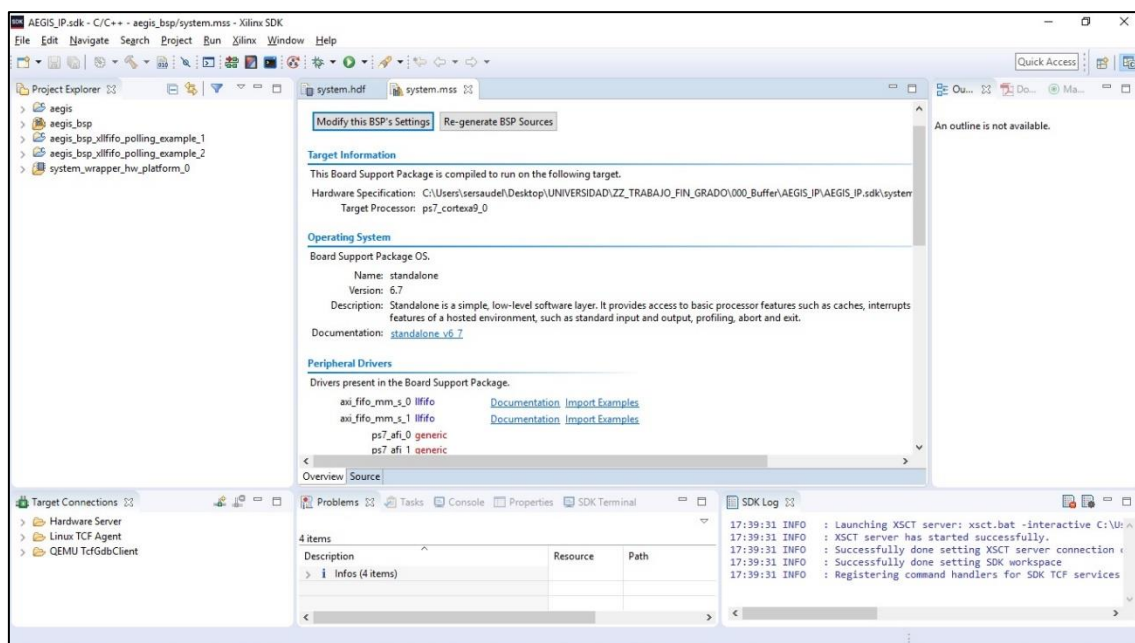


Figura 20. Entorno de desarrollo Xilinx SDK.

En este entorno se observan distintas partes como son el explorador del proyecto, donde se muestra toda la jerarquía del mismo, la zona de trabajo y la consola de comandos. Cuando se inicia SDK en un directorio concreto, únicamente se encuentra el módulo hardware que se ha exportado previamente desde Vivado. Para ejecutar una aplicación en el dispositivo, es necesario crear previamente una plataforma software para el módulo hardware importado, dicha plataforma es denominada *Board Support Package*. El proceso de creación de una nueva aplicación de usuario, permite importar fuentes de un programa o crearlas en el entorno así como elegir el lenguaje de programación (C++, por ejemplo) y el sistema operativo (Standalone, por ejemplo). Una vez compilada la aplicación y generado el ejecutable correspondiente, se puede programar el dispositivo con la plataforma hardware y ejecutar la aplicación de usuario sobre la misma.

4.2. Desarrollo del hardware del sistema empotrado

Los sistemas empotrados que se han desarrollado en este proyecto están compuesto básicamente por dos elementos; un módulo cifrador y un procesador ARM en el que se ejecutará el software de verificación del funcionamiento del cifrador, y que también sería el encargado de llevar a cabo cualquier otro tipo de tareas que hiciesen uso del cifrador como un periférico específico. En este apartado se describen las tareas llevadas a cabo para la creación de un módulo IP del cifrador que pueda ser utilizado como periférico del ARM. Todas estas etapas son comunes para cualquier cifrador presentado en la competición CAESAR, aunque para ilustrar todo el proceso se ha tomado el cifrador autenticado “AEGIS” como modelo.

4.2.1. Implementación de cifrador.

El primer paso para construir un módulo IP que implemente el cifrador AEGIS como periférico del sistema de procesado disponible en el dispositivo Zynq-7000, requiere tomar el cifrador presentado en la competición y llevar a cabo una serie de ajustes. Estos ajustes consisten en tomar el programa generador de Test Vectors del cifrador y configurarlo de tal forma que nos presente los datos *pdi*, *sdi* y *do* en un formato de 32 bits. Esto se debe a que el cifrador original tiene una anchura de datos públicos de entrada de 256 bits, mientras que el módulo IP se conectará al procesador a través de buses AXI-Stream con 32 bits de datos.

Para facilitar la verificación del diseño con vectores de test proporcionados por los autores, también es necesario configurar el programa generador de Test Vectors del cifrador de tal forma que presente los datos *pdi*, *sdi* y *do* en un formato de 32 bits. Al ejecutar AETVgen para este cifrador tras haber modificado la anchura se obtienen los ficheros *pdi.txt*, *sdi.txt* y *do.txt*. En la Figura 21 se muestra parte del contenido de *pdi.txt* en formato de 32 bits para el software de AEGIS

```
#### Authenticated Encryption
#### MsgID= 5, KeyID= 3 Ad Size = 0, Pt Size = 1
# Instruction: Opcode=Activate Key
INS = 70000000
# Instruction: Opcode=Authenticated Encryption
INS = 20000000
# Info :                               Npub, EOI=0 EOT=1, Last=0, Length=16 bytes
HDR = D2000010
DAT = B0B1B2B3B4B5B6B7B8B9BABBBBCBDBEBF
# Info :                               Associated Data, EOI=0 EOT=1, Last=0, Length=0 bytes
HDR = 12000000
# Info :                               Plaintext, EOI=1 EOT=1, Last=1, Length=1 bytes
HDR = 47000001
DAT = FF000000
```

Figura 21. Formato de instrucción modificada del fichero *pdi.txt*.

Una vez modificado el formato de las instrucciones, se crea un proyecto en Vivado con el nombre CIPHER-32 (Sustituyendo CIPHER por el nombre del cifrador en cada caso). En el caso de AEGIS, AEGIS-32. A continuación se procede a incluir cada una de las fuentes de diseño y simulación de este primer diseño. Todas las fuentes proporcionadas por ATHENA están disponibles para su consulta en el apartado de CAESAR: “*GMU source code of round 3*” en [13].

Las fuentes del diseño están incluidas en el archivo de distribución del propio cifrador. En el caso del cifrador AEGIS están incluidas en “AEGIS_GMU_v1.1.zip”.

Estas fuentes son:

```
AEAD.vhd,  
AEAD_Arch.vhd,  
AEAD_pkg.vhd,  
AEAD_Wrapper.vhd,  
AES_invmap.vhd,  
AES_map.vhd,  
AES_MixColumn.vhd,  
AES_MixColumns.vhd,  
AES_mul.vhd,  
AES_pkg.vhd,  
AES_Round.vhd,  
AES_sbox.vhd,  
AES_ShiftRows.vhd,  
AES_SubBytes.vhd,  
CipherCore.vhd,  
CipherCore_Control.vhd,  
CipherCore_Datapath.vhd,  
fwft_fifo.vhd,  
PostProcessor.vhd y  
PreProcessor.vhd.
```

Al incluir los ficheros anteriores como fuentes de diseño se selecciona como `top_module` AEAD.VHD.

Las fuentes de simulación del comportamiento del diseño también vienen proporcionadas en el archivo del cifrador.

Estas fuentes son:

```
AEAD_TB.vhd y  
Std_logic_1164_additions.vhd.
```

Además de incluir estos ficheros como fuentes de simulación, es necesario añadir los ficheros “pdi.txt”, “sdi.txt” y “do.txt” (adaptados a 32 bits según se ha comentado antes) que serán empleados por el testbench para leer los datos de entrada y comparar a la salida del sistema los datos esperados y los proporcionados por el cifrador.

La adaptación del tamaño de los buses que utiliza el cifrador para recibir y transmitir datos públicos y secretos se lleva a cabo modificando los parámetros `G_PWIDTH` y `G_SWIDTH` en las líneas 39 y 40 del fichero AEAD_TB.vhd como se muestra en la Figura 22.


```

39      G_PWIDTH      : integer := 32;
40      G_SWIDTH       : integer := 32;

```

Figura 22. Modificación de la anchura de los buses de datos públicos.

Una vez añadidas todas las fuentes, el proyecto quedará en el caso de AEGIS como se muestra en la Figura 23.

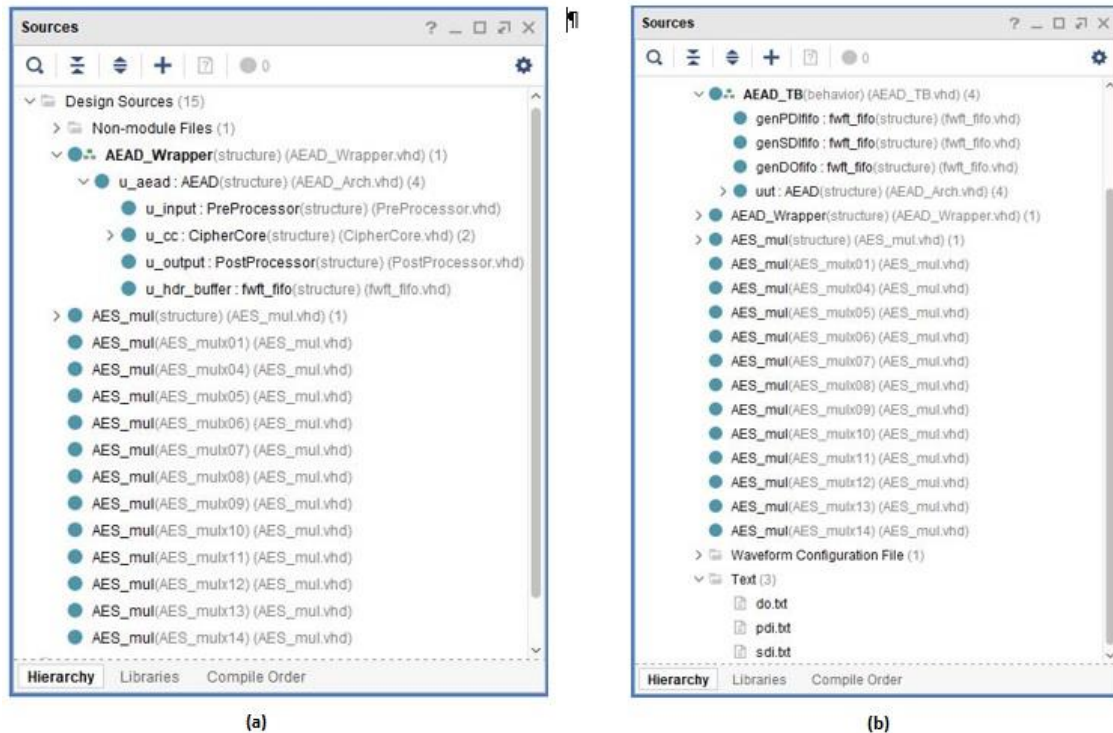


Figura 23. Fuentes de AEGIS-32, a) Fuentes de diseño, b) Fuentes de simulación.

Tras completar la descripción del cifrador, el siguiente paso consiste en verificar el funcionamiento del diseño, para lo que se llevará a cabo una simulación funcional del mismo, como se muestra en la Figura 24. Se puede observar que el diseño en el caso de AEGIS funciona correctamente en la simulación, pues las FIFOs de entrada de datos mencionadas en el apartado 4.4.1 introducen los datos en el cifrador autenticado por medio de las entradas *fpdi_din* y *fsdi_din*. Estos datos son procesados y se observa más abajo que tras un tiempo se activan las señales *do_ready* y *do_valid* indicando que se están obteniendo datos de salida por el puerto *do*. Dichos datos son transferidos a través de la FIFO de salida *fdo_dout* y de acuerdo con el protocolo de comunicación (*handshake*) definido por las señales *fdo_dout_valid* y *fdo_dout_ready*. Utilizando las facilidades de Vivado, la representación de las formas de onda se podrá ampliar para ver cada uno de los datos introducidos y corroborar que efectivamente coinciden con los datos que muestran los ficheros *pdi.txt*, *sdi.txt* y *do.txt*.

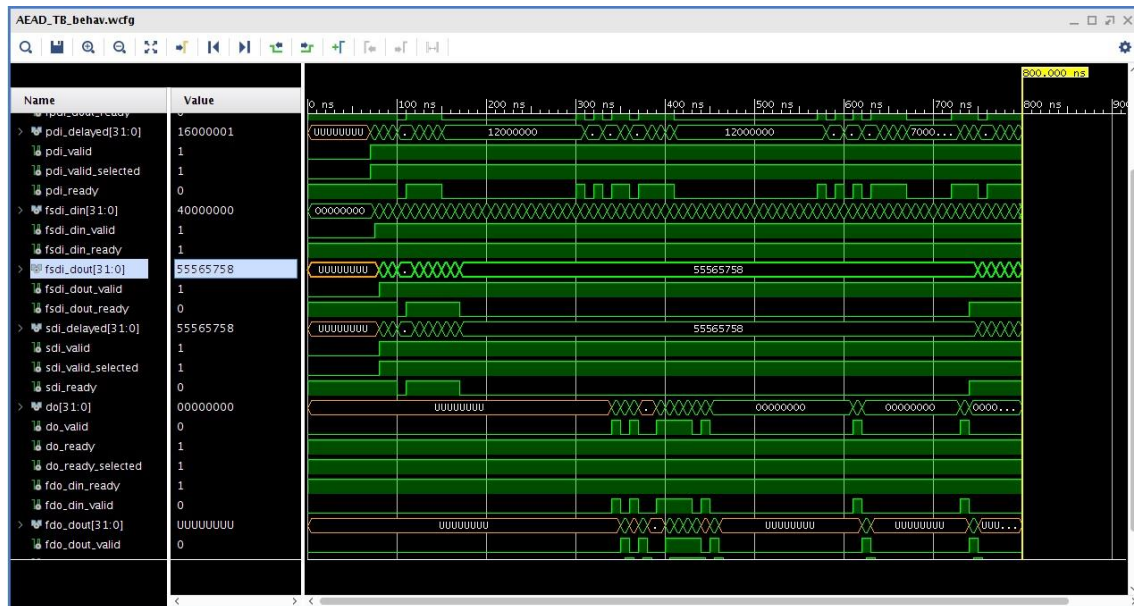


Figura 24. Simulación de AEGIS-32.

Una vez verificada la simulación funcional, interesa conocer cuánto ocupa (cuántos recursos utiliza) el diseño del cifrador en la FPGA seleccionada para este proyecto. Para ello se han llevado a cabo una serie de procesos de síntesis e implementación con distintas estrategias, para observar cómo cambia el uso de recursos de la placa según la estrategia seleccionada.

Los procesos de síntesis e implementación llevados a cabo han sido:

- Síntesis por defecto e implementación por defecto.
- Síntesis “Flow Area Optimized High” e implementación “Area Explore”.
- Síntesis “Flow Performace Optimized High” e implementación “Performacnce Refine Placement”.
- Síntesis “Flow Area Optimized High” e implementación “Flow Run Physical Optimized”.

Debido a que todos los procesos obtenían resultados muy similares de consumo de recursos pero ninguno superaba los valores de síntesis e implementación por defecto, esta es la estrategia elegida para futuras implementaciones. La

Tabla 2 muestra una comparación de consumo de recursos de la FPGA con cada uno de los procesos de síntesis e implementación mencionados.

	Default/Default	Flow área optimized high/Area explore	Flow perf opt high/ Perf Refine Placement	Flow área opt high/Flow Run physical optimized
Slice LUTs	7250	7540	7489	7350
Slice Registers	2118	2126	2132	2126
F7 Muxes	2048	2048	2048	2048
F8 Muxes	1024	1024	1024	1024
Slice	1951	2035	2026	1988
LUT as Logic	7228	7510	7467	7328
LUT as Memory	22	22	22	22
LUT FF Pairs	1416	1507	1532	1414
Bonded IOB	5	5	5	5

Tabla 2 Comparación de todas las estrategias implementadas para la creación de cipher-32.

4.2.2. Generación del módulo IP del cifrador.

Una vez realizada la verificación funcional del cifrador a través de la simulación del mismo, procedemos a convertirlo en un módulo IP conectable al procesador ARM. Este módulo IP, estará compuesto por el cifrador, CIPHER-32, y tres FIFOs que serán las encargadas de transmitir la información de entrada al *PreProcessor* y la de salida del *PostProcessor*. Cabe destacar que estas FIFOs (ahora internas al IP) son similares a las incluidas en el fichero de testbench utilizado para verificar el funcionamiento del diseño AEGIS-32, como se muestra en el esquema del módulo IP en la Figura 25.

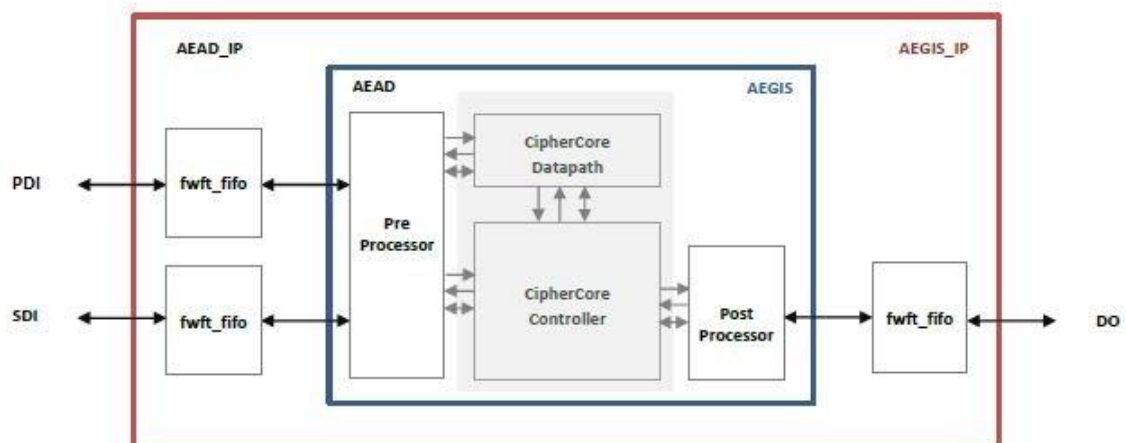


Figura 25. Esquema del módulo AEAD_IP.

Por tanto, para diseñar este módulo IP se ha creado un nuevo proyecto al cual se han añadido la nueva fuente AEAD_IP.vhd como fuente de diseño además de las usadas para crear el cifrador, CIPHER-32.

Para verificar la funcionalidad de este módulo mediante simulación se ha desarrollado un nuevo fichero de test-bench, AEAD_IP_TB.vhd similar al usado en el apartado anterior, pero que en esta ocasión instancia a AEAD_IP.vhd. Como en el diseño anterior, se incluirá este fichero como fuente para la simulación así como Std_logic_1164_additions.vhd y los ficheros de texto pdi.txt, sdi.txt y do.txt generados previamente.

La Figura 26 muestra la jerarquía del proyecto una vez incluidas todas las fuentes de diseño y de simulación, para el caso de AEGIS.

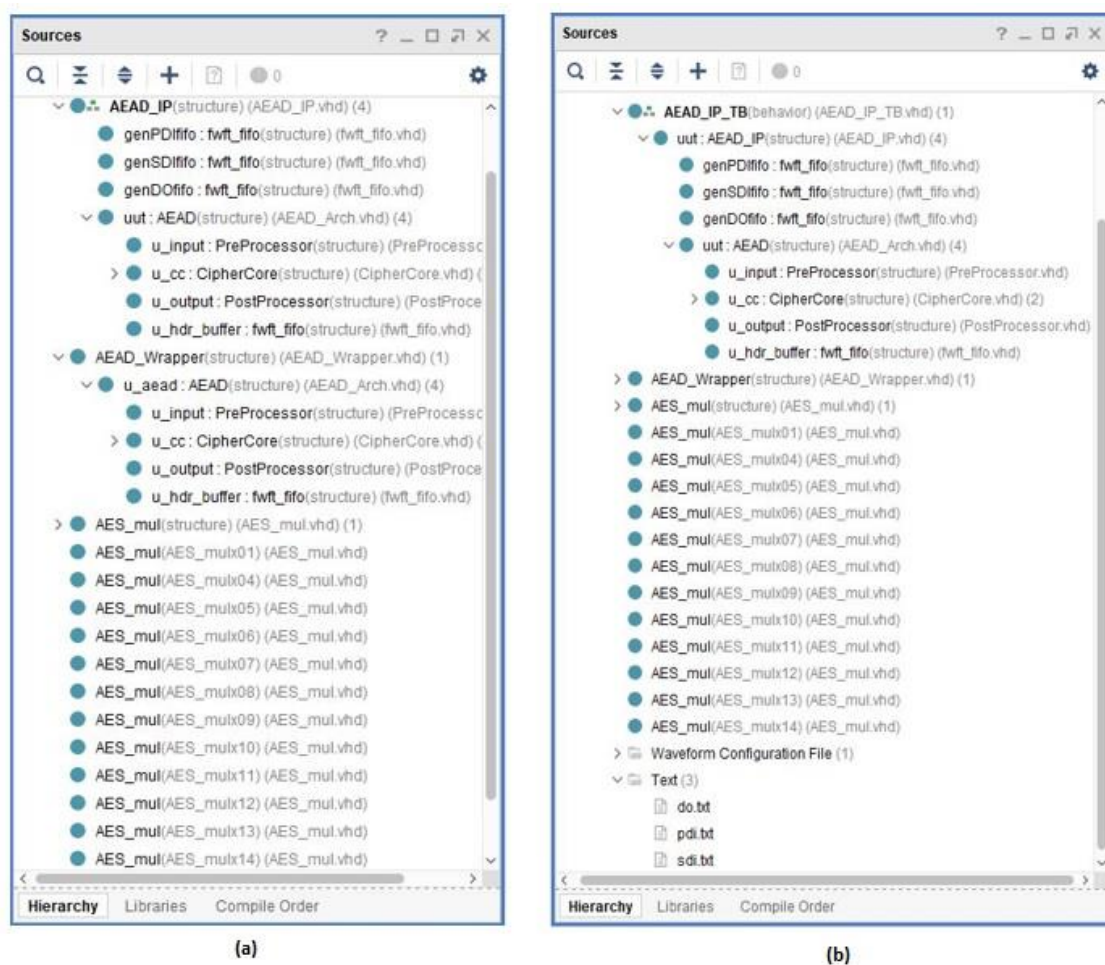


Figura 26. Fuentes del módulo AEGIS-IP, a) Fuentes de diseño, b) Fuentes de simulación.

Tras la adición de todas las fuentes se lleva a cabo la simulación del comportamiento marcando como LA ENTIDAD “AEAD_IP_TB” como nivel principal del diseño (*set as top*). El resultado obtenido se muestra en la Figura 28, donde se puede comprobar que las FIFOs de entrada y la de salida están intercambiando información con el cifrador. Las señales de protocolo “*fdout_valid*” y “*fdout_ready*” se encargan de

informar, respectivamente, de que el emisor dispone de un dato que puede ser enviado y de que el receptor está listo para aceptarlo.

El siguiente paso para convertir el cifrador en un módulo IP es incluir una serie de interfaces que faciliten su conexión a algunos de los buses estándares soportados por el procesador ARM. En este caso, puesto que las entradas y salidas del cifrador se conectan a través de las FIFOs, la elección natural será el bus AXI4-Stream

Para completar el desarrollo del periférico, se deben seguir una serie de pasos, el primero de ellos es utilizar una de las facilidades que ofrece Vivado para la creación y empaquetamiento de un IP, tal como se muestra en la Figura 27.

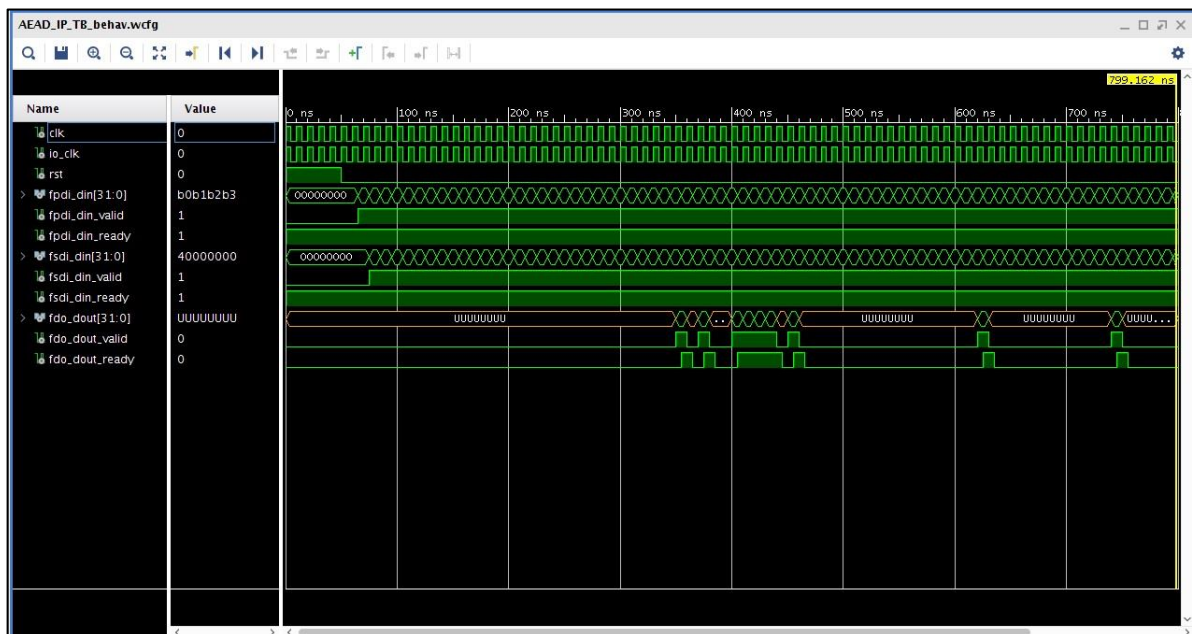


Figura 28. Simulación del comportamiento de AEGIS-IP.

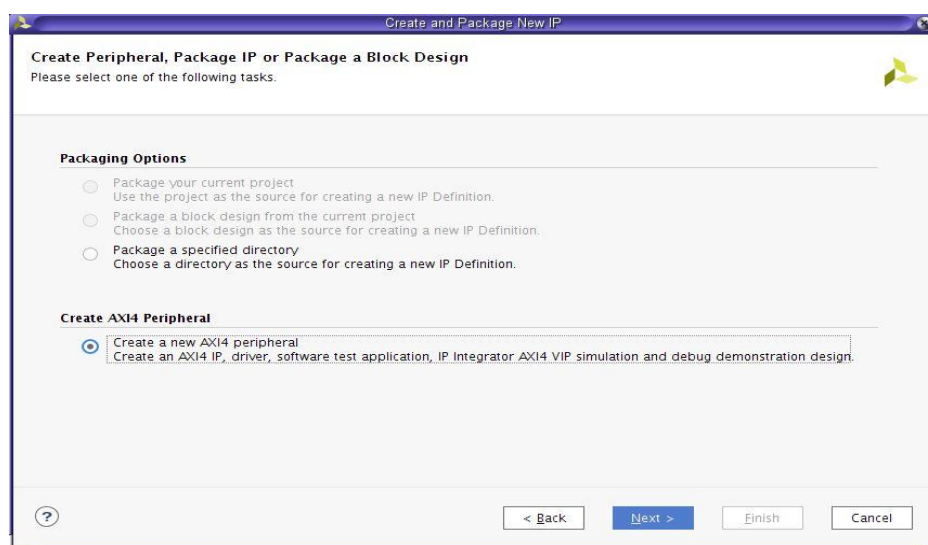


Figura 27. Asistente de creación de un nuevo paquete IP.

Tras proporcionarle un nombre al periférico, en el caso de AEGIS el periférico se llamará “aegis” se añaden y configuran las interfaces AXI4 adecuadas. Las entradas del periférico cifrador de tipo AXI-Stream, y configuradas como esclavas, mientras que la salida, de tipo AXI4-Stream y actuando como maestra, aparece en la Figura 29. Como se ha comentado anteriormente, la anchura de los buses AXI4-Stream es de 32 bits.

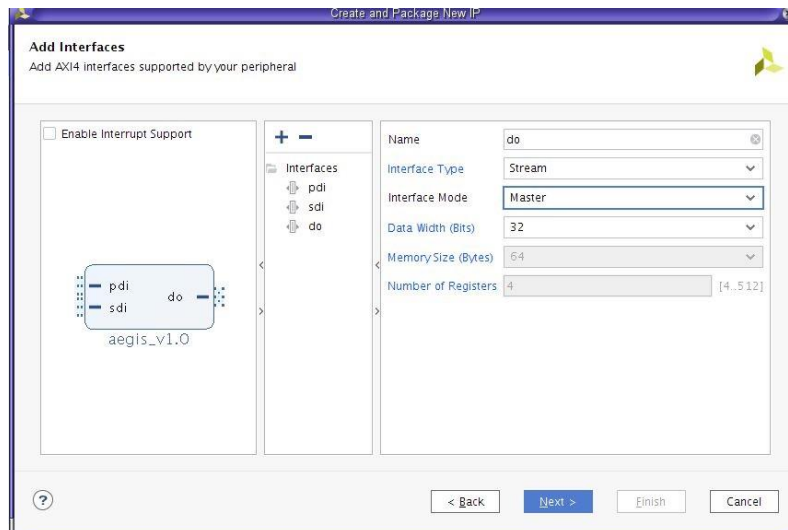


Figura 29. Salida del periférico cifrador.

Tras completar los pasos para la creación de este nuevo periférico, pero antes de utilizarlo en el sistema empotrado, es conveniente asegurar por medio de una simulación que funciona correctamente. Para ello se ha generado un nuevo proyecto CIPHER-BD en el que el diseño se define mediante un diagrama de bloque usando la herramienta IP integrator. En este diseño se instancia el módulo creado previamente, en el caso de AEGIS: aegis_v1_0_0. Además, se añade un bloque “Utility vector logic” configurado como inversor y se conecta a las señales de reset del bloque, ya que estas señales son activas en alta, mientras que en el primer testbench que se va a utilizar para la simulación la señal de reset es activa en baja.

Los puertos correspondientes a las entradas de reloj (*clk*) y reset (*rst*) se crean mediante la opción “create port” que proporciona una ventana para introducir los parámetros necesarios, como se muestra a continuación en la Figura 30.

Para crear los puertos de las entradas “pdi” y “sdi” se seleccionan estas señales en el cifrador “aegis” y se hacen externas con la opción “Make interface”. En el caso de la salida el procedimiento es el mismo, sin embargo el nombre de la interfaz se modifica a “dto” para que no se llame “do” ya que es una palabra reservada por Vivado y generaría un conflicto con la herramienta.

Quedando, después de completar el diagrama de bloques, como resultado final el diseño mostrado en la Figura 31.

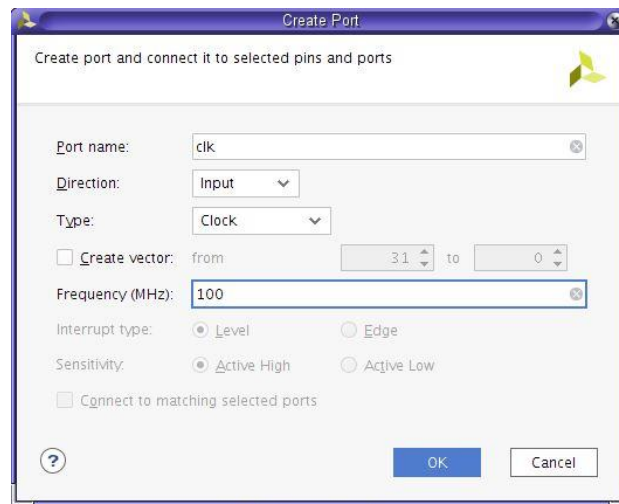


Figura 30. Configuración de la señal de reloj con los parámetros adecuados.

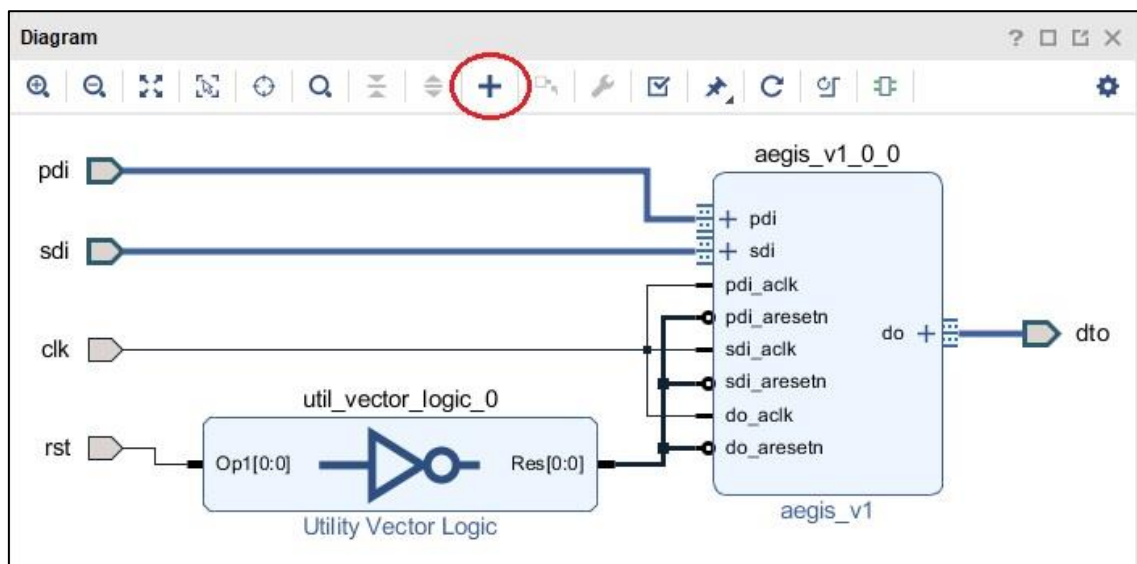


Figura 31. Diagrama de bloque cipher usando "aegis".

A continuación, se procede a validar el diseño y a crear un "envoltorio" ("*HDL WRAPPER*") para el mismo. El fichero VHL que define dicho envoltorio, se convierte automáticamente en una fuente de diseño. Este proceso se muestra en la Figura 32. Tras crear el HDL Wrapper, denominado "cipher_wrapper" (en este caso "aegis_wrapper"), se procede a incluir las fuentes necesarias para la simulación. Estas fuentes se muestran en la Figura 33. Donde AEAD_BD_TB.vhd es un test-bench similar a AEGIS_IP_TB.vhd, pero que en esta ocasión instancia al módulo "cipher_wrapper" (en este caso "aegis_wrapper").

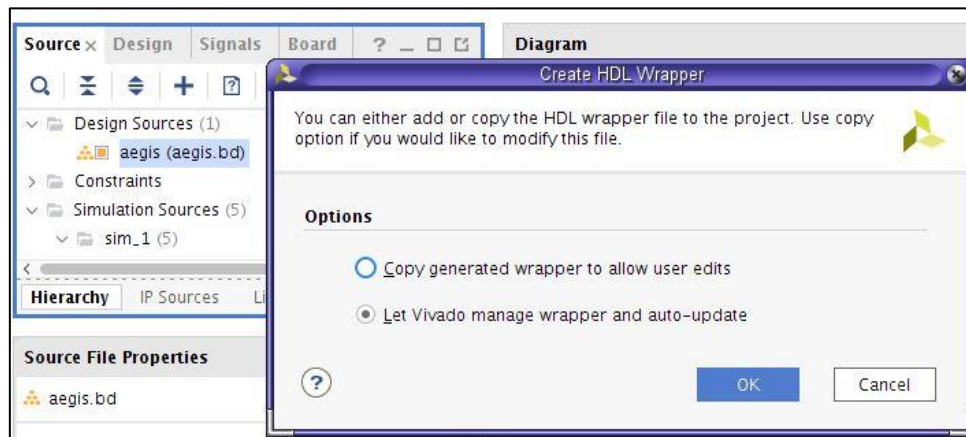


Figura 32. Creación del envoltorio del diseño.

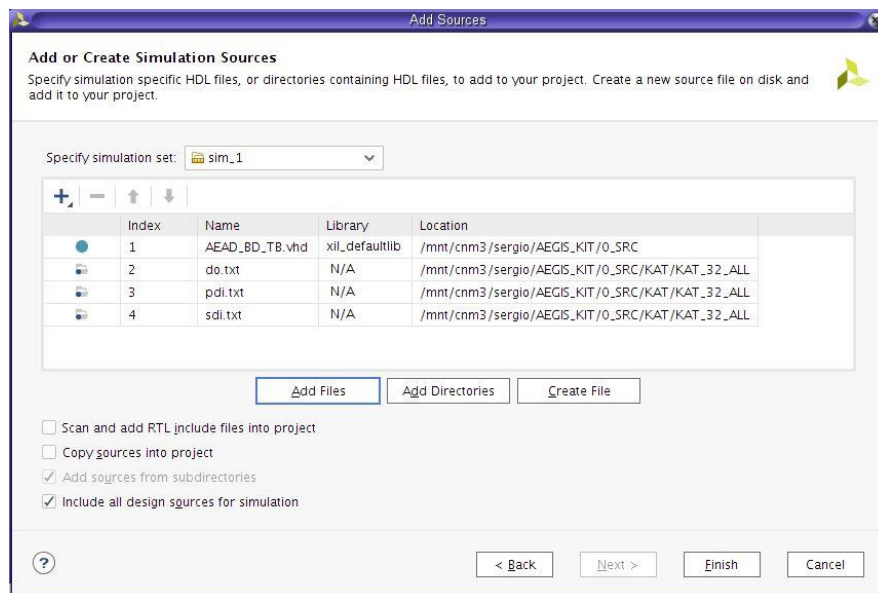


Figura 33. Fuentes de simulación de AEGIS-BD.

Tras llevar a cabo los pasos descritos, la jerarquía del proyecto en el caso de AEGIS queda como se muestra en la Figura 34. El resultado obtenido tras llevar a cabo la simulación funcional de AEGIS_BD se muestra la Figura 35, en la que puede apreciarse un comportamiento similar a los obtenidos al simular el cifrador antes de su transformación en módulo IP.

cifrador, así como otros IPs disponibles en el catálogo de IPs de Vivado necesarios para realizar la conexión de ambos.

Como se ha comentado previamente, las interfaces del módulo IP que implementa el cifrador son del tipo AXI4-Stream. El procesador puede intercambiar información a través de estas interfaces usando mecanismos diferentes. El primero es enviando y recibiendo los datos directamente, a través del bus AXI4 o AXI4-Lite, utilizando para ello un módulo IP denominado “AXI-Stream FIFO”. Cuando se usa la otra alternativa el procesador controla la operación de un IP denominado “AXI DMA” que permite el intercambio directo de información entre el cifrador y la memoria del sistema. Las dos opciones serán detalladas en los apartados siguientes.

A) Conexión con buses AXI4-Lite y AXI4.

Estos dos tipos de buses comparten muchos elementos. Para seguir un orden creciente de complejidad se describirá en primer lugar el uso de AXI4-Lite y posteriormente se hará otro diseño con AXI4.

Tras crear un nuevo diseño con *IP Integrator* e incluir el sistema de procesado, el siguiente paso del diseño del sistema empotrado consiste en importar el módulo IP del cifrador. Para ello se abre el asistente de creación de un nuevo IP y se marca la opción de “package a specified directory”. Se toma la ruta del repositorio del módulo IP a importar y seguimos las opciones del asistente hasta finalizar. Hecho esto se abrirá una nueva ventana de edición de IP donde se puede empaquetar el módulo IP importado para ser usado en el proyecto actual.

Una vez hecho esto, se procede a incluir dos bloques “AXI-Stream FIFO”, el primero de ellos se encargará de la transmisión de datos secretos “sdi” al módulo IP del cifrador, (aegis_v1 en la Figura 39), mientras que el otro se encargará de dos tareas: la transmisión de los datos públicos “pdi” al módulo IP del cifrador y, a su vez, la recepción de los datos de salida “do”.

Los módulos “AXI-Stream FIFO” pueden proporcionar interfaces de transmisión y de recepción de datos. En el bloque de la Figura 39 encargado de enviar y recibir los datos públicos se distinguen ambas interfaces con los nombres “AXI_STR_TXD” para la recepción y “AXI_STR_RXD” para la transmisión. El bloque que envía los datos secretos al cifrador solo usa la primera de estas interfaces. Adicionalmente, el “AXI-Stream FIFO” puede incorporar un bus AXI-Stream de control, “AXI STR TXC”, que será deshabilitado en este caso.

La interfaz de configuración del “AXI-Stream FIFO” permite también seleccionar qué tipo de bus será utilizado para conectarlo al procesador. En este primer diseño los dos bloques se configuran con buses AXI4-Lite, como se muestra en la Figura 36 y en la Figura 37.

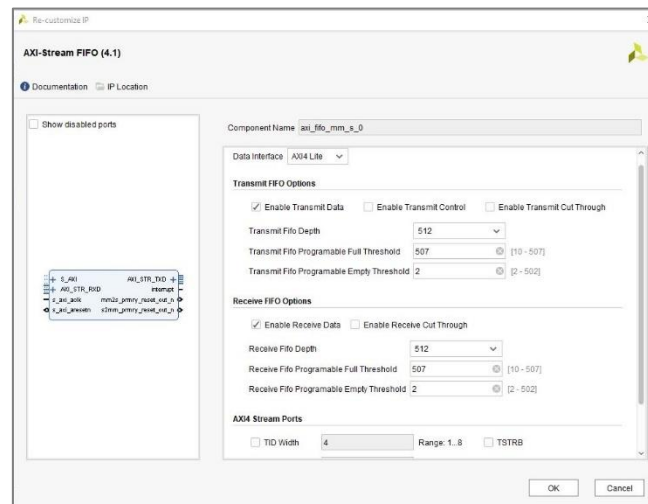


Figura 36. Configuración del módulo “AXI-Stream FIFO” para conectarlo al procesador a través del bus AXI4-Lite.

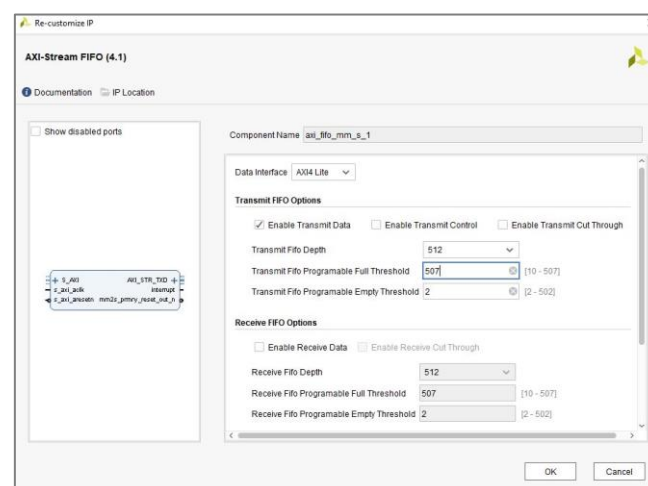


Figura 37. Configuración del módulo “AXI-Stream FIFO” para conectarlo al procesador a través del bus AXI4-Lite.

IP Integrator facilita la conexión automática de los componentes del diagrama de bloques. Al utilizar esta funcionalidad se incorporan al diseño un bloque “AXI Interconnect”, encargado de conectar los buses AXI4 esclavos de los “AXI-STREAM FIFO” con el bus AXI4 maestro del procesador. También se añade al sistema un bloque “Processor System Reset” encargado de generar distintas señales de reset a partir del reloj proporcionado por el procesador.

Con todos los bloques importados, se añade una entrada extra al procesador, la de habilitación de interrupciones, que será conectada directamente al bloque que transmite los datos públicos y recibe los datos de salida. Este proceso se realiza mediante la interfaz de configuración del procesador que se muestra en la Figura 38.

Finalmente se realiza la conexión de todos los componentes de acuerdo con el esquema que se muestra la Figura 39 y se lleva a cabo la validación del diseño.

Una vez validado el diseño, se procede a generar el “HDL Wrapper” tal como se había hecho en diseños anteriores y se lleva a cabo la síntesis e implementación del sistema. Por último, el diseño será exportado al entorno SDK descrito anteriormente en el que será usado como plataforma hardware para el desarrollo de aplicaciones.

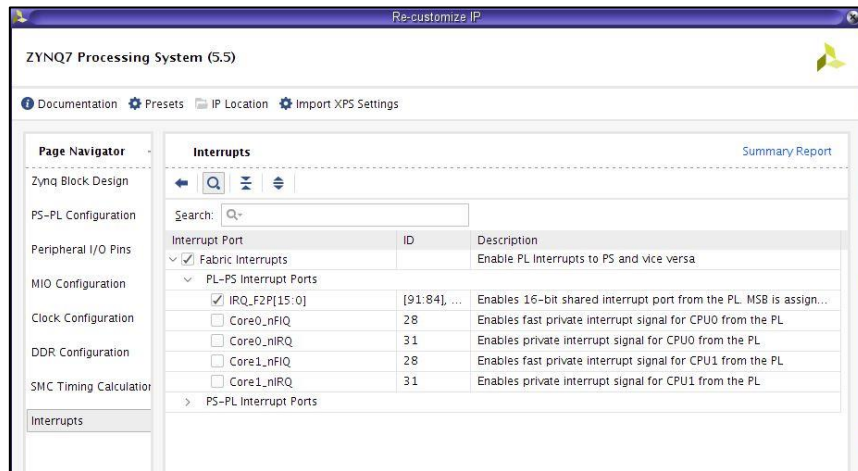


Figura 38. Interfaz de configuración del procesador ARM.

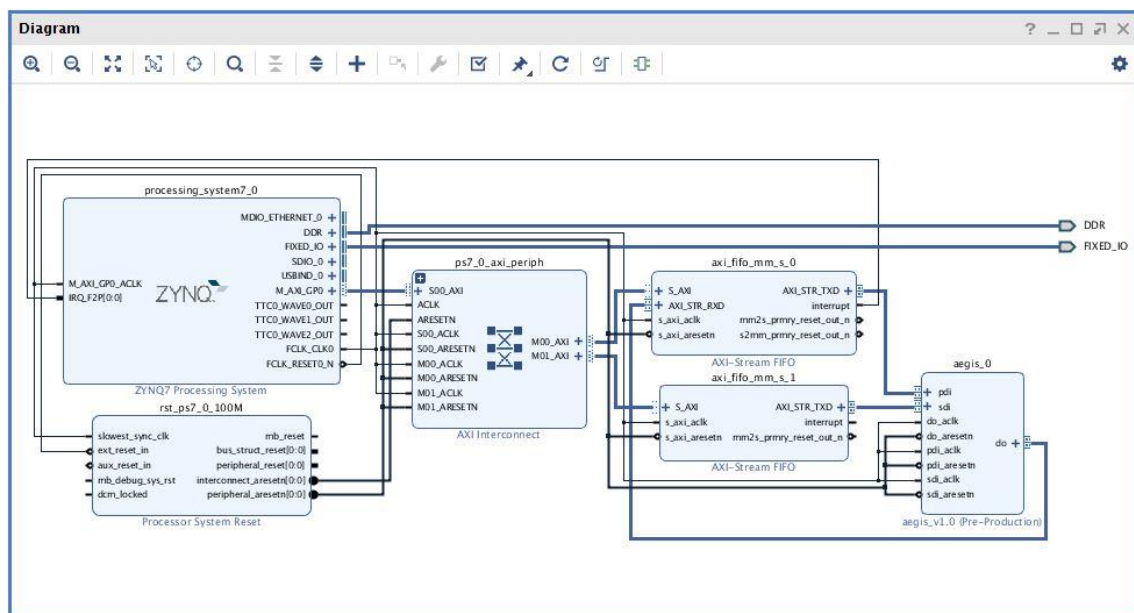


Figura 39. Conexión de los bloques del sistema empotrado usando AXI4-Lite.

El segundo diseño que se ha llevado a cabo utiliza la versión completa de AXI4 como bus de comunicación entre el procesador y los módulos “AXI-Stream FIFO”. Para ello, además de configurar adecuadamente estos bloques es necesario hacer algunos cambios que se muestran en la Figura 40.

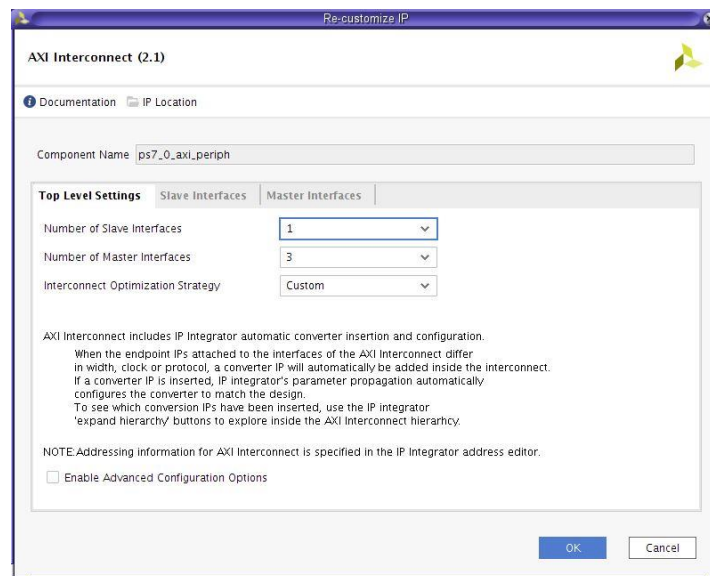


Figura 40. Configuración "AXI Interconnect".

Tras la configuración de ambos componentes, se lleva a cabo la conexión completa del diseño mostrado en la Figura 41 y se procede a su validación:

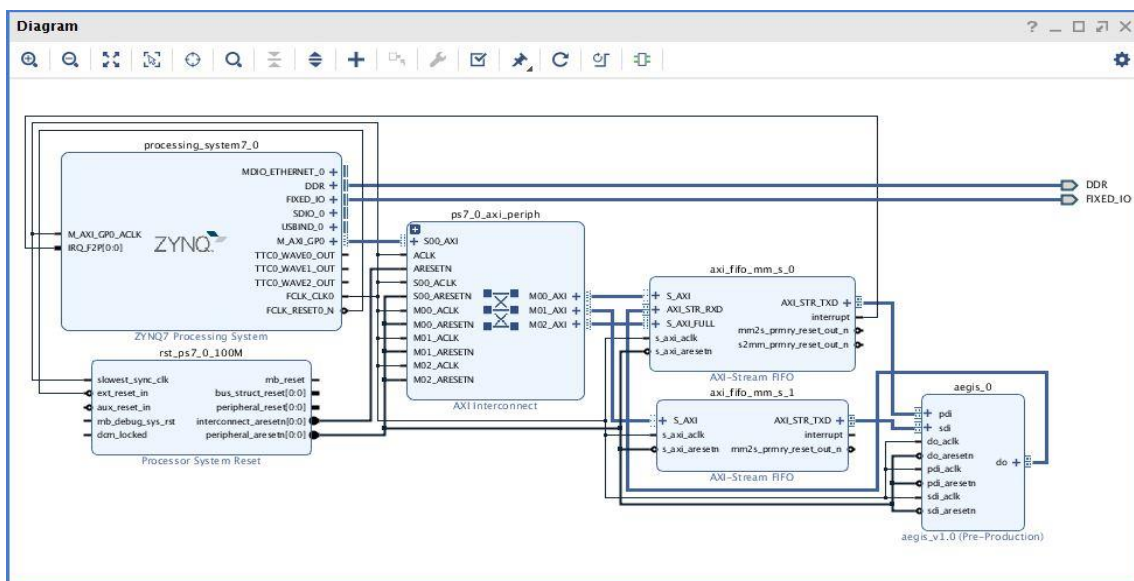


Figura 41. Conexionado de los bloques del sistema empotrado.

Posteriormente, se crea el "HDL Wrapper" y se lleva a cabo la síntesis e implementación.

Tras la generación del "bitstream" se procede a exportar el diseño Hardware como en el caso anterior para ser usado con el entorno SDK.

B) Conexión mediante DMA.

El bloque AXI "Direct Memory Access" (AXI DMA) [23] proporciona un acceso directo de gran ancho de banda entre la memoria y los periféricos de destino a través de buses AXI4-Stream. Sus capacidades opcionales de recopilación de dispersión también descargan las tareas de movimiento de datos a los procesadores. Los registros de inicialización, estado y gestión se acceden a través de una interfaz esclava AXI4-Lite.

Este componente se ha usado en el último diseño del sistema empujado llevado a cabo, en el que se ha usado para proporcionar una transmisión directa a memoria de los datos públicos intercambiados con el cifrador. A continuación, se muestra la configuración del componente en la Figura 42.

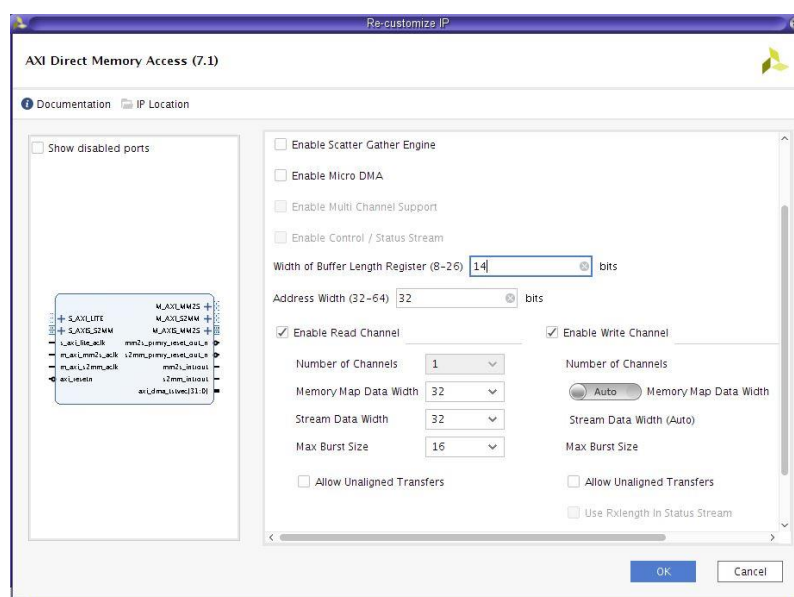


Figura 42. Configuración del AXI DMA.

Además de este IP, también es necesario añadir en el diseño otro componente, denominado "AXI SMART CONNECT", que se encarga de realizar una conexión maestro esclavo entre el AXI DMA y los buses de acceso al procesador y la memoria. La configuración de este componente se muestra en la Figura 43.

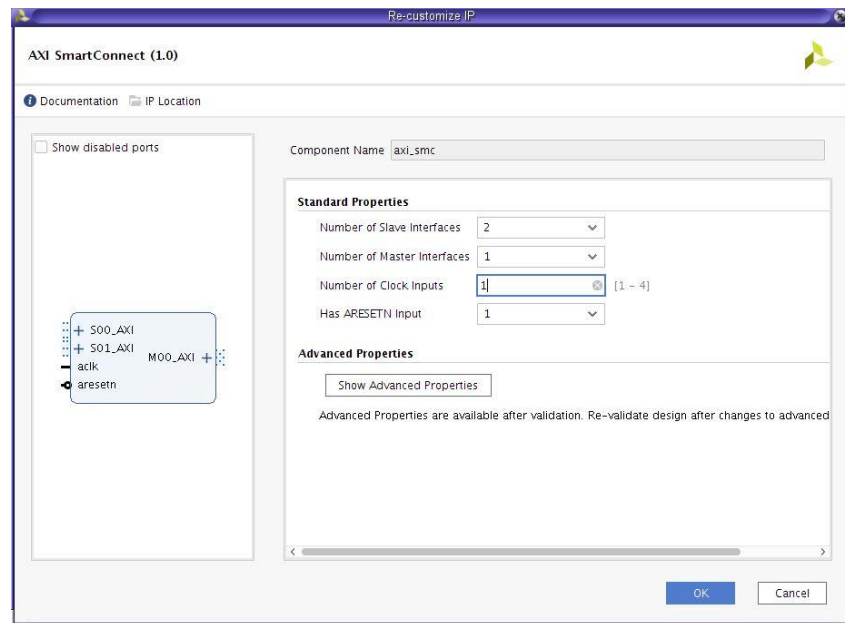


Figura 43. Configuración del bus AXI SMART-CONNECT.

Por último, para conectar al procesador las dos señales de interrupción generadas por el bloque AXI DMA, se añade otro componente denominado “Concat”. Su configuración se muestra en la Figura 44.

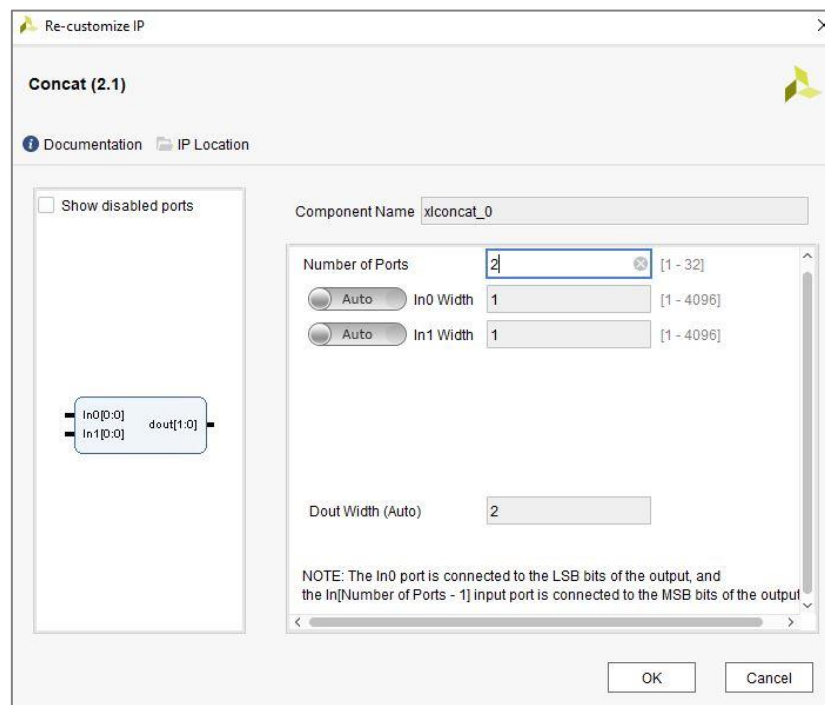


Figura 44. Configuración del componente Concat.

Una vez añadidos todos los componentes necesarios, se conectan los mismos y se valida el diagrama de bloques del diseño tal como muestra la Figura 45.

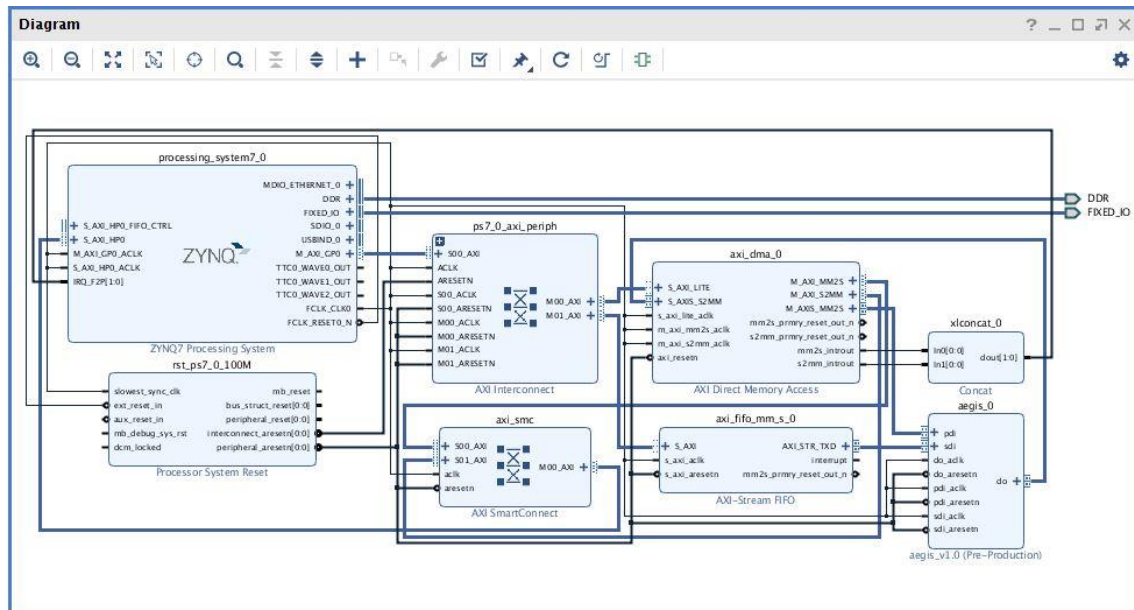


Figura 45. Diagrama de bloques del sistema empotrado.

Tras generar el “HDL Wrapper” la jerarquía completa del proyecto usando AEGIS como cifrador se muestra en la Figura 46:

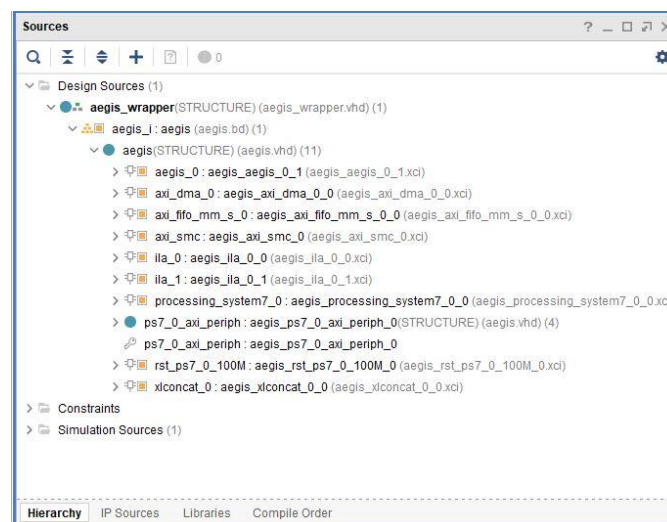


Figura 46. Jerarquía del diseño ARM con acceso directo a memoria.

Se procede a exportar el diseño Hardware para ser usado con el programa SDK como en las versiones anteriores del diseño.

4.3. Desarrollo del software del sistema empotrado

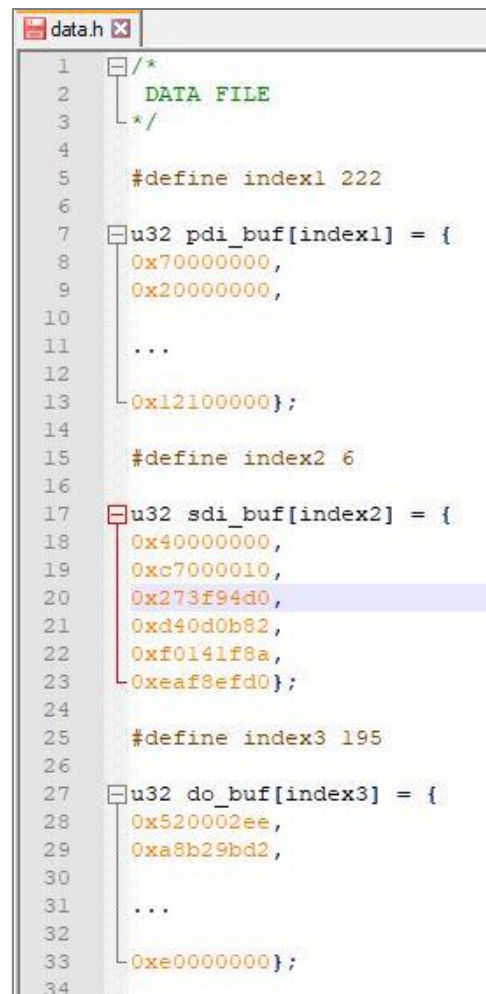
En este apartado, se describen los diferentes elementos software utilizados para verificar el funcionamiento del cifrador (en este caso, AEGIS) en cada uno de los sistemas empotrados que se han desarrollado en el apartado anterior. Para comprobar la funcionalidad de los diseños y comparar sus prestaciones se van a seguir tres estrategias distintas: la primera consiste en la verificación software del cifrador cuando el código C que define su comportamiento es compilado y ejecutado en el procesador ARM del sistema empotrado. Las otras dos estrategias están dirigidas a verificar el funcionamiento de la implementación hardware del cifrador por medio de *polling* [24], la segunda, y mediante interrupciones [25] del procesador, la tercera.

4.3.1. Adaptación del software de ATHENa a las limitaciones del sistema empotrado.

Para comprobar la funcionalidad de las implementaciones software y hardware de los diferentes cifradores se han empleado los vectores de test genéricos aportados por sus autores, así como otros más específicos generados mediante el programa de generación de vectores de test proporcionado por el proyecto ATHENa. No obstante, ha sido necesario realizar algunas modificaciones y desarrollar nuevos componentes software para aprovechar estas utilidades y llevar a cabo la verificación y comparación de los sistemas empotrados en sus distintas variantes.

La primera de ellas consistió en volver a compilar y reconfigurar la herramienta AETVGen de acuerdo con la anchura de los buses utilizados en la implementación hardware de los módulos IP. Una vez hecho esto, fue posible regenerar los vectores de test originales y obtener otros nuevos compatibles con los IPs de los distintos cifradores. Otra adaptación necesaria fue el desarrollo de un programa para facilitar la generación de fichero de cabecera para albergar los datos públicos, los datos secretos y los datos de salida, estos últimos utilizados para comprobar que coinciden con el resultado que proporciona el cifrador. Esta adaptación fue necesaria debido a que el procesador empotrado no tiene capacidad para manejar ficheros, por lo que no puede abrir los ficheros “pdi.txt”, “sdi.txt” y “do.txt” generados como salida para AETVGen para verificar las implementaciones hardware de los cifradores. El contenido de un fichero de cabecera se muestra en la Figura 47.

Una última cuestión relacionada con el software utilizado en este caso para comparar las prestaciones de las distintas alternativas de implementación, consistió en la inclusión de una rutina de medida de tiempos, para determinar el tiempo exacto que se tarda en cifrar un mensaje, tanto en las aplicaciones encaminadas a verificar la implementación del cifrador mediante software empotrado, como en las que se utiliza el módulo IP con las distintas variantes de conexión del procesador. El código correspondiente a dicha rutina de medida de tiempos se muestra en la Figura 48.

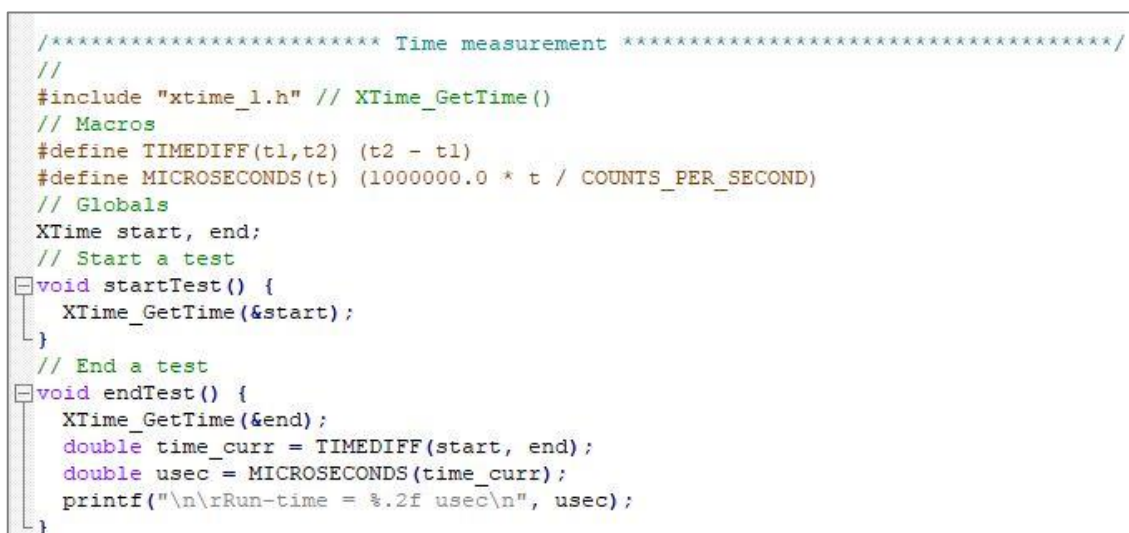


```

1  /*
2  DATA FILE
3  */
4
5  #define index1 222
6
7  u32 pdi_buf[index1] = {
8      0x70000000,
9      0x20000000,
10     ...
11
12     0x12100000};
13
14     #define index2 6
15
16
17     u32 sdi_buf[index2] = {
18         0x40000000,
19         0xc7000010,
20         0x273f94d0,
21         0xd40d0b82,
22         0xf0141f8a,
23         0xeaf8efd0};
24
25     #define index3 195
26
27     u32 do_buf[index3] = {
28         0x520002ee,
29         0xa8b29bd2,
30         ...
31
32         0xe0000000};
33
34

```

Figura 47. Detalle del fichero de cabecera "data.h"



```

/***** Time measurement *****/
//
#include "xtime_1.h" // XTime_GetTime()
// Macros
#define TIMEDIFF(t1,t2) (t2 - t1)
#define MICROSECONDS(t) (1000000.0 * t / COUNTS_PER_SECOND)
// Globals
XTime start, end;
// Start a test
void startTest() {
    XTime_GetTime(&start);
}
// End a test
void endTest() {
    XTime_GetTime(&end);
    double time_curr = TIMEDIFF(start, end);
    double usec = MICROSECONDS(time_curr);
    printf("\n\rRun-time = %.2f usec\n", usec);
}

```

Figura 48. Rutina de medida de tiempo creada para las aplicaciones software de verificación.

4.3.2. Implementación software del cifrador en el sistema empotrado.

La primera tarea de verificación de los cifradores presentados a la competición CAESAR, se llevó a cabo mediante software. Utilizando para ello las fuentes proporcionadas por los autores y que se encuentran disponibles en el repositorio del proyecto ATHENa [13]. En el caso del cifrador AEGIS las fuentes son las siguientes:

```
aes.c,  
api.h,  
crypto_aead.h,  
encrypt.c,  
generate_test_vector_128bitkey_128bitiv.c.
```

Una vez verificado que dichas fuentes son compatibles con las herramientas de desarrollo de Xilinx para el procesador ARM, así como su correcta funcionalidad (la coincidencia de resultados proporcionados por el cifrador y los esperados según la herramienta de verificación de test) el código es modificado para incluir la rutina de cálculo de tiempos explicada en el apartado anterior. Dicha rutina se incluye en el fichero “generate_test_vector_128bitkey_128bitiv.c” que es idéntico para los tres esquemas de conexión del módulo IP descritos en este mismo capítulo. La jerarquía de cada una de las aplicaciones creadas para la verificación de la implementación software del cifrador sobre el sistema empotrado se muestran en la Figura 49. Los tiempos empleados en cifrar y descifrar los mensajes por cada una de ellas se mostrarán en el capítulo siguiente.

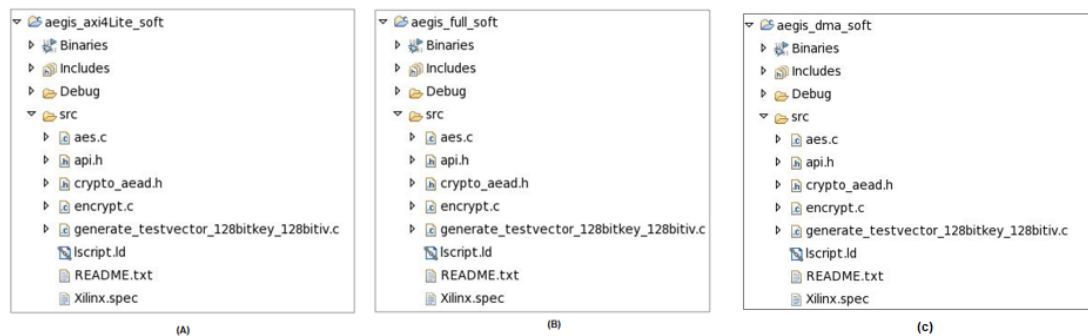


Figura 49. Jerarquía de las aplicaciones de verificación de la implementación software del cifrador en el sistema empotrado. En A) IP conectado con FIFOs a través de buses AXI4-Lite, en B) con buses AXI4 y en C) IP conectado mediante acceso directo a memoria.

4.3.3. Software de verificación del módulo IP (AXI4-Lite y AXI4).

Las aplicaciones utilizadas para la verificación de los sistemas empotrados que utilizan FIFOs para interconectar el módulo IP del cifrador y el procesador ARM son similares, con independencia del tipo de bus (AXI4-Lite o AXI4) utilizado por los módulos “AXI-Stream FIFO”. Por este motivo serán descritos de forma conjunta en este apartado.

Una vez exportada de Vivado a SDK la plataforma hardware correspondiente al sistema empotrado, se ejecuta el programa SDK para desarrollar la aplicación software que permite verificar su funcionamiento. Para ello se genera previamente un *Board Support Package* (BSP) que proporciona rutinas básicas de soporte para manejar distintos elementos del sistema empotrado, e incluye los drivers generados por el asistente utilizado para convertir el diseño del cifrador en un módulo IP.

Con independencia del esquema de conexión utilizado, la aplicación de verificación de los módulos IP puede seguir dos esquemas: uno basado en la consulta de datos de salida del cifrador (*polling*) y el otro basado en la activación de señales de interrupción por parte de algunos componentes del sistema para indicar el final de sus operaciones. En ambos casos estas aplicaciones incorporan una rutina de medida de tiempos para comparar los rendimientos de diferentes estrategias.

A) Verificación del cifrador mediante *Polling*.

La primera alternativa desarrollada para verificar el funcionamiento del módulo IP de cifrado usa técnicas de *polling* para determinar cuándo termina la transmisión de los datos públicos de salida generados por el módulo IP de cifrado (en este caso, AEGIS). La aplicación está compuesta de varias fuentes las cuales son:

aegis_poll.c,
data.h.

El fichero de cabecera data.h contiene todos los datos que se van a transmitir al módulo IP de cifrado por medio del programa principal. Este programa principal, aegis_poll, es el mismo para los sistemas empotrados con AXI4 y AXI4-Lite. El programa lleva cabo una función denominada “TestAegisPoll” que se encarga de realizar varias tareas: comienza su ejecución inicializando la configuración de las FIFOs que se van a encargar de transmitir los datos públicos y privados y recibir posteriormente los datos de salida. Una vez se comprueba que se han configurado correctamente, empieza el test. Para llevarlo a cabo, la variable “Status” es la encargada de ejecutar las funciones “TxSend” para transmitir los datos y “RxReceive” para recoger los resultados obtenidos.

Primero, se transmiten los datos públicos (los cuales deben contener menos de 500 valores para no exceder la profundidad de las FIFOs implementadas) y los datos secretos, a los que, al ser de menor tamaño, no es necesario ponerle ninguna restricción de tamaño. Tras esto, se recogen los datos de salida por medio de la misma FIFO que transmitió los datos públicos. Para finalizar el test, se lleva a cabo una función de comparación entre los datos del fichero “do.txt” y los datos recogidos por la FIFO, los cuales deben coincidir.

Para visualizar el proceso, se puede usar el puerto serie y para ello se han añadido distintos modos de depuración “Debug Level”. Este mecanismo permite elegir la información mostrada por la aplicación: ninguna información (Debug = 0), información sobre la inicialización correcta de los componentes (Debug = 1), información enviada y recibida hacia y desde el cifrador (Debug = 2).

Una vez compilada la aplicación, el proyecto SDK contendrá, además de las fuentes, el fichero binario “.elf” que permite ejecutarla en el procesador ARM. La placa de desarrollo que previamente debe ser programada con el hardware correspondiente. También es necesario configurar cómo se va a ejecutar la aplicación ya que SDK ofrece distintas alternativas. La utilizada ha sido la configuración “System Debugger” ya sea en local o en remoto, como ha sido el caso. Las jerarquías de los proyectos realizados con AXI4 y AXI4-Lite se muestran en la Figura 50.

Una vez iniciada la ejecución de la aplicación software, se pueden visualizar los resultados mediante cualquier aplicación de terminal serie conectada al puerto serie. Los resultados obtenidos serán mostrados en el capítulo siguiente.

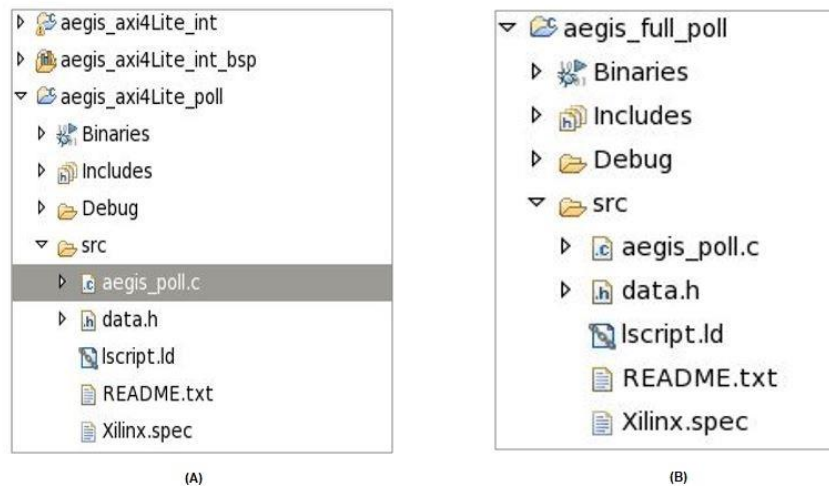


Figura 50. Fuentes de la aplicación basada en *polling* para los sistemas empotrados que utilizan buses AXI4-Lite en A) y AXI4 en B).

B). Verificación cifrador mediante interrupciones.

La otra alternativa empleada para verificar la operación del módulo IP de cifrado usa interrupciones como método para determinar la finalización del envío de datos entre el módulo IP de cifrado y el procesador. La aplicación está compuesta de las siguientes fuentes:

aegis_int.c,
data.h.

La utilidad del fichero de cabecera data.h ha sido explicada anteriormente en el apartado 4.3.1. Este fichero es incluido por el programa principal “aegis_int”, que va a ser el mismo tanto para el sistema empotrado que utiliza AXI4-Lite como para el que usa AXI4. La función principal del programa ejecutado en este caso se denomina

“TestAegisInt”. Esta función realiza diversas tareas: la primera consiste en definir la configuración de las FIFOs para que envíen y reciban los datos correspondientes. Una vez configuradas correctamente las FIFOs, se configuran las interrupciones, mediante la función “SetupInterruptSystem” y comienza el test, con la llamada a la función “TxSend” para enviar los datos públicos y secretos del cifrador. A continuación el programa entra en un ciclo infinito a la espera de que se produzca una interrupción indicando que la FIFO conectada a la salida del cifrador ha recibido los datos generados por este.

“aegis_int.c” incluye también la definición de las funciones de manejo de interrupción: “FifoHandler”, “FifoRecvHandler”, “FifoSendHandler” y “FifoErrorHandler”. Cuando el procesador recibe una interrupción, la función “FifoHandler” se encarga de llamar a la rutina concreta encargada de atenderla y, posteriormente, reinicializa la máscara de interrupción correspondiente con la función “XLIFifo IntClear” para evitar que los datos sean procesados nuevamente.

Cuando la interrupción indica que la FIFO ha recibido los datos de salida del cifrador el procesador ejecuta la función “FifoRecvHandler” para acceder a dichos datos. Por último, para verificar que se ha ejecutado la aplicación correctamente, estos datos son comparados con la salida esperada obtenida al generar los vectores de test.

Al igual que en la aplicación anterior, se ha añadido código para seleccionar distintos modos de depuración “Debug”, que proporcionan diferentes niveles de detalle sobre los resultados de la prueba a través de la salida estándar del sistema (habitualmente conectada al puerto serie de la placa de desarrollo). Debug = 0 solo muestra el resultado de la rutina de medida de tiempos, Debug = 1 incluye, además, información sobre la inicialización de las FIFOs y así hasta cuatro modos de depuración.

Las aplicaciones de verificación basadas en interrupciones incorporan la misma función de medida de tiempo usada anteriormente, de forma que sea posible, comparar mediante medidas objetivas los rendimientos de las diferentes estrategias.

Los procesos de compilación de la aplicación y de generación y uso del fichero ejecutable son similares a los del apartado anterior. La jerarquía de las aplicaciones software basadas en interrupciones para sistemas conectados mediante buses AXI4-Lite y AXI4 se muestra en la Figura 51.

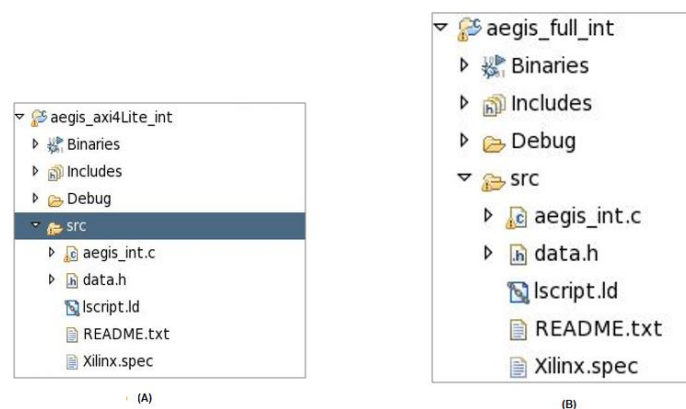


Figura 51. Fuentes de la aplicación basada en interrupciones para los sistemas empotrados que emplean buses AXI4-Lite en (A) y con buses AXI4 en (B).

4.3.4. Software del módulo IP (AXI DMA)

En este apartado, se describen las aplicaciones utilizadas para verificar la funcionalidad del módulo IP del cifrador cuando este se interconecta con el resto del sistema mediante un módulo AXI DMA que le permite acceder directamente a la memoria del sistema para recibir y enviar los datos públicos sin la intervención del procesador. También para esta configuración del sistema empujado es posible seguir una estrategia masada en *polling* [24], o en interrupciones [25]. Al utilizar un esquema de conexionado entre el procesador y el módulo IP cifrador distinto al de los sistemas anteriores, el software de verificación requiere algunas modificaciones.

A) Verificación del cifrador mediante *polling*.

Cuando se usa el método de consulta para la recepción de los datos públicos del módulo IP de cifrado, las fuentes que componen la aplicación son las siguientes:

```
aegis_DMA_poll.c,  
data.h.
```

Al igual que en los casos anteriores, el fichero de cabecera “data.h” contiene los datos que se van a enviar al cifrador y los que se espera recibir del mismo. El programa principal “aegis_DMA_poll” lleva a cabo la función “Aegis_DMA_SimplePoll”, con una estructura muy similar a las ya vistas. Comienza inicializando el dispositivo “XAXiDMA” y configurándolo de tal forma que inhabilite las interrupciones. Una vez hecho esto ya está configurado para la transmisión de los datos públicos y recepción de los datos de salida, tal como se explicó en el apartado 4.2.3.3. Tras configurar el acceso directo a memoria, se configura la FIFO que se encarga de transmitir los datos secretos al cifrador. En este caso tampoco es necesaria ninguna restricción de tamaño pues sigue siendo muy inferior al máximo permitido. Al terminar la configuración, y antes de iniciar el test, el programa almacena en la zona de memoria controlada por el módulo AXI-DMA los datos públicos definidos en el fichero de cabecera. Al llevar a cabo este paso se modifica el formato con que se escriben los datos en memoria para adaptarlo al requerido por el cifrador.

Una vez se inicia el test de verificación, se transmiten en primer lugar los datos secretos utilizando la misma función “TxSend” descrita previamente. Para transmitir los datos públicos, primero se ejecuta la función “Xil_DcacheFlushRange” para limpiar los búfers internos en caso de que esté habilitada la caché de datos. Tras el envío de los datos públicos con la la función “XAXiDma_SimpleTransfer”, se espera a que el AXI DMA complete la transferencia de los datos de salida, para lo que se emplea la función “XAXiDma_Busy”. Como en los casos anteriores, cuando el test termina se ejecuta la función que compara los datos recibidos del cifrador con los datos esperados que aparecían en los ficheros “do.txt” y “data.h”. Para finalizar, se resetea el módulo AXI DMA.

La jerarquía del proyecto se muestra en la Figura 52.

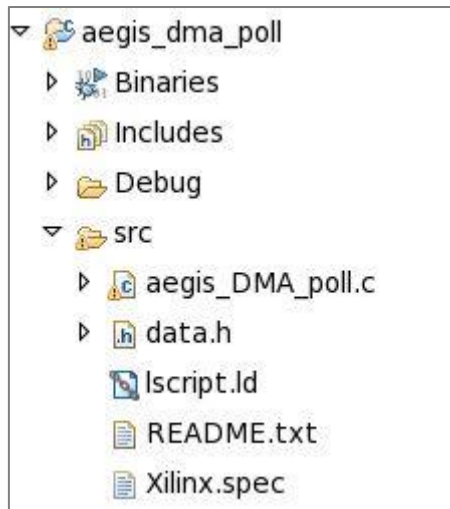


Figura 52. Fuentes de la aplicación de verificación basada en *polling* para configuraciones del sistema empotrado que emplea buses AXI-DMA.

B) Verificación del cifrador mediante interrupciones.

La otra estrategia de verificación del sistema empotrado que emplea acceso directo a memoria usa interrupciones identificar el final de la operación del IP de cifrado. La aplicación está compuesta por las siguientes fuentes:

aegis_DMA_int.c,
data.h.

Al igual que en las aplicaciones anteriores, el fichero de cabecera “data.h” contiene los datos que se van a enviar y se esperan recibir del cifrador. El programa principal “aegis_DMA_int” comienza con la configuración del módulo “XAXiDMA” encargado de la transferencia de los datos públicos de entrada y salida, así como de las FIFO a través de las que se envían los datos secretos. Con ambas configuraciones realizadas, y tras configurar el esquema de interrupciones, el programa comienza el test de verificación transmitiendo en primer lugar los datos secretos, que y a continuación los datos públicos utilizando las funciones comentadas en el apartado anterior. A continuación la ejecución del programa entra en un lazo infinito hasta que se activan las banderas “TxDone” y “RxDone”.

Estas banderas son controladas por las rutinas de manejo de interrupciones, “SetupIntrSystem”, “RxIntrHandler” y “TxIntrHandler”, cuya funcionalidad es similar a la descrita en el apartado 4.3.3-B aunque utilizan rutinas del sistema diferentes.

Cuando se ejecutan las rutinas de interrupción encargadas de enviar los datos públicos de entrada y recibir los datos de salida del cifrador las banderas “TxDone” y “RxDone” se ponen a 1, el programa sale del ciclo de espera y acaba el test para proceder a comparar los datos obtenidos con los esperados. Finalmente se produce un reset del bloque AXI DMA para dejar al sistema listo para una ejecución posterior de la aplicación. La jerarquía del proyecto se muestra en la Figura 53.

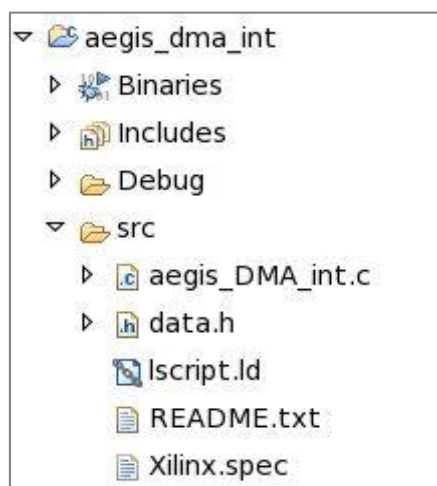


Figura 53. Jerarquía de la aplicación de verificación mediante interrupciones para la configuración del sistema empotrado que emplea AXI-DMA.

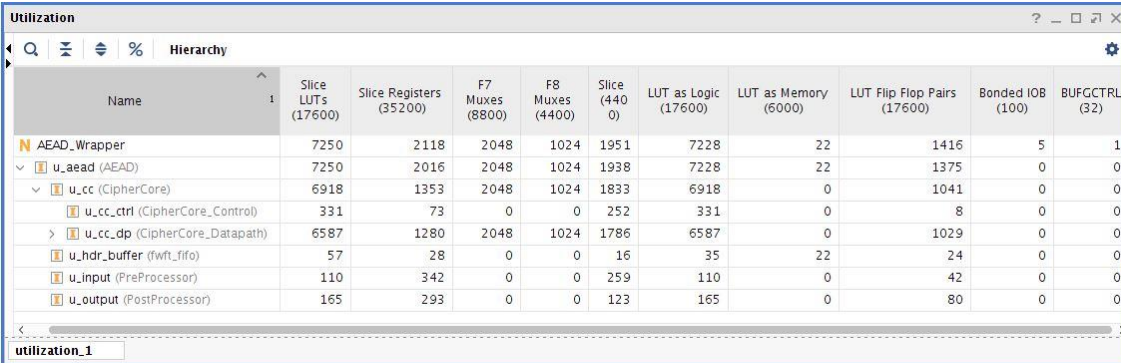
CAPÍTULO 5

Resultados obtenidos

En este capítulo se presentan de forma conjunta los resultados obtenidos en las distintas etapas de desarrollo hardware y software descritas en el capítulo anterior. En primer lugar se analiza el resultado de la implementación hardware del cifrador AEGIS utilizando la estrategia por defecto de síntesis e implementación proporcionadas por el entorno Vivado. A continuación se analizan el consumo de potencia y la ocupación de recursos de la FPGA cuando se incorpora este cifrador como módulo IP de sistemas empujados que utilizan los diferentes esquemas de conexión discutidos en el capítulo 4. Los resultados de verificación de los sistemas empujados se incluyen también en el capítulo, lo que permite evaluar los rendimientos alcanzados por cada uno de ellos (usando estrategias basadas en consultas sucesivas o en interrupciones), así como comparar estos resultados con los que proporciona la implementación software del cifrador, considerando los efectos de habilitar o deshabilitar las cachés de instrucciones y/o datos del procesador ARM. En el capítulo se recogen, por último, los principales resultados obtenidos al aplicar el flujo de diseño propuesto en este trabajo a otro de los cifradores finalistas de la competición CAESAR, el cifrador ACORN, lo que permite, a su vez, comparar resultados de cifradores englobados en dos de las diferentes categorías establecidas en dicha competición.

5.1. Implementación hardware del cifrador

Las herramientas de síntesis e implementación del entorno Vivado pueden ser configuradas de acuerdo con distintas estrategias que permiten obtener soluciones especialmente adaptadas para cumplir un determinado objetivo de diseño (consumo de recursos, velocidad de operación, tiempo de desarrollo, etc.), así como establecer compromisos adecuados entre estos objetivos. Los resultados post-Implementación hardware del cifrador AEGIS-32 descrito en el apartado 4.2.2 cuando utiliza la estrategia de síntesis por defecto e implementación por defecto se muestran en la Figura 54 y Figura 55.



Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	Slice (440 0)	LUT as Logic (17600)	LUT as Memory (6000)	LUT Flip Flop Pairs (17600)	Bonded IOB (100)	BUFGCTRL (32)
AEAD_Wrapper	7250	2118	2048	1024	1951	7228	22	1416	5	1
u_aead (AEAD)	7250	2016	2048	1024	1938	7228	22	1375	0	0
u_cc (CipherCore)	6918	1353	2048	1024	1833	6918	0	1041	0	0
u_cc_ctrl (CipherCore_Control)	331	73	0	0	252	331	0	8	0	0
u_cc_dp (CipherCore_DataPath)	6587	1280	2048	1024	1786	6587	0	1029	0	0
u_hdr_buffer (fwft_fifo)	57	28	0	0	16	35	22	24	0	0
u_input (PreProcessor)	110	342	0	0	259	110	0	42	0	0
u_output (PostProcessor)	165	293	0	0	123	165	0	80	0	0

Figura 54. Utilización detallada de los recursos de la FPGA tras la implementación de AEGIS-32.

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	LUT Flip Flop Pairs (17600)	Bonded IOB (100)	BUFGCTRL (32)
AEAD_Wrapper	41.19%	6.02%	23.27%	23.27%	44....	41.07%	0.37%	8.05%	5.00%	3.13%
u_aead (AEAD)	41.19%	5.73%	23.27%	23.27%	44....	41.07%	0.37%	7.81%	0.00%	0.00%
u_cc (CipherCore)	39.31%	3.84%	23.27%	23.27%	41....	39.31%	0.00%	5.91%	0.00%	0.00%
u_cc_ctrl (CipherCore_Control)	1.88%	0.21%	0.00%	0.00%	5.7...	1.88%	0.00%	0.05%	0.00%	0.00%
u_cc_dp (CipherCore_Datapath)	37.43%	3.64%	23.27%	23.27%	40....	37.43%	0.00%	5.85%	0.00%	0.00%
u_hdr_buffer (hwft_fifo)	0.32%	0.08%	0.00%	0.00%	0.3...	0.20%	0.37%	0.14%	0.00%	0.00%
u_input (PreProcessor)	0.63%	0.97%	0.00%	0.00%	5.8...	0.63%	0.00%	0.24%	0.00%	0.00%
u_output (PostProcessor)	0.94%	0.83%	0.00%	0.00%	2.8...	0.94%	0.00%	0.45%	0.00%	0.00%

Figura 55. Utilización detallada porcentual de los recursos de la FPGA tras la implementación de AEGIS-32.

5.2. Incorporación del cifrador en un sistema empotrado

En este apartado se muestran los resultados, en términos de consumo de recursos de la FPGA, de cada uno de los sistemas empotrados descritos en el apartado 4.2.3 y que usan el módulo IP del cifrador autenticado AEGIS desarrollado en el apartado 4.2.3.

5.2.1. Conexión del módulo IP mediante FIFOs

Como era lógico esperar, los resultados de ocupación y consumo de potencia varían en función del tipo de bus (AXI4-Lite o AXI4) empleado para conectar el módulo IP al procesador a través de FIFOs. Para cada una de estas configuraciones se obtiene:

A) FIFOs conectados con buses AXI4-Lite:

El consumo total de recursos de la FPGA usando esta configuración se muestra en la Figura 56, Figura 57 y Figura 58.

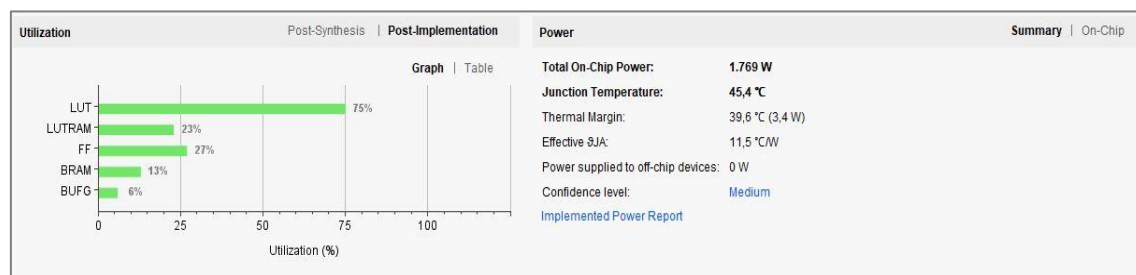


Figura 56. Utilización de la FPGA tras la implementación del sistema empotrado utilizando FIFOs conectadas mediante AXI4-Lite.

También se ha obtenido el resultado detallado en términos de consumo de potencia, que se muestra en la Figura 59

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	LUT Flip Flop Pairs (17600)	Block RAM Tile (60)
aegis_wrapper	13284	9344	2493	1024	4155	11926	1358	4265	7.5
aegis_i (aegis)	12786	8593	2492	1024	3955	11452	1334	3923	7.5
aegis_0 (aegis_ae...)	8352	2191	2432	1024	2211	7562	790	1505	0
axi_fifo_mm_s_0 (a...)	601	634	0	0	240	540	61	247	2
axi_fifo_mm_s_1 (a...)	275	362	0	0	128	274	1	127	1
ila_0 (aegis_ila_0_0)	998	1574	20	0	420	858	140	572	1.5
ila_1 (aegis_ila_0_1)	997	1574	20	0	430	857	140	570	1.5
ila_2 (aegis_ila_1_0)	998	1574	20	0	462	858	140	563	1.5
processing_system...	0	0	0	0	0	0	0	0	0
ps7_0_axi_periph (...)	489	647	0	0	191	428	61	261	0
rst_ps7_0_100M (a...)	18	37	0	0	11	17	1	16	0
dbg_hub (dbg_hub)	498	751	1	0	218	474	24	340	0

Figura 57. Utilización de recursos de la FPGA tras la implementación del sistema empotrado utilizando FIFOs conectadas mediante buses AXI4-Lite.

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	LUT Flip Flop Pairs (17600)	Block RAM Tile (60)
aegis_wrapper	75.48%	26.55%	28.33%	23.27%	94...	67.76%	22.63%	24.23%	12.50%
aegis_i (aegis)	72.65%	24.41%	28.32%	23.27%	89...	65.07%	22.23%	22.29%	12.50%
aegis_0 (aegis_ae...)	47.45%	6.22%	27.64%	23.27%	50...	42.97%	13.17%	8.55%	0.00%
axi_fifo_mm_s_0 (a...)	3.41%	1.80%	0.00%	0.00%	5.4...	3.07%	1.02%	1.40%	3.33%
axi_fifo_mm_s_1 (a...)	1.56%	1.03%	0.00%	0.00%	2.9...	1.56%	0.02%	0.72%	1.67%
ila_0 (aegis_ila_0_0)	5.67%	4.47%	0.23%	0.00%	9.5...	4.88%	2.33%	3.25%	2.50%
ila_1 (aegis_ila_0_1)	5.66%	4.47%	0.23%	0.00%	9.7...	4.87%	2.33%	3.24%	2.50%
ila_2 (aegis_ila_1_0)	5.67%	4.47%	0.23%	0.00%	10...	4.88%	2.33%	3.20%	2.50%
processing_system...	0.00%	0.00%	0.00%	0.00%	0.0...	0.00%	0.00%	0.00%	0.00%
ps7_0_axi_periph (...)	2.78%	1.84%	0.00%	0.00%	4.3...	2.43%	1.02%	1.48%	0.00%
rst_ps7_0_100M (a...)	0.10%	0.11%	0.00%	0.00%	0.2...	0.10%	0.02%	0.09%	0.00%
dbg_hub (dbg_hub)	2.83%	2.13%	0.01%	0.00%	4.9...	2.69%	0.40%	1.93%	0.00%

Figura 58. Utilización porcentual de la FPGA tras la implementación del sistema empotrado utilizando FIFOs conectadas mediante buses AXI4-Lite.

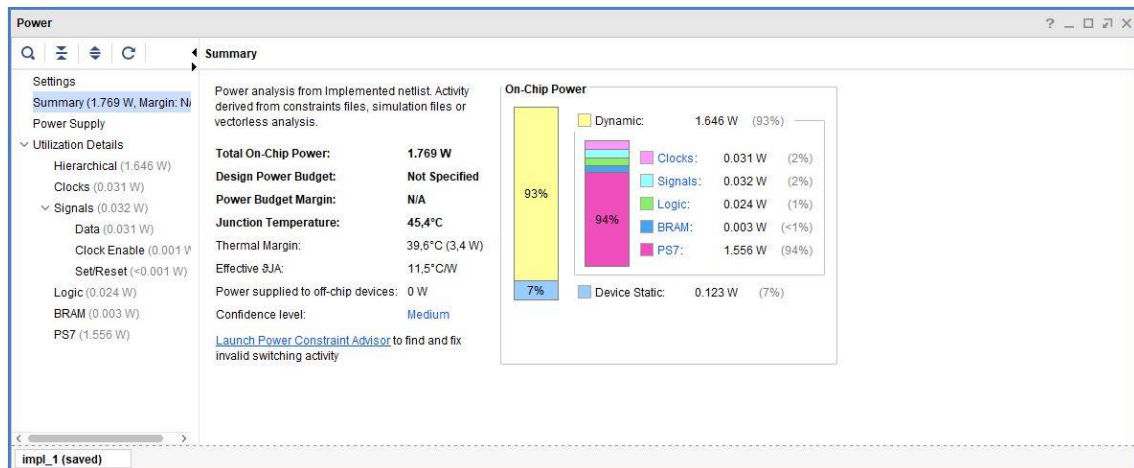


Figura 59. Potencia consumida por la FPGA para el sistema empotrado que utiliza FIFOs conectadas mediante buses AXI4-Lite.

B) FIFOs conectadas con buses AXI4.

El consumo de recursos de la FPGA para esta configuración del sistema empotrado se muestra en la Figura 60 y Figura 61. Como puede observarse, el empleo de un bus AXI4 con mayor funcionalidad para transferir los datos públicos del cifrador implica un incremento de algo más del 2% en la ocupación de recursos de la FPGA.

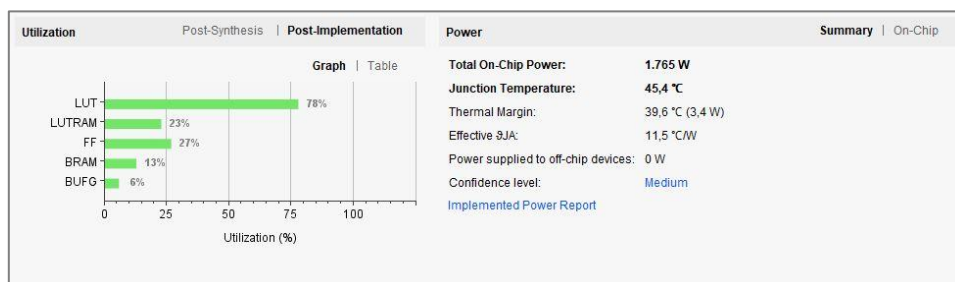


Figura 60 Utilización de la FPGA tras la implementación del sistema empotrado utilizando una FIFO conectada mediante bus AXI4.

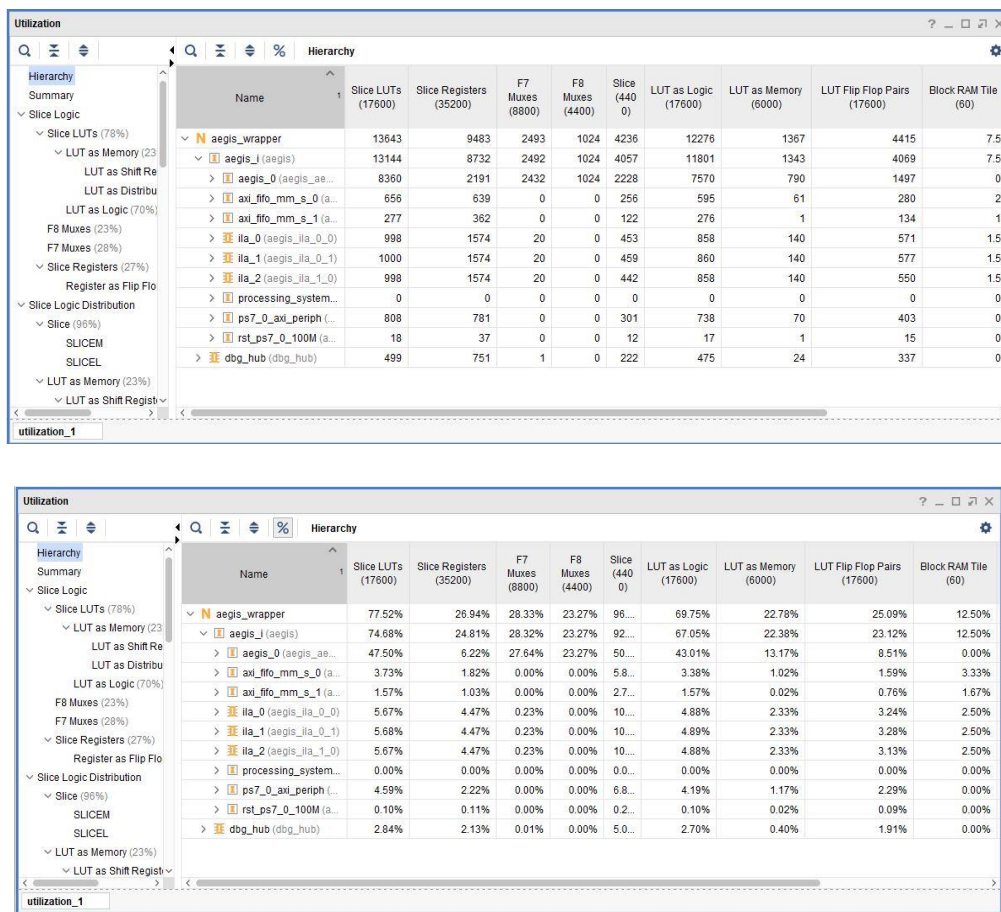


Figura 61.Utilización de los recursos de la FPGA tras la implementación del sistema empotrado utilizando una FIFO conectada al bus AXI4.

El resultado de potencia consumida en este caso es similar al de la configuración anterior. El consumo se muestra en la Figura 62.

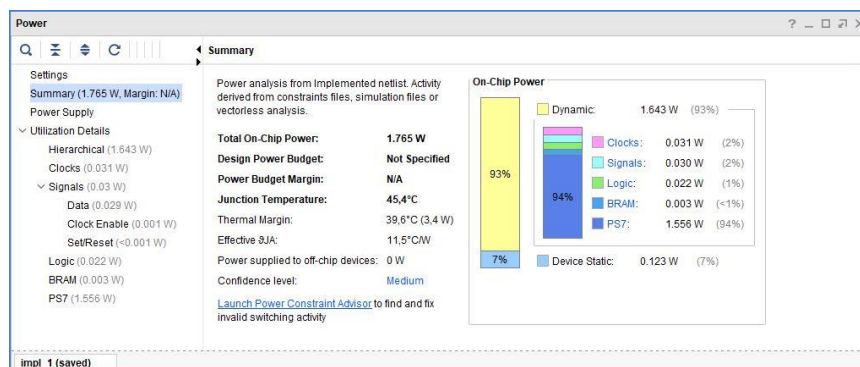


Figura 62. Potencia consumida por la FPGA para el sistema empotrado que emplea una FIFO conectada mediante bus AXI4.

5.2.2. Conexión del módulo IP usando DMA.

La configuración descrita en el apartado B) incluye un módulo AXI DMA para realizar la transferencia de datos públicos hacia y desde el cifrador. Los resultados de ocupación de recursos y consumo de potencia obtenidos tras la implementación de este sistema empotrado se muestran en la Figura 63, Figura 64 y Figura 65. Al contener el sistema empotrado componentes adicionales a los de las configuraciones anteriores es normal que la utilización de recursos de la FPGA sea mayor.



Figura 63. Utilización de la FPGA tras la implementación del sistema empotrado que utiliza DMA.

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	LUT Flip Flop Pairs (17600)	Block RAM Tile (60)
N aegis_wrapper	13643	9483	2493	1024	4236	12276	1367	4415	7.5
I aegis_l (aegis)	13144	8732	2492	1024	4057	11801	1343	4069	7.5
I aegis_0 (aegis_ae...)	8360	2191	2432	1024	2228	7570	790	1497	0
I axi_fifo_mm_s_0 (a...)	656	639	0	0	256	595	61	280	2
I axi_fifo_mm_s_1 (a...)	277	362	0	0	122	276	1	134	1
I ila_0 (aegis_ila_0_0)	998	1574	20	0	453	858	140	571	1.5
I ila_1 (aegis_ila_0_1)	1000	1574	20	0	459	860	140	577	1.5
I ila_2 (aegis_ila_1_0)	998	1574	20	0	442	858	140	560	1.5
I processing_system...	0	0	0	0	0	0	0	0	0
I ps7_0_axi_periph (...)	808	781	0	0	301	738	70	403	0
I rst_ps7_0_100M (a...)	18	37	0	0	12	17	1	15	0
I dbg_hub (dbg_hub)	499	751	1	0	222	475	24	337	0

Figura 64. Utilización de los recursos de la FPGA tras la implementación del sistema empotrado que utiliza DMA.

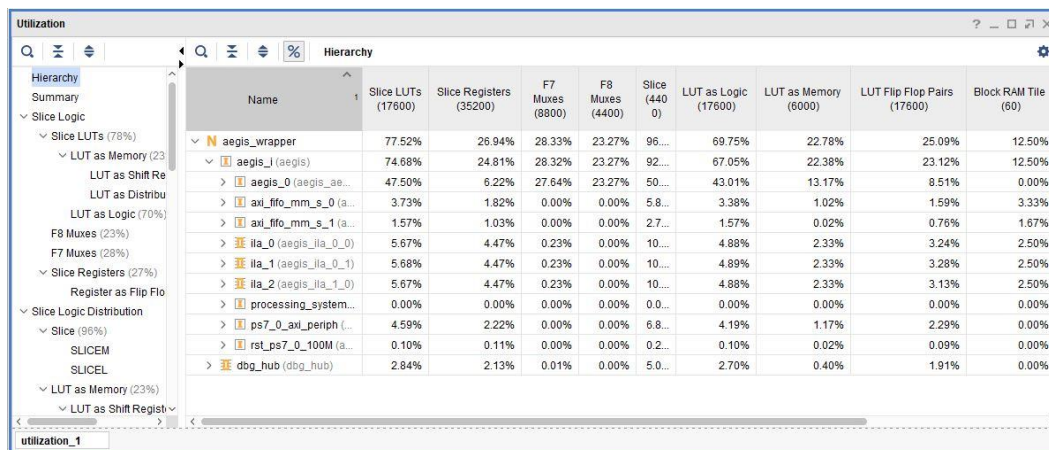


Figura 65. Utilización porcentual de los recursos de la FPGA tras la implementación del sistema empotrado que utiliza DMA.

En cuanto a la potencia consumida, también es de esperar que sea mayor que en las configuraciones anteriores. Dicha potencia consumida se detalla en la Figura 66.

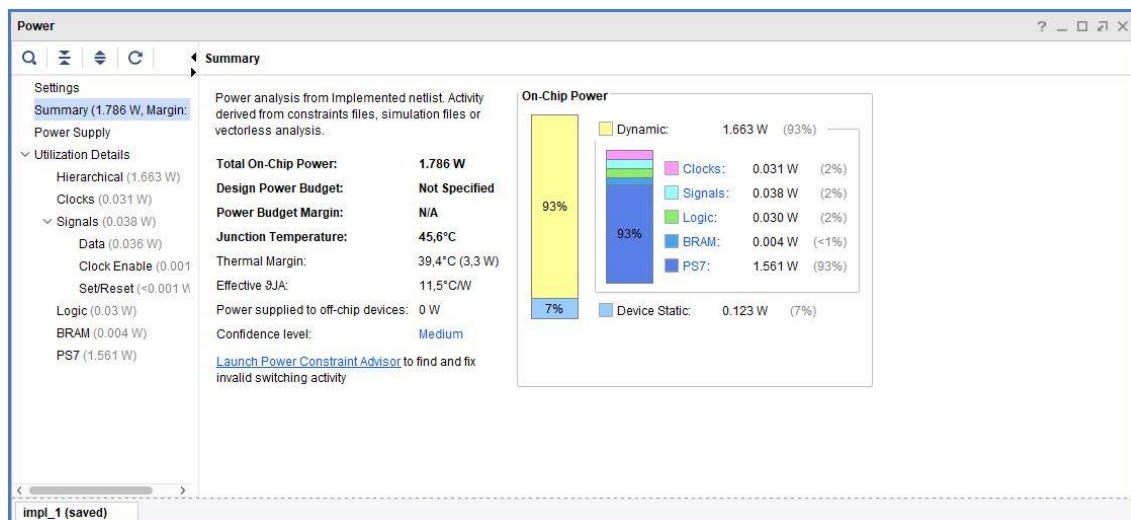


Figura 66. Potencia consumida por la FPGA tras implementar el sistema que emplea el módulo AXI DMA.

5.3. Verificación de los sistemas empotrados

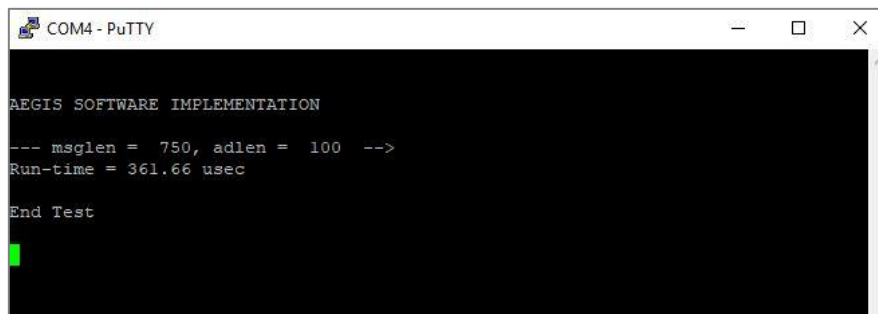
En este apartado se van a mostrar los tiempos de ejecución de cada una de las aplicaciones software desarrolladas en el apartado 4.3 para verificar el correcto funcionamiento de los sistemas empotrados. Se comenzará analizando la implementación software del cifrador sobre el sistema empotrado, para estudiar, posteriormente las aplicaciones de verificación mediante *polling* e interrupciones para cada uno de los sistemas empotrados. Todos los programas de verificación admiten

distintos niveles de depuración, aunque será el nivel 0 el elegido para comparar los tiempos de cifrado de los sistemas empotrados (ya que los niveles de depuración superiores provocan numerosas operaciones de salida que impiden calcular con exactitud los tiempos realmente invertidos en la operación del módulo de cifrado). Por otra parte, la verificación de la implementación software permite habilitar o deshabilitar las cachés de datos y de instrucciones, lo que nos va a permitir evaluar las ventajas que supone el uso de estas cachés en la plataforma utilizada para soportar los sistemas empotrados.

5.3.1. Implementación del cifrador mediante software empotrado.

La primera aplicación de verificación es, la verificación por software realizada con el sistema empotrado con buses AXI4-Lite: En esta verificación se ejecuta el algoritmo sobre el procesador con la misma configuración de memoria, por lo que los resultados son independientes de la plataforma seleccionada.

Se han obtenido los resultados mostrados en la Figura 67, Figura 68 y Figura 69. La primera muestra el mejor resultado mientras que las otras dos figuras muestran los tiempos si deshabilitamos las cachés



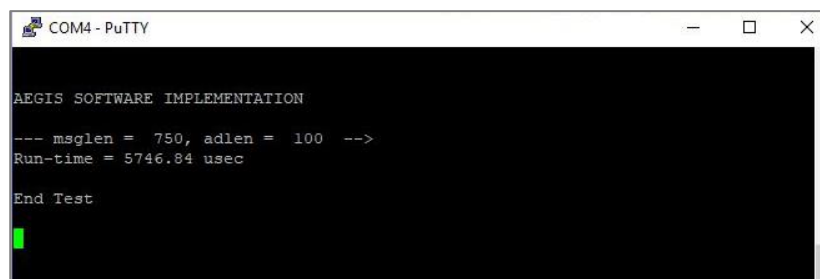
```
COM4 - PuTTY

AEGIS SOFTWARE IMPLEMENTATION

--- msglen = 750, adlen = 100 -->
Run-time = 361.66 usec

End Test
```

Figura 67. Tiempo de ejecución de la implementación software del cifrador con las cachés de datos e instrucciones habilitadas



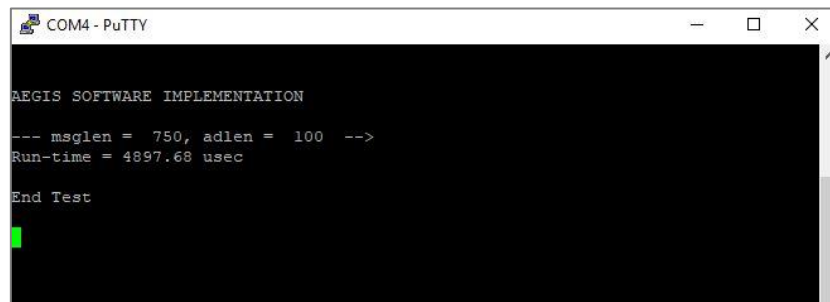
```
COM4 - PuTTY

AEGIS SOFTWARE IMPLEMENTATION

--- msglen = 750, adlen = 100 -->
Run-time = 5746.84 usec

End Test
```

Figura 68. Tiempo de ejecución de la implementación software del cifrador sin la caché de datos ni la de instrucciones habilitadas.



```
COM4 - PuTTY

AEGIS SOFTWARE IMPLEMENTATION

--- msglen = 750, adlen = 100 -->
Run-time = 4897.68 usec

End Test

█
```

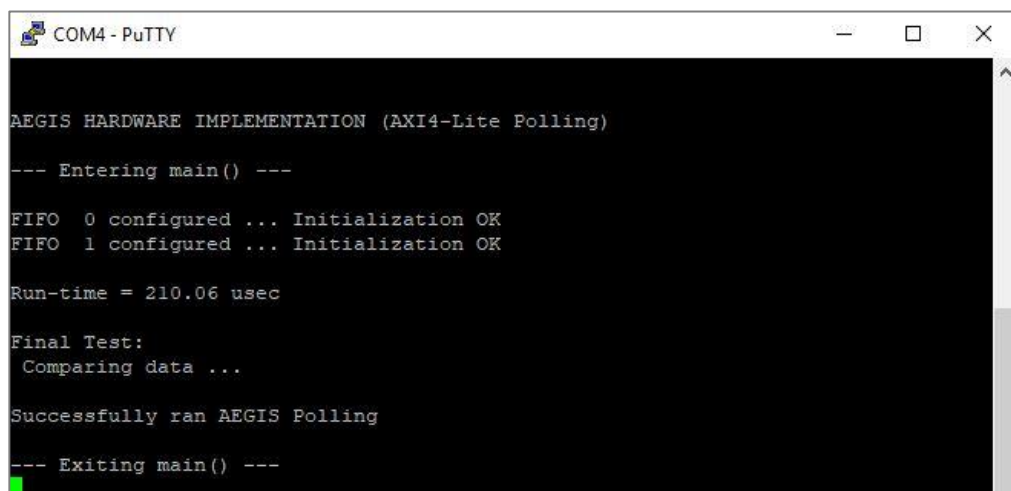
Figura 69. Tiempo de ejecución de la implementación software del cifrador sin la caché de datos habilitada.

5.3.2. Sistemas empotrados con FIFOs (AXI4-Lite y AXI4).

Los resultados obtenidos al ejecutar las aplicaciones de verificación basadas en *polling* e interrupciones para evaluar el rendimiento de los sistemas empotrados que utilizan FIFOs son los siguientes:

A) Conexión de FIFOs mediante buses AXI4-Lite.

La Figura 70 muestra la salida obtenida tras la ejecución sobre el sistema empotrado que utiliza este esquema de conexión de la aplicación de verificación basada en *polling*. La salida corresponde al uso del modo “Debug=0”, utilizado para obtener el tiempo de cifrado proporcionado por la implementación hardware del cifrador. Por otra parte, en la Figura 71 se muestra la salida de la aplicación en el modo “Debug=1”, que indica si se han inicializado correctamente las FIFOs y si se está desarrollando el cifrado con normalidad. Por las razones comentadas anteriormente, el tiempo invertido en el cifrado no es en este caso significativo.



```
COM4 - PuTTY

AEGIS HARDWARE IMPLEMENTATION (AXI4-Lite Polling)

--- Entering main() ---

FIFO 0 configured ... Initialization OK
FIFO 1 configured ... Initialization OK

Run-time = 210.06 usec

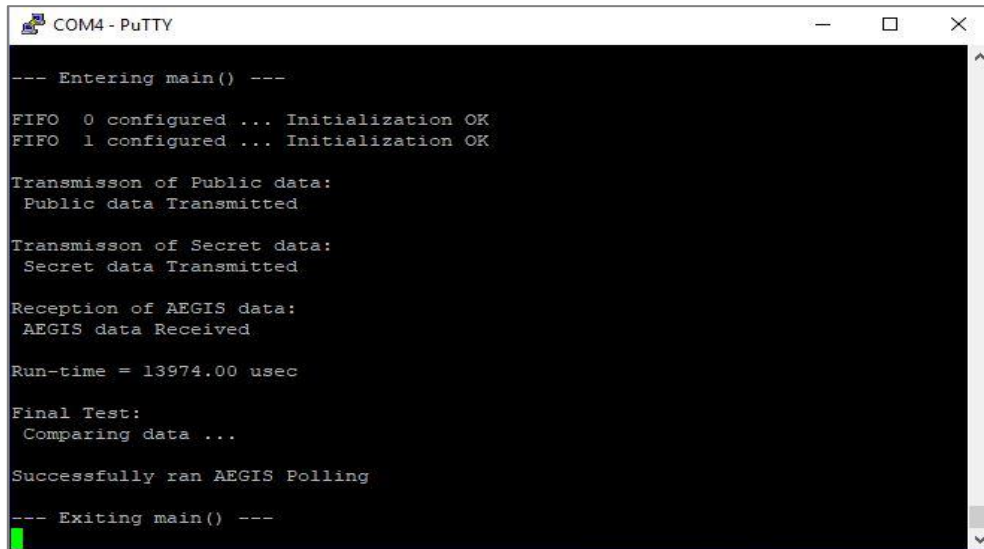
Final Test:
Comparing data ...

Successfully ran AEGIS Polling

--- Exiting main() ---

█
```

Figura 70. Ejecución de la aplicación de verificación mediante polling con el modo de depuración Debug 0 en el sistema con FIFOs y AXI4-Lite.



```
COM4 - PuTTY

--- Entering main() ---

FIFO 0 configured ... Initialization OK
FIFO 1 configured ... Initialization OK

Transmisson of Public data:
Public data Transmitted

Transmisson of Secret data:
Secret data Transmitted

Reception of AEGIS data:
AEGIS data Received

Run-time = 13974.00 usec

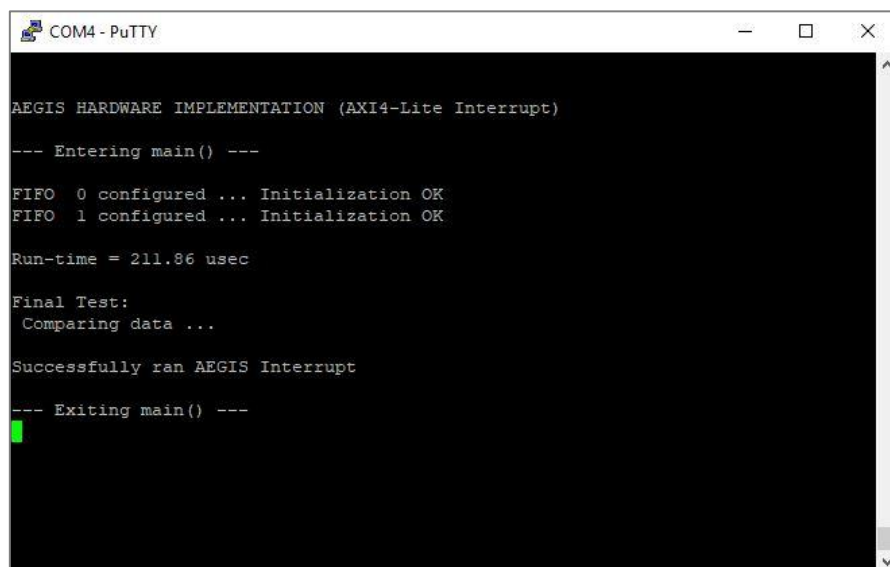
Final Test:
Comparing data ...

Successfully ran AEGIS Polling

--- Exiting main() ---
```

Figura 71. Ejecución de la verificación mediante *polling* con el modo de depuración Debug 1 en el sistema con FIFOs y AXI4-Lite.

En cuanto a las aplicaciones de verificación basadas en interrupciones, igualmente se han ejecutado usando distintos modos de depuración. El modo de depuración “Debug=0”, es de nuevo el elegido para evaluar el tiempo de cifrado de esta configuración y compararlo con el resto de las realizadas. También se puede nuevamente ejecutar en el modo de depuración “Debug=1” para verificar que el proceso se ejecuta correctamente. En la Figura 72, se muestra la verificación en modo “Debug=0”, mientras que en la Figura 73 se muestra la verificación en modo “Debug=1”.



```
COM4 - PuTTY

AEGIS HARDWARE IMPLEMENTATION (AXI4-Lite Interrupt)

--- Entering main() ---

FIFO 0 configured ... Initialization OK
FIFO 1 configured ... Initialization OK

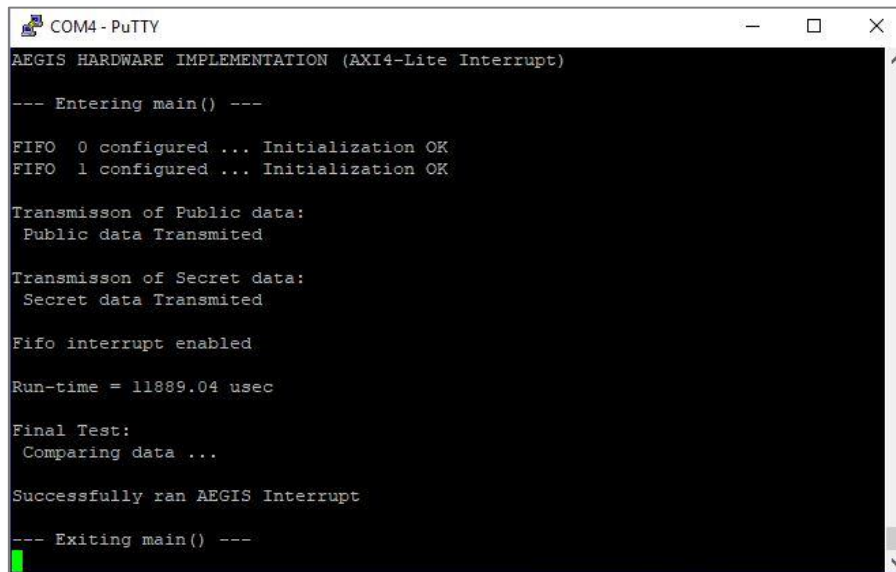
Run-time = 211.86 usec

Final Test:
Comparing data ...

Successfully ran AEGIS Interrupt

--- Exiting main() ---
```

Figura 72. Ejecución de la aplicación de verificación por interrupciones en modo Debug 0 en el sistema con FIFOs y AXI4-Lite.



```
COM4 - PuTTY
AEGIS HARDWARE IMPLEMENTATION (AXI4-Lite Interrupt)
--- Entering main() ---
FIFO 0 configured ... Initialization OK
FIFO 1 configured ... Initialization OK
Transmisson of Public data:
Public data Transmited
Transmisson of Secret data:
Secret data Transmited
Fifo interrupt enabled
Run-time = 11889.04 usec
Final Test:
Comparing data ...
Successfully ran AEGIS Interrupt
--- Exiting main() ---
```

Figura 73. Ejecución de la aplicación de verificación por interrupciones en modo Debug 1 en el sistema con FIFOs y AXI4-Lite.

B) Conexionado de una FIFO mediante bus AXI4.

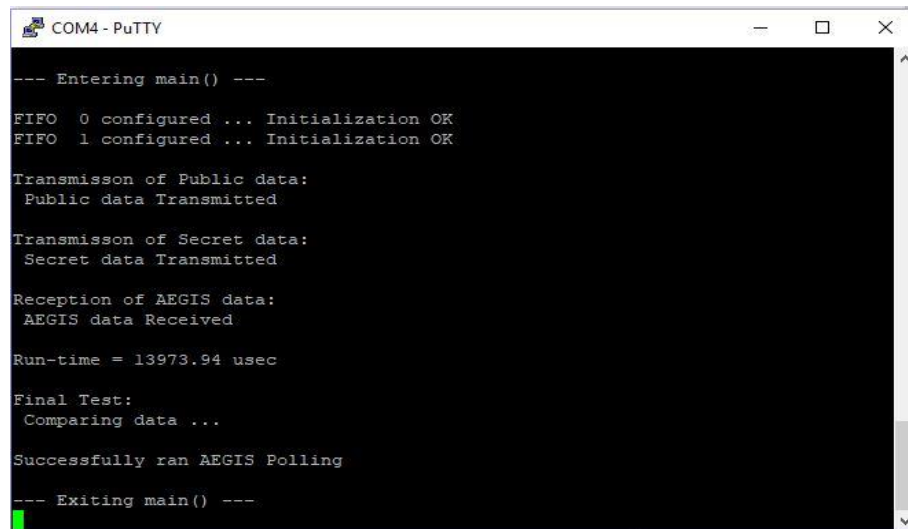
Siguiendo la misma metodología que en el caso anterior, se han obtenido los tiempos que invierte el cifrador en realizar su función cuando el módulo IP está conectado al resto del sistema siguiendo este otro esquema de conexión.

Los resultados de ejecutar la aplicación de verificación basada en *polling* para este sistema empotrado se muestran en la Figura 74 para el modo “Debug=0” y en la Figura 75 para el modo “Debug=1”.



```
COM4 - PuTTY
AEGIS HARDWARE IMPLEMENTATION (Polling)
--- Entering main() ---
FIFO 0 configured ... Initialization OK
FIFO 1 configured ... Initialization OK
Run-time = 209.85 usec
Final Test:
Comparing data ...
Successfully ran AEGIS Polling
--- Exiting main() ---
```

Figura 74 Ejecución de la aplicación de verificación mediante *polling* en modo Debug 0 en el sistema que usa una FIFO conectada al bus AXI4.



```
COM4 - PuTTY

--- Entering main() ---

FIFO 0 configured ... Initialization OK
FIFO 1 configured ... Initialization OK

Transmisson of Public data:
Public data Transmitted

Transmisson of Secret data:
Secret data Transmitted

Reception of AEGIS data:
AEGIS data Received

Run-time = 13973.94 usec

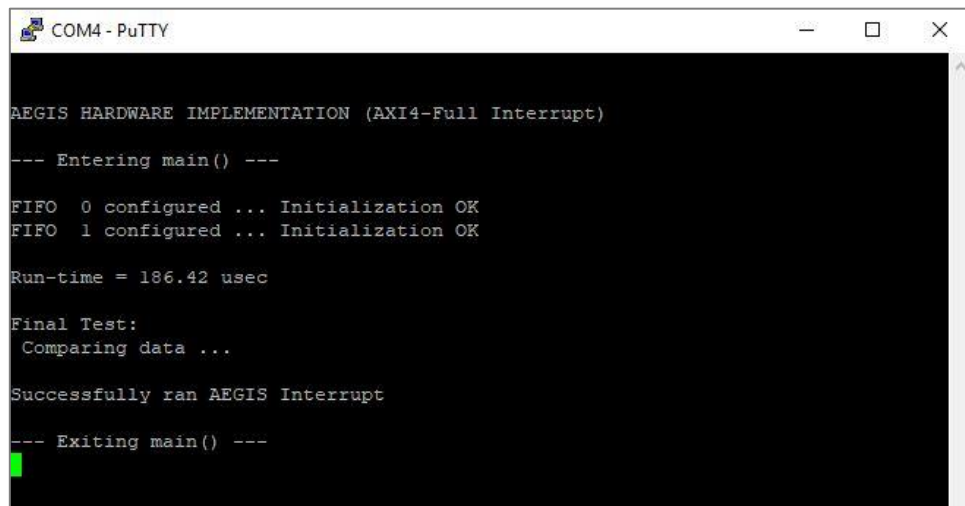
Final Test:
Comparing data ...

Successfully ran AEGIS Polling

--- Exiting main() ---
```

Figura 75. Ejecución de la aplicación de verificación mediante *polling* en modo Debug 1 en el sistema que usa una FIFO conectada al bus AXI4.

En cuanto a las aplicaciones de verificación basadas en interrupciones, igualmente se han ejecutado usando distintos modos de depuración. El modo de depuración “Debug=0”, es de nuevo el elegido para evaluar el tiempo de cifrado de esta configuración y compararlo con el resto de las realizadas. También se puede nuevamente ejecutar en el modo de depuración “Debug=1” para verificar que el proceso se ejecuta correctamente. En la Figura 76, se muestra la verificación en modo “Debug=0” mientras que en la Figura 77 se muestra la verificación en modo “Debug=1”.



```
COM4 - PuTTY

AEGIS HARDWARE IMPLEMENTATION (AXI4-Full Interrupt)

--- Entering main() ---

FIFO 0 configured ... Initialization OK
FIFO 1 configured ... Initialization OK

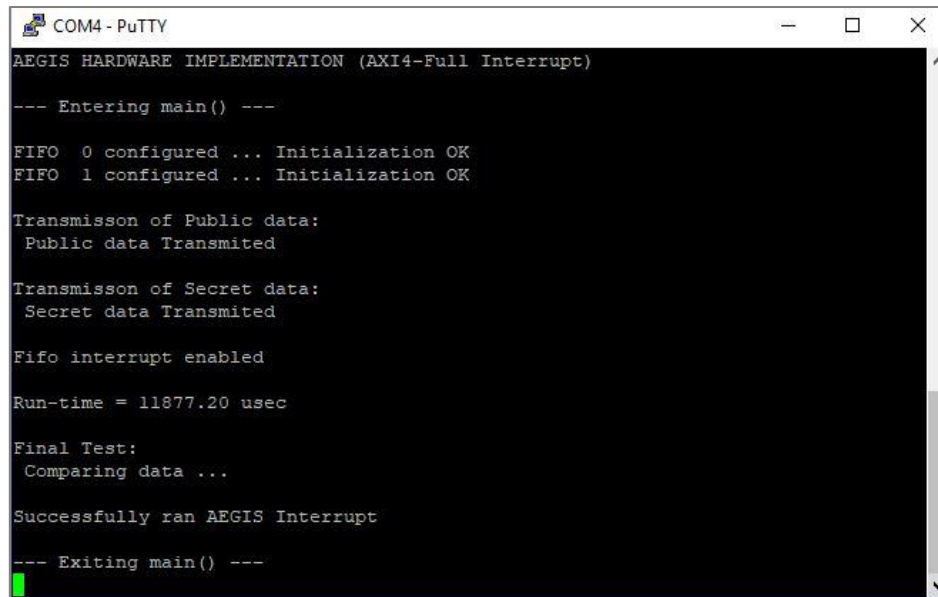
Run-time = 186.42 usec

Final Test:
Comparing data ...

Successfully ran AEGIS Interrupt

--- Exiting main() ---
```

Figura 76. Ejecución de la aplicación de verificación por interrupciones en modo Debug 0 en el sistema que usa una FIFO conectada al bus AXI4.



```
COM4 - PuTTY
AEGIS HARDWARE IMPLEMENTATION (AXI4-Full Interrupt)

--- Entering main() ---

FIFO 0 configured ... Initialization OK
FIFO 1 configured ... Initialization OK

Transmisson of Public data:
Public data Transmitted

Transmisson of Secret data:
Secret data Transmitted

Fifo interrupt enabled

Run-time = 11877.20 usec

Final Test:
Comparing data ...

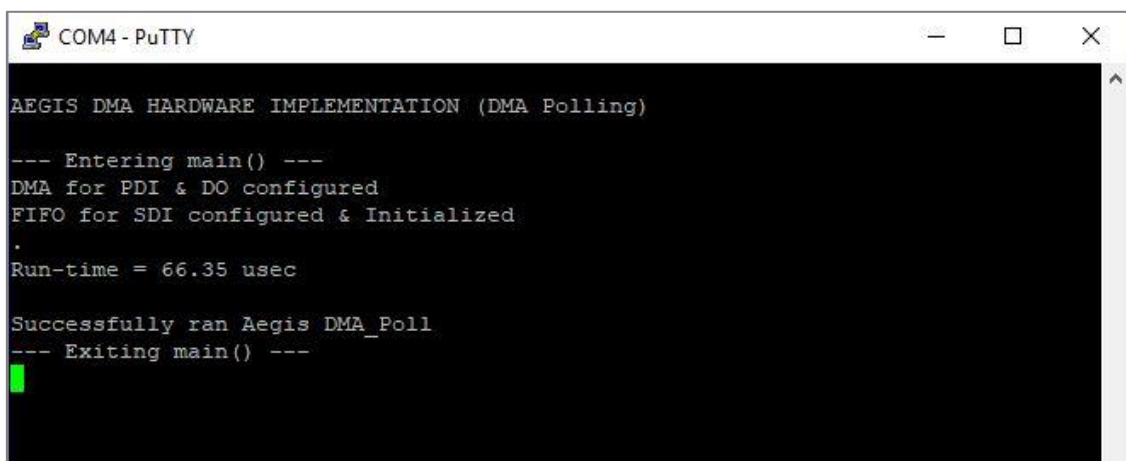
Successfully ran AEGIS Interrupt

--- Exiting main() ---
```

Figura 77. Ejecución de la aplicación de verificación por interrupciones en modo Debug 1 en el sistema que usa una FIFO conectada al bus AXI4.

5.3.3. Sistema empotrado conectado mediante DMA.

Utilizando la misma metodología, se obtienen en primer lugar los resultados obtenidos con la aplicación de verificación mediante *polling* para los modos de depuración “Debug = 0” y “Debug = 1”. En la Figura 78 se muestra la salida de la aplicación proporcionando el tiempo invertido por el sistema para llevar a cabo la operación de cifrado, mientras que en la Figura 79 se muestra nuevamente la ejecución de la aplicación en el modo de depuración “Debug=1” para verificar que el proceso se ejecuta correctamente.



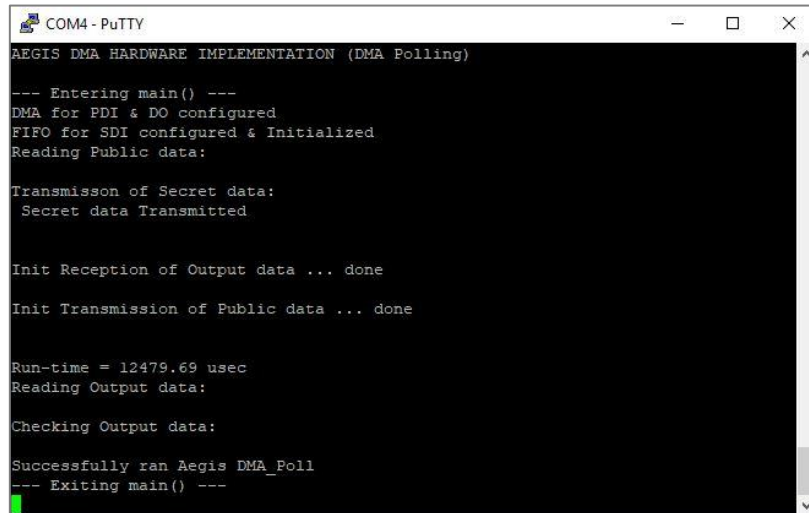
```
COM4 - PuTTY
AEGIS DMA HARDWARE IMPLEMENTATION (DMA Polling)

--- Entering main() ---
DMA for PDI & DO configured
FIFO for SDI configured & Initialized
.
Run-time = 66.35 usec

Successfully ran Aegis DMA_Poll

--- Exiting main() ---
```

Figura 78. Ejecución de la aplicación de verificación mediante *polling* en modo Debug 0 en el sistema que emplea el módulo DMA.



```
COM4 - PuTTY
AEGIS DMA HARDWARE IMPLEMENTATION (DMA Polling)

--- Entering main() ---
DMA for PDI & DO configured
FIFO for SDI configured & Initialized
Reading Public data:

Transmission of Secret data:
Secret data Transmitted

Init Reception of Output data ... done
Init Transmission of Public data ... done

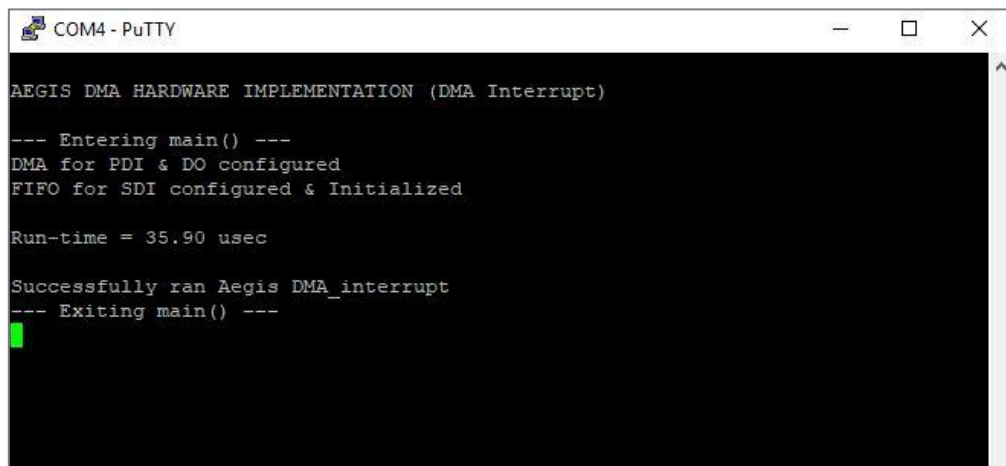
Run-time = 12479.69 usec
Reading Output data:

Checking Output data:

Successfully ran Aegis DMA_Poll
--- Exiting main() ---
```

Figura 79. Ejecución de la aplicación de verificación mediante *polling* en modo Debug 1 en el sistema que emplea el módulo DMA.

Los resultados obtenidos con la aplicación de verificación basada en interrupciones siguiendo la misma metodología anterior de los modos de depuración se muestran en la Figura 80 y Figura 81 para los modos de depuración “Debug=0” y “Debug=1” respectivamente.



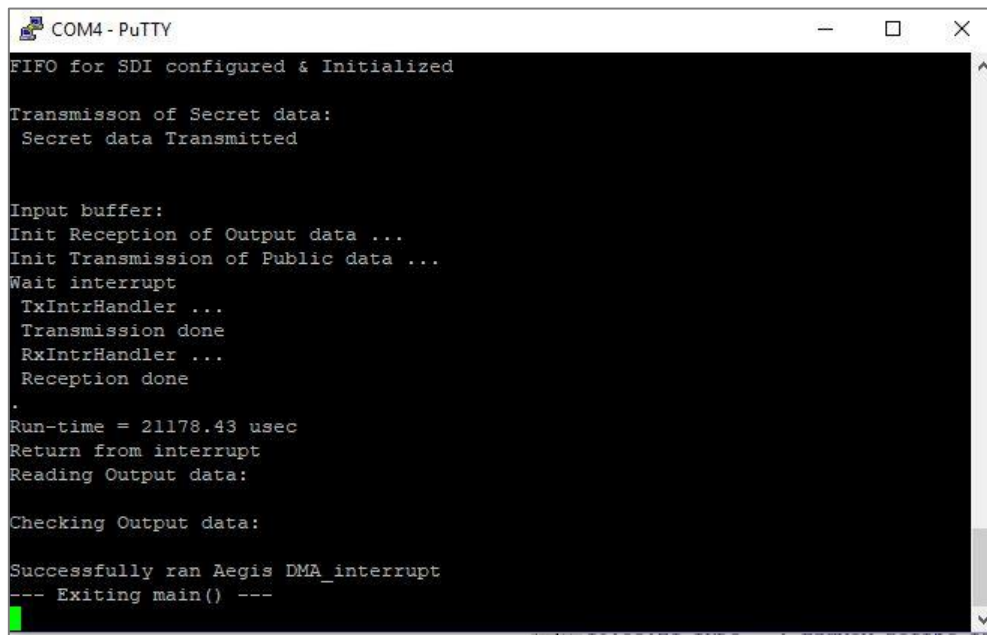
```
COM4 - PuTTY
AEGIS DMA HARDWARE IMPLEMENTATION (DMA Interrupt)

--- Entering main() ---
DMA for PDI & DO configured
FIFO for SDI configured & Initialized

Run-time = 35.90 usec

Successfully ran Aegis DMA_interrupt
--- Exiting main() ---
```

Figura 80. Ejecución de la aplicación de verificación por interrupciones en modo Debug 0 en el sistema que emplea el módulo DMA.



```

COM4 - PuTTY
FIFO for SDI configured & Initialized

Transmission of Secret data:
Secret data Transmitted

Input buffer:
Init Reception of Output data ...
Init Transmission of Public data ...
Wait interrupt
TxIntrHandler ...
Transmission done
RxIntrHandler ...
Reception done
.
Run-time = 21178.43 usec
Return from interrupt
Reading Output data:
Checking Output data:

Successfully ran Aegis DMA_interrupt
--- Exiting main() ---

```

Figura 81. Ejecución de la aplicación de verificación por interrupciones en modo Debug 1 en el sistema que emplea el módulo DMA.

5.3.4. Comparación de los distintos esquemas de conexión.

La Tabla 3 recopila los resultados de utilización de los recursos de la FPGA de cada uno de los sistemas empotrados para hacer una comparativa del tamaño de los mismos.

COMPONENTE	AXI4-Lite	AXI4	AXI DMA
Slice LUTs	13287 / 75.48%	13643 / 77.52%	15048 / 85.50%
Slice Registers	9344 / 26.55%	9483 / 26.94%	11807 / 33.54%
F7 Muxes	2493 / 28.33%	2493 / 28.33%	2497 / 28.17%
F8 Muxes	1024 / 23.27%	1024 / 23.27%	1024 / 23.27%
Slice	4155 / 94%	4236 / 96%	4308 / 97%
LUT as Logic	11926 / 67.76%	12276 / 69.75%	13343 / 75.81%
LUT as Memory	1358 / 22.63%	1367 / 22.78%	1705 / 28.48%
LUT Flip Flop Pairs	4265 / 24.23%	4415 / 25.09%	5371 / 30.52%
Block RAM Tile	7.5 / 12.50%	7.5 / 12.50%	6 / 10%

Tabla 3. Comparación del consumo de recursos de la FPGA para cada uno de los sistemas empotrados.

Con los resultados que se muestran, la mejor opción en cuanto a tamaño parece ser el sistema empotrado con buses AXI4-Lite seguido muy de cerca por el sistema empotrado con buses AXI4. El consumo de recursos del sistema empotrado con buses AXI DMA es claramente superior al de los anteriores, sin embargo, es necesario comparar los tiempos que tardan en realizar el software de verificación con distintas estrategias para llegar a una conclusión final respecto a qué sistema empotrado es más adecuado para una situación determinada.

Los datos proporcionados por las aplicaciones de verificación desarrolladas para los sistemas empotrados propuestos en este trabajo permiten llevar a cabo una comparación temporal de las diferentes propuestas. En la

Tabla 4 se muestran (en microsegundos) los resultados temporales obtenidos.

	AXI4-Lite	AXI4	AXI DMA
Software sin caché de datos ni de instrucciones	5746.84 (~2873)		
Software sin caché de datos	4897.72 (~2448,9)		
Software con las caché	361.7 (~180.85)		
Polling	210.06	209.85	66.35
Interrupciones	211.86	186.42	35.9

Tabla 4. Comparación de tiempos de ejecución para los distintos sistemas empotrados y diferentes opciones de verificación.

Las aplicaciones de verificación de la implementación software del cifrador incluyen una operación de cifrado y descifrado. Entre paréntesis aparece el tiempo estimado para cada una de las operaciones suponiendo que ambas tienen la misma duración.

A la vista de los resultados que se han obtenido se pueden obtener las siguientes conclusiones:

- Para implementaciones software del cifrador sobre el procesador ARM, la influencia de la cachés de instrucciones y, en mayor medida, de la caché de datos es fundamental, permitiendo una reducción entre el 90% y el 95% en el tiempo de ejecución del test de verificación.
- En los sistemas empotrados que emplean FIFOs, la diferencia entre conectar la FIFO que transfiere los datos públicos mediante un bus AXI con funcionalidad total (AXI4) o hacerlo con uno de funcionalidad reducida (AXI4-Lite) se traduce en un incremento de alrededor del 2% en el consumo de recursos de la FPGA.

- El sobrecoste requerido para aprovechar la funcionalidad del “modo de transmisión por ráfagas” que proporciona el bus AXI4 apenas tiene incidencia en cuanto a los tiempos de ejecución de ambas alternativas cuando el programa de verificación utiliza *polling*. Esto es consecuencia de los tiempos invertidos en el ciclo de consulta que se introducen en el programa de verificación para asegurar que los datos de salida no son transferidos hasta que no ha terminado la operación del cifrador. De hecho, con los tamaños de los vectores de test utilizados no se consigue una verdadera aceleración mediante la implementación hardware del cifrador, ya que los tiempos obtenidos son similares a los conseguidos mediante software cuando se habilitan las cachés de instrucciones y datos del procesador ARM.
- El uso de un mecanismo de interrupciones en los programas de verificación sí pone de manifiesto una mejora significativa (12% de reducción) en el tiempo de ejecución del sistema que utiliza AXI4 frente al que emplea AXI4-Lite.
- La utilización de esquemas de acceso a memoria mediante el bloque AXI DMA permite poner de manifiesto la aceleración hardware conseguida mediante la implementación del cifrador como un periférico del procesador ARM. Los tiempos de ejecución se reducen en un 68% y un 83% frente a los del sistema con AXI4-Lite cuando se emplean técnicas de *polling* y de interrupciones, respectivamente. Este incremento en prestaciones va acompañado, sin embargo, de un aumento en torno al 8% o al 10% en el consumo de recursos de la FPGA con respecto a los empleados en los sistemas con FIFOs conectadas mediante el bus AXI4 o AXI4-Lite, respectivamente.

Atendiendo a los criterios de la competición CAESAR, donde el caso de uso 1 (“Lightweight”) premia un tamaño menor del cifrador, se elegiría el sistema empotrado con buses AXI4-Lite, mientras que para el caso de uso 2 (“High-Performance”) se seleccionaría el sistema empotrado que usa el bloque AXI DMA.

5.4. Implementación de otros cifradores.

Con el objetivo de mostrar que el procedimiento desarrollado en el capítulo 4 es común para todos los cifradores autenticados que cumplen los requisitos de la competición CAESAR, en esta sección se va a utilizar dicho procedimiento con otro de los finalistas de la competición, el cifrador autenticado ACORN. Para ello solo será necesario llevar a cabo los ajustes que se describen en los siguientes apartados.

.

5.4.1. Implementación del cifrador ACORN.

Este proceso es similar al descrito para el diseño del bloque “AEGIS-32”. A partir del código VHDL que describe la operación del cifrador autenticado ACORN que proporciona ATHENA [13], se llevan a cabo los procesos de síntesis, implementación y verificación del sistema siguiendo el flujo de diseño RTL facilitado por las herramientas del entorno Vivado. Cabe destacar que la descripción del cifrador ACORN utiliza la versión “lightweight” de la interfaz hardware proporcionada por ATHENA en el paquete “caesar_hw_devpkg-2.0 – src_rtl_low”. Las fuentes utilizadas en este caso son:

- Fuentes de diseño:

- Acorn_Control.vhd,
 - Acorn_Datath.vhd,
 - ACORN_pkg.vhd,
 - ACORN_StateUpdate.vhd,
 - AEAD.vhd,
 - AEAD_pkg.vhd,
 - CAESAR_LWAPI_pkg.vhd,
 - CipherCore.vhd,
 - design_pkg.vhd,
 - fwft_fifo.vhd,
 - GeneralComponents_pkg.vhd,
 - PostProcessor.vhd,
 - PreProcessor.vhd,
 - Reg.vhd,
 - Register_s.vhd,
 - RegLd.vhd,
 - RegN.vgd,
 - RegNL.vhd,
 - shift_reg.vhd,
 - SPDRam.vhd.vhd,
 - std_logic_1164_additions.vhd,
 - StepDownCountLD.vhd,
 - UpCountRst.vhd.

- Fuentes de simulación:

- AEAD_TB.vhd,
 - std_logic_1164_additions.vhd,
 - pdi.txt,
 - sdi.txt
 - do.txt.

La verificación funcional del diseño, mostrada en la Figura 82, permite comprobar el correcto funcionamiento del cifrador empleando los vectores de test generados por la utilidad AETVgen y almacenados en los ficheros “pdi.txt”, “sdi.txt”, y “do.txt”. En la figura se puede observar, asimismo, que el tamaño de los buses de entrada y salida del cifrador es ahora de 8 bits.

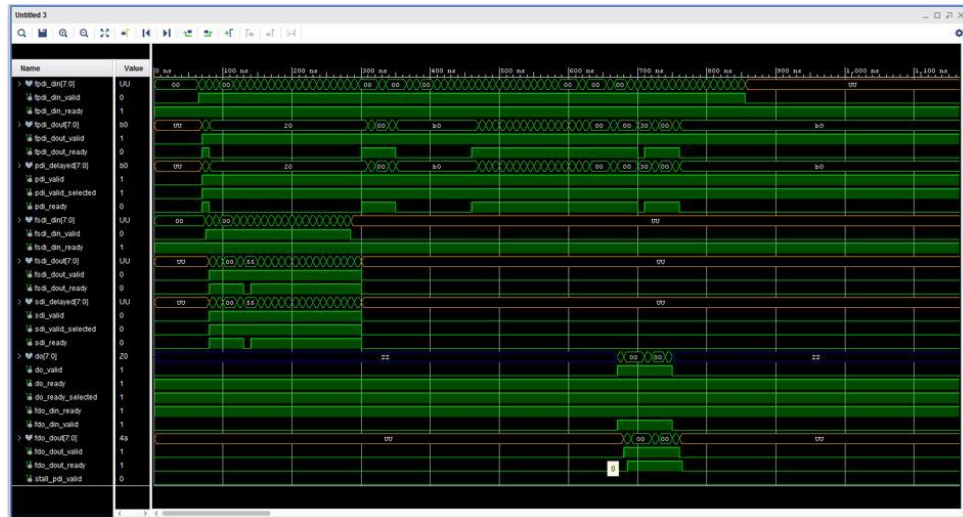


Figura 82. Resultado de simulación del comportamiento del cifrador autenticado ACORN.

El consumo de recursos de los diferentes elementos que componen el cifrador tras realizar la síntesis e implementación el diseño, se detalla en la Figura 83.

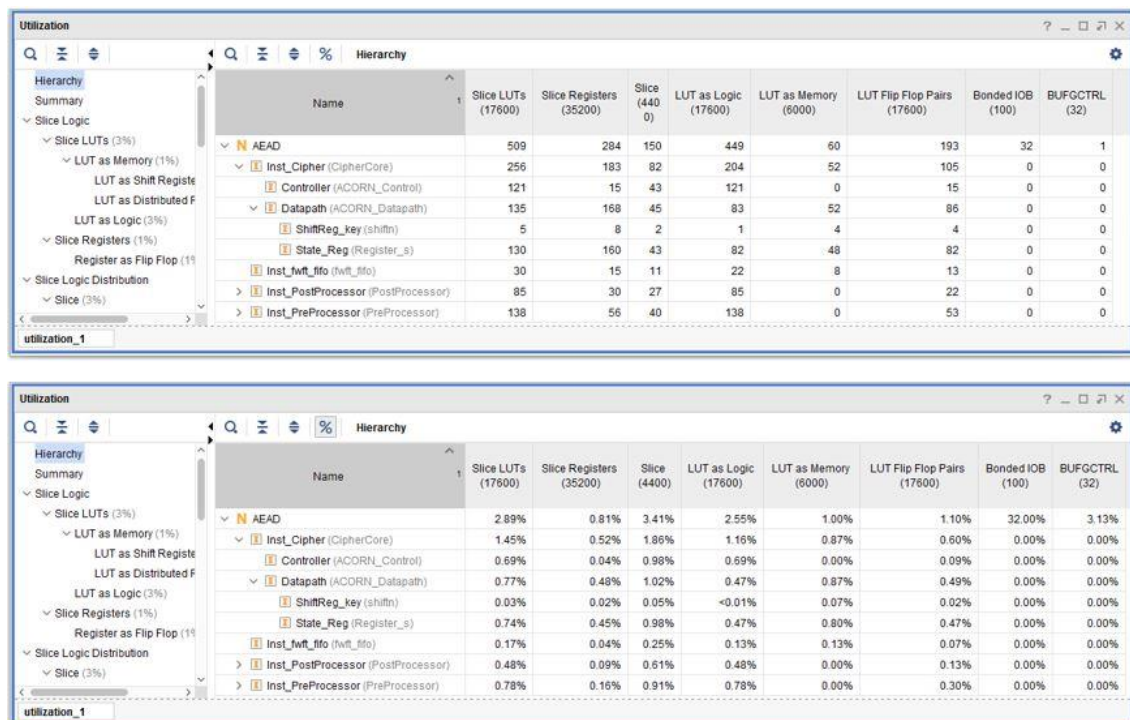


Figura 83. Utilización de recursos de la FPGA para el cifrador autenticado ACORN.

5.4.2. Generación del módulo IP del cifrador.

La creación del módulo IP del ACORN es también muy similar a la del AEGIS. De nuevo se tienen que incluir en el diseño las FIFOs de transferencia de datos y modificar la anchura de los buses de entrada y salida en los ficheros “AEAD_IP” y “AEAD_IP_TB” para reducirlos a un tamaño de 8 bits (en AEGIS fue necesario reducirlo a 32 bits). Dicha modificación se lleva a cabo definiendo los parámetros “G_PWIDTH” = “G_SWIDTH” = 8.

Además de los ficheros VHDL indicados en el apartado anterior, las fuentes usadas para la implementación de este módulo son los siguientes:

- Fuentes de diseño: AEAD_IP.vhd.
- Fuentes de simulación: AEAD_IP_TB.vhd.

La Figura 84 muestra la jerarquía del proyecto Vivado, tanto para síntesis como para implementación, una vez añadidas todas las fuentes. El esquemático del módulo IP creado tras la etapa de elaboración del diseño se muestra en la Figura 85. La simulación funcional del diseño empleando los vectores de test usados previamente muestra los resultados que aparecen en la Figura 86

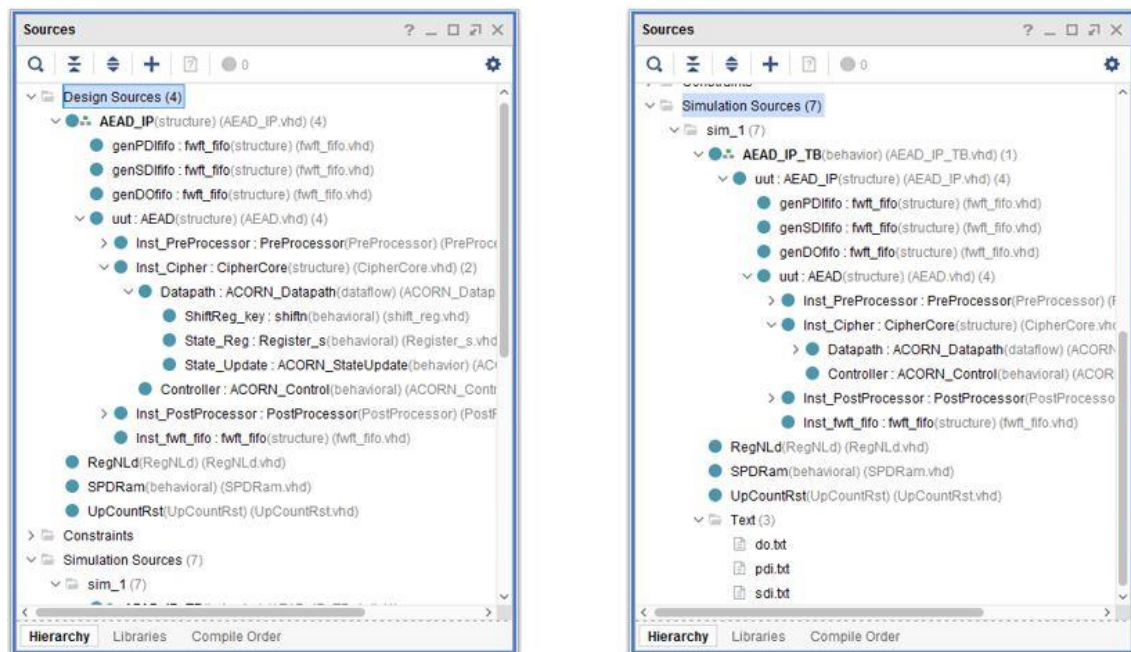


Figura 84. Fuentes del módulo IP del cifrador autenticado ACORN.

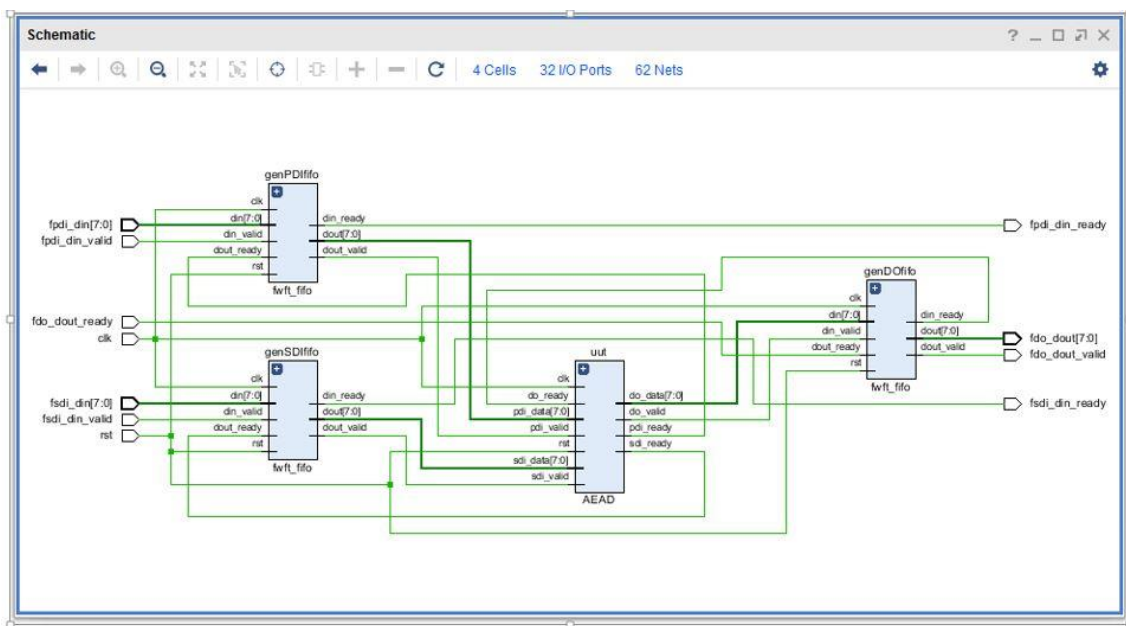


Figura 85. Esquemático del módulo IP del cifrador ACORN.

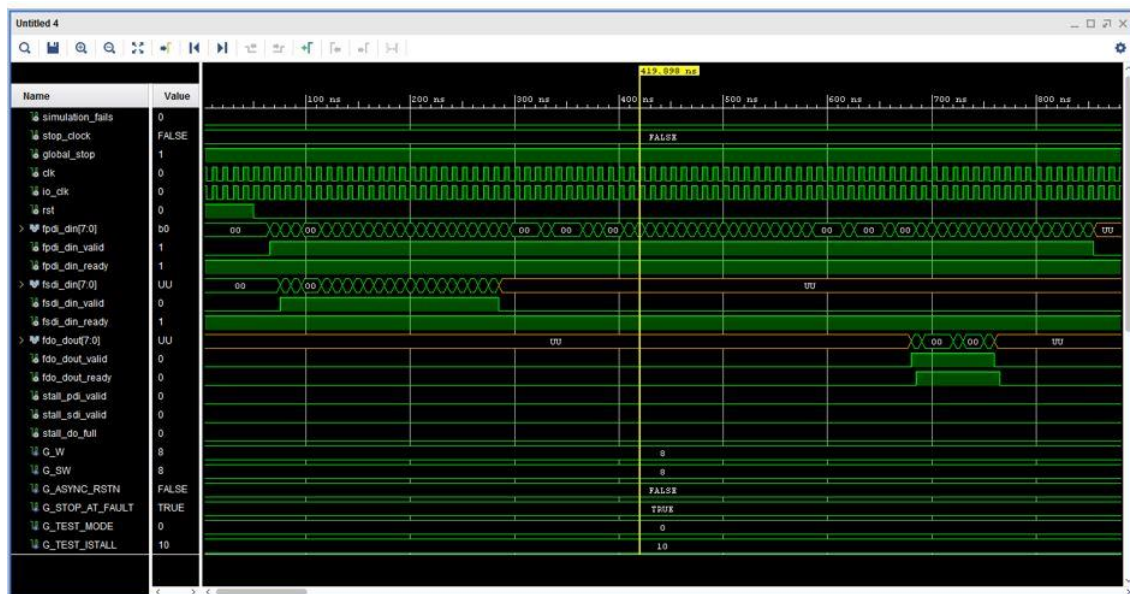


Figura 86. Simulación del comportamiento del módulo IP del cifrador ACORN.

La definición de las interfaces AXI4-Stream y la generación del módulo IP como periférico se lleva a cabo de forma idéntica a la empleada en el caso del AEGIS. Por último, se comprueba la inclusión del módulo en el catálogo de IPs de Vivado y se verifica su comportamiento por simulación, instanciándolo en un diagrama de bloques de IP Integrator, como se muestra en la Figura 87

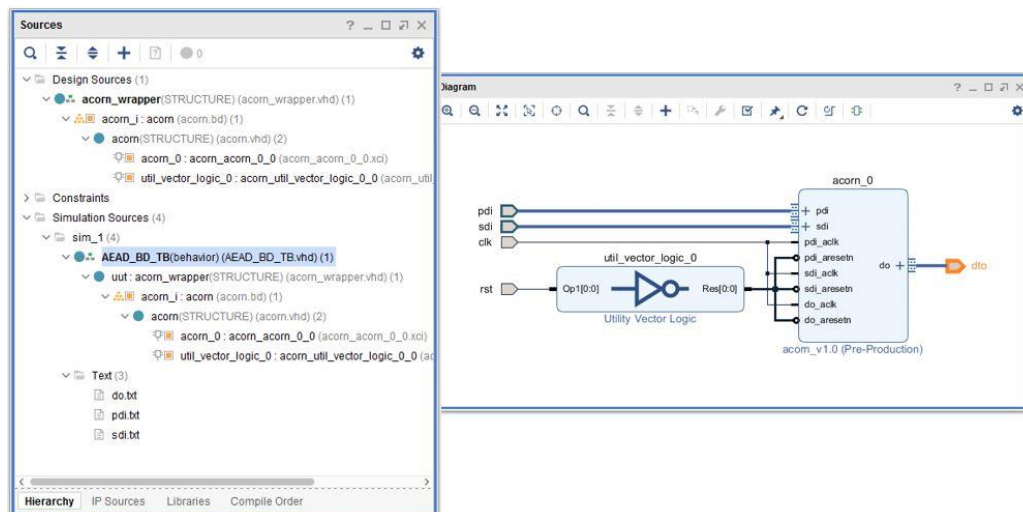


Figura 87. Fuentes y diagrama de bloques del periférico AXI del módulo IP del cifrador ACORN.

Para la verificación funcional del diseño, se crea el “HDL wrapper” y se realiza la simulación, obteniendo los resultados que se muestran en la Figura 88.

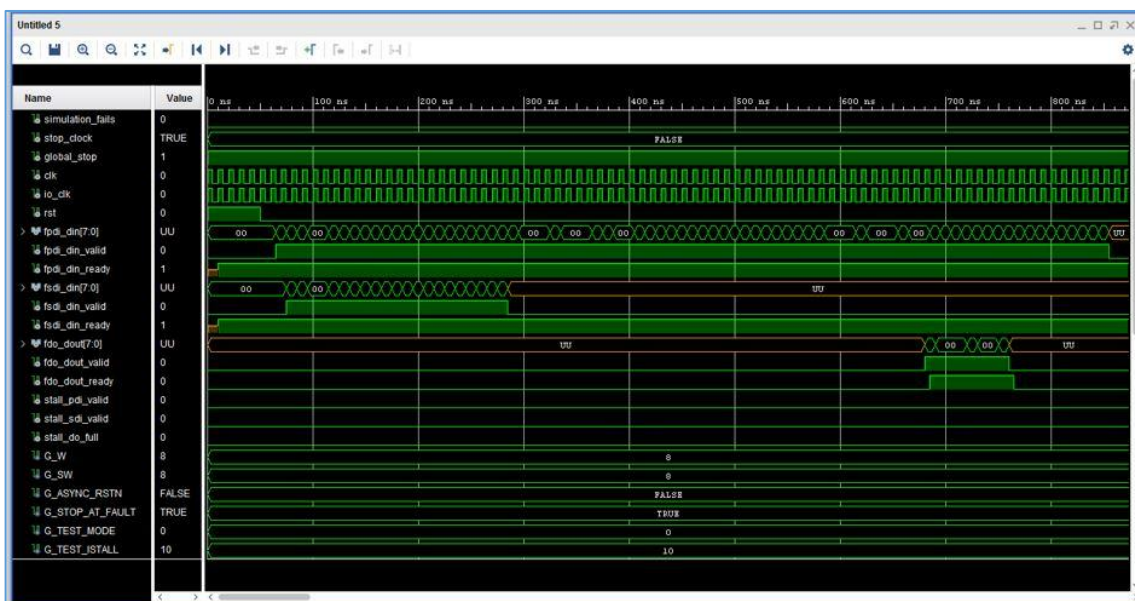


Figura 88. Resultado de la simulación del comportamiento del periférico del módulo IP de ACORN.

5.4.3. Inclusión del cifrador ACORN en un sistema empuetrado.

El módulo IP del cifrador ACORN está ahora listo para ser incorporado como periférico del procesador ARM en un sistema empuetrado. En principio podrían utilizarse los tres esquemas de interconexión empleados en el caso del AEGIS. No obstante, al

tratarse en este caso de un cifrador encuadrado en la categoría “lightweight”, no tiene demasiado sentido utilizar los esquemas basados en el bus AXI4 o el módulo AXI-DMA, por lo que nos limitaremos a describir su incorporación en un sistema empotrado que utilice AXI-Stream FIFOs con buses AXI4-Lite.

Para la creación del sistema empujado con este cifrador autenticado, hay que tener en cuenta que los módulos “AXI-Stream FIFO” utilizan buses de 32 bits de ancho mientras que los buses del ACORN tienen una anchura de 8 bits. Es necesario por tanto usar algún tipo de elemento que optimice la transmisión y recepción de datos entre las FIFOs y el cifrador. Este periférico se denomina “AXI4-Stream Data Width Converter” [26] y, además de adaptar el tamaño de los buses de las FIFOs y el cifrador, convierte los paquetes de 32 bits enviados al cifrador en cuatro paquetes de 8 bits (y al contrario con los datos de salida del cifrador) manteniendo el protocolo de gestión de bus.

El diagrama de bloques del sistema empotrado se muestra en la Figura 89. Como en el caso del AEGIS, se utiliza un “AXI-Stream FIFO” para el intercambio de datos públicos entre el procesador y el cifrador y otro “AXI-Stream FIFO” para el envío de los datos secretos. Ambos están configurados para que se conecten al procesador mediante un bus AXI4_Lite. En este caso se utilizan también tres bloques “AXI4-Stream Data Width Converter” para adaptar los dos canales de transmisión y el de recepción a las características del módulo IP que implementa el cifrador ACORN.

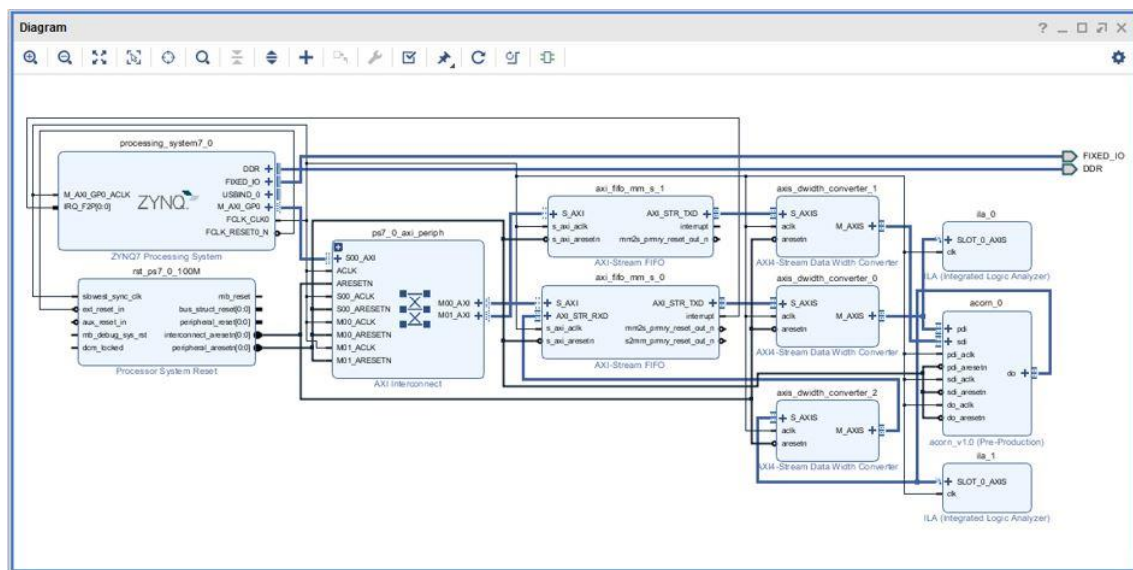


Figura 89. Sistema empotrado con el módulo IP del cifrador ACORN.

El reporte de utilización de recursos de la FPGA para este sistema empujado se muestra en la Figura 90. Como puede observarse en el diagrama de la Figura 89 y la tabla de la Figura 90, el sistema incorpora dos bloques ILA (“Integrated Logic Analyzer”) que han sido utilizados en la etapa de depuración de la plataforma hardware con objeto de verificar el correcto formato de transmisión de los datos entre las FIFOs y el cifrador a través de los bloques “AXI4-Stream Data Width Converter”.

Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	Slice (1330 0)	LUT as Logic (53200)	LUT as Memory (17400)	LUT Flip Flop Pairs (53200)	Block RAM Tile (140)	Bonded IOPADs (130)	BUFG (3)
acorn_wrapper	4709	5835	114	2060	4085	624	2351	4	130	
acorn_j (acorn)	4225	5096	114	1843	3625	600	2022	4	0	
acorn_0 (acorn_acorn_0_0)	900	365	96	272	648	252	255	0	0	
axi_tfm_mm_s_0 (acorn_axi_tfm_mm_s_0_0)	602	634	0	256	541	61	252	2	0	
axi_tfm_mm_s_1 (acorn_axi_tfm_mm_s_1_0)	276	362	0	124	275	1	130	1	0	
axis_dwidth_converter_0 (acorn_axis_dwidth_converter_0_0)	31	50	0	15	31	0	16	0	0	
axis_dwidth_converter_1 (acorn_axis_dwidth_converter_1_0)	29	48	0	17	29	0	14	0	0	
axis_dwidth_converter_2 (acorn_axis_dwidth_converter_0_1)	18	49	0	19	18	0	9	0	0	
ila_0 (acorn_ila_0_1)	926	1452	9	483	814	112	528	0.5	0	
ila_1 (acorn_ila_1_1)	926	1452	9	494	814	112	527	0.5	0	
processing_system7_0 (acorn_processing_system7_0_0)	0	0	0	0	0	0	0	0	0	
ps7_0_axi_periph (acorn_ps7_0_axi_periph_0)	490	647	0	210	429	61	266	0	0	
rst_ps7_0_100M (acorn_rst_ps7_0_100M_0)	18	37	0	11	17	1	16	0	0	
dbg_hub (dbg_hub)	484	739	0	242	460	24	327	0	0	
inst (xsdbm_v3_0_0_xsdbm)	484	739	0	242	460	24	327	0	0	

Figura 90. Consumo de recursos de la FPGA para el sistema empotrado que utiliza el cifrador ACORN.

Por otra parte, para llevar a cabo las distintas aplicaciones software de verificación ha sido necesario adaptar el programa de generación del fichero de cabecera, “data.h”, para que envíe correctamente los datos públicos y privados de forma ordenada en paquetes de 8 en vez de 32 bits.

En cuanto al comportamiento temporal de este cifrador, la Figura 91 muestra los tiempos invertidos por el código C de referencia ejecutado sobre el procesador ARM para tres configuraciones distintas: a) con las cachés de instrucciones y datos habilitadas; b) con la caché de datos desahilitada; y c) con ambas cachés deshabilitadas. En todos los casos se ha utilizado como entrada a cifrar el mismo mensaje utilizado en el caso del cifrador AEGIS. Como puede observarse, los tiempos de ejecución son considerablemente superiores a los obtenidos con el otro cifrador.

```

COM47 - PuTTY
ACORN SOFTWARE IMPLEMENTATION

--- msglen = 750, adlen = 100 -->
Run-time = 133495.79 usec

End Test

ACORN SOFTWARE IMPLEMENTATION

--- msglen = 750, adlen = 100 -->
Run-time = 2659808.50 usec

End Test

ACORN SOFTWARE IMPLEMENTATION

--- msglen = 750, adlen = 100 -->
Run-time = 2765244.38 usec

End Test

```

Figura 91. Tiempos de ejecución en Software del cifrador ACORN con distintas configuraciones de memoria caché.

Finalmente, la Figura 92 ilustra el uso de las facilidades de depuración hardware proporcionadas por Vivado a través del “Hardware Manager” para verificar la funcionalidad del diseño. En concreto, las formas de onda que aparecen en la figura, correspondiente a los últimos ciclos de transmisión de los datos públicos de entrada al cifrador, puede apreciarse cómo el tamaño de estos datos es de 8 bits y que se ha añadido un dato con valor “00” coincidente con la señal TLAST para hacer que el número de Bytes transmitidos sea múltiplo de 4 (32 bits).

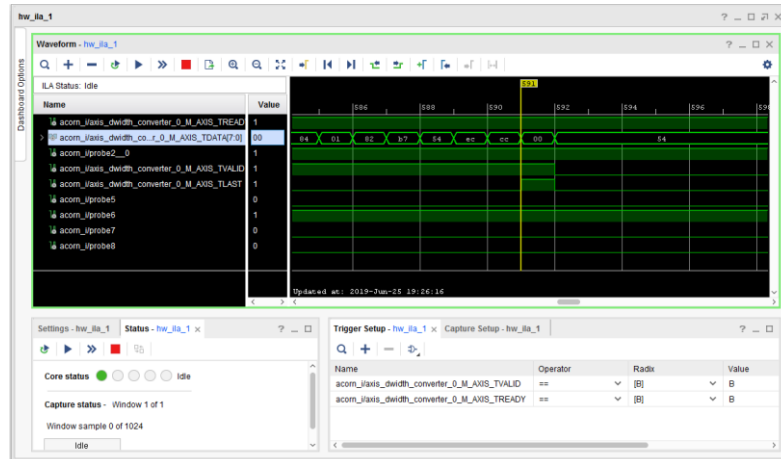


Figura 92. Verificación del funcionamiento del diseño a través de la herramienta de depuración proporcionada por el "Hardware Manager".

5.4.4. Comparación de los recursos utilizados entre AEGIS y ACORN.

La Tabla 5 muestra una comparación de los recursos consumidos por los sistemas empujados que incorporan los dos cifradores analizados en este trabajo. Aunque las dos implementaciones corresponden a configuraciones que emplean el mismo esquema de conexionado mediante buses AXI4-Lite, los datos muestran una clara diferencia en los recursos necesarios para implementar el cifrador AEGIS, encuadrado en la categoría “High-Performance” de la competición AEGIS, frente al cifrador ACORN, representante de la categoría “lightweight” de dicha competición.

	AEGIS	ACORN
Slice LUTs	13287	4709
Slice Registers	9344	5835
F7 Muxes	2493	114
F8 Muxes	1024	0
Slice	4155	2060
LUT as Logic	11926	4085
LUT as Memory	1358	624
LUT Flip Flop Pairs	4265	2351
Block RAM Tile	7.5	0

Tabla 5. Comparación de los recursos utilizados por los cifradores AEGIS y ACORN.

Como se puede observar, el consumo de los recursos de la placa es muy inferior en el caso del cifrador autenticado ACORN, tal como era de esperar teniendo en cuenta sus características.

Conclusiones y trabajo futuro

- El trabajo descrito en esta memoria ha permitido poner en pie un flujo de diseño que facilita el desarrollo y evaluación de distintas soluciones para la implementación hardware de cifradores autenticados y su incorporación como periféricos aceleradores en sistemas empotrados para distintos casos de aplicación.
- Como punto de partida se seleccionaron dos cifradores finalistas en las distintas categorías establecidas en la competición CAESAR. El cifrador AEGIS, perteneciente a la categoría “High-Performance” y utilizado a lo largo de la memoria para describir en detalle cada una de las fases de desarrollo, y el cifrador ACORN, englobado en la categoría “Lightweight”, que nos permitirá comprobar el nivel de generalización de la metodología propuesta en el trabajo.
- De especial ayuda para la realización del trabajo ha resultado el material proporcionado por el proyecto ATHENa, disponible a través de su web, que incluye los códigos C de referencia de las distintas propuestas, código VHDL para facilitar su implementación hardware manteniendo una interfaz de comunicación común, y una serie de utilidades para unificar el proceso de generación de vectores de test.
- Como plataforma hardware para la implementación de los cifradores y su inclusión en sistemas empotrados se ha utilizado una placa de desarrollo Zybo que incorpora un dispositivo Zynq-7000 de Xilinx. Este dispositivo combina lógica programable de las FPGAs de la Serie-7 de este fabricante con un sistema de procesamiento de doble núcleo ARM Cortex-A9, lo que facilita la comparación de implementaciones software y hardware de los cifradores.
- Todos los diseños se han llevado a cabo con las herramientas del entorno Vivado, utilizando las distintas facilidades accesibles a través de su entorno de desarrollo unificado (IDE) para realizar las distintas etapas de síntesis y verificación necesarias para llevar a cabo la implementación del cifrador, su conversión en módulo IP y la integración en un sistema empotrado.
- Para adaptar el código proporcionado por los autores a las características específicas de la plataforma utilizada en el trabajo ha sido necesario ajustar algunos de los parámetros de diseño del cifrador considerado.
- También ha sido necesario modificar en consonancia los programas de generación de vectores de test proporcionados por ATHENa, así como proveer al sistema empotrado de un mecanismo para que el procesador pueda acceder a dichos vectores de test sin tener que acceder a un sistema de ficheros (inexistente en una implementación “bare-metal”) y desarrollar un programa para facilitar su uso.
- Las implementaciones de los cifradores como periféricos del procesador ARM utilizan buses AXI4-Stream, compatibles con las líneas de datos y señales de protocolos definidos para los cifradores participantes en CAESAR. Esta característica asegura que las soluciones aportadas en este trabajo son directamente aplicables para la implementación de cualquiera de los cifradores participantes en la competición.

- Para llevar a cabo la interconexión de los módulos IP de los cifradores con el resto de los componentes del sistema empotrado se han explorado dos soluciones: una basada en el uso del módulo “AXI-Stream FIFO” y otra en el del bloque “AXI DMA”. Las FIFOs empleadas en el primer caso pueden, a su vez, conectarse a la infraestructura de comunicaciones del sistema empotrado utilizando buses AXI4 o AXI4-Lite.
- Los resultados de ocupación de recursos de la FPGA de las diferentes alternativas, en combinación con los tiempos de ejecución de las diferentes opciones obtenidos por aplicaciones de verificación que usan técnicas de polling o de interrupciones, proporcionan las medidas necesarias para establecer compromisos coste/rendimiento adecuados para diferentes tipos de aplicaciones (como se comenta en el apartado 5.3.4)
- Por último, conviene resaltar que, aparte de cubrir los objetivos propuestos, este trabajo puede ser considerado como punto de partida para el desarrollo de otro conjunto de tareas que permitan avanzar en esta línea de trabajo. Entre los posibles trabajos futuros cabe destacar:
 - La automatización del flujo de diseño mediante ficheros de comandos Tcl para acortar los tiempos empleados en la verificación y caracterización de cada propuesta.
 - La realización de un estudio comparativo que incluya la optimización de los parámetros de los cifradores y abarque un mayor número de cifradores presentados a la competición CAESAR.
 - La extensión de la metodología, procedimientos y esquemas de conexión a los cifradores propuestos en otras competiciones actualmente abiertas (como la “Post-Quantum Cryptography”, cuyos participantes en segunda ronda fueron anunciados en enero de 2019).

Bibliografía

- [1] D. Salomon, Coding for Data and Computer Communications, Northridge, California: Springer, 2005.
- [2] C. Paar y J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, Heidelberg: Springer, 2009.
- [3] Wikipedia, «Wikipedia,» 25 Julio 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Data_Encryption_Standard. [Último acceso: 20 Febrero 2019].
- [4] US Department of Commerce, National Institute of Standards and Technology, «FIPS PUB 197,» 26 Noviembre 2001. [En línea]. Available: <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>. [Último acceso: 26 Marzo 2019].
- [5] W. Stallings, Cryptography And Network Security: Principles and Practice, Quinta ed., Londres: Pearson Education-Prentice Hall, 2011.
- [6] D. J. Bernstein, «Cryptographic Competitions,» 1 Enero 2013. [En línea]. Available: <https://competitions.cr.yp.to>. [Último acceso: 20 Febrero 2019].
- [7] M. E. F. M. M. S. Christoph Dobraunig, «competitions.cr.yp.to,» 20 Febrero 2019. [En línea]. Available: <http://competitions.cr.yp.to/round3/asconv12.pdf>. [Último acceso: 3 Junio 2019].
- [8] H. Wu, «Cryptographic competitions,» 15 Septiembre 2016. [En línea]. Available: <https://competitions.cr.yp.to/round3/acornv3.pdf>. [Último acceso: 8 Abril 2019].
- [9] H. Wu y B. Preneel, «Cryptographic competitions,» 15 Septiembre 2016. [En línea]. Available: <http://competitions.cr.yp.to/round3/aegisv11.pdf>. [Último acceso: 22 Febrero 2019].
- [10] P. R. Ted Krovetz, «competitions.cr.yp.to,» 20 Febrero 2019. [En línea]. Available: <http://competitions.cr.yp.to/round3/ocbv11.pdf>. [Último acceso: 3 Junio 2019].
- [11] I. N. T. P. Y. S. Jeremy Jean, «competitions.cr.yp.to,» 20 Febrero 2019. [En línea]. Available: <http://competitions.cr.yp.to/round3/deoxysv141.pdf>. [Último acceso: 3 Junio 2019].
- [12] A. B. N. D. A. L. B. M. a. o. Elena Andreeva, «competitions.cr.yp.to,» 20 Febrero 2019. [En línea]. Available: <http://competitions.cr.yp.to/round3/colmv1.pdf>. [Último acceso: 3 Junio 2019].

- [13] George Mason University, «ATHENA: Automated Tool for Hardware Evaluation», 2010. [En línea]. Available: <https://cryptophy.gmu.edu/athena/index.php?id=ATHENa>. [Último acceso: 7 Abril 2019].
- [14] CMP Technology, «Dr.Dobb's The world of software development», 2009. [En línea]. Available: <http://www.drdoobs.com/security/keccak-the-new-sha-3-encryption-standard/240154037>. [Último acceso: 7 Abril 2019].
- [15] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, M. U. Sharif y K. Gaj, «ATHENA. Automated Tool for Hardware Evaluation», 6 Diciembre 2015. [En línea]. Available: <https://eprint.iacr.org/2015/669>. [Último acceso: 19 Abril 2019].
- [16] DIGILENT, «Digilent Inc.», 11 Abril 2016. [En línea]. Available: https://reference.digilentinc.com/_media/zybo:zybo_rm.pdf. [Último acceso: 19 Abril 2019].
- [17] ARM, «ARM», 2 Febrero 2012. [En línea]. Available: https://static.docs.arm.com/ddi0388/i/DDI0388I_cortex_a9_r4p1_trm.pdf?_ga=2.3169719.603239806.1555579385-1664733447.1555579385. [Último acceso: 19 Abril 2019].
- [18] ARM, «gstitt.ece.ufl.edu», 28 Octubre 2011. [En línea]. Available: http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf. [Último acceso: 19 Abril 2019].
- [19] ARM, «ARM», 3 MARZO 2010. [En línea]. Available: https://static.docs.arm.com/ihl0051/a/IHL0051A_amba4_axi4_stream_v1_0_protocol_spec.pdf. [Último acceso: 15 MAYO 2019].
- [20] XILINX, «XILINX», 7 Marzo 2011. [En línea]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf. [Último acceso: 19 Abril 2019].
- [21] XILINX, «XILINX», 2019. [En línea]. Available: <https://www.xilinx.com/products/design-tools/vivado.html>. [Último acceso: 19 Abril 2019].
- [22] XILINX, «XILINX», 2019. [En línea]. Available: <https://www.xilinx.com/products/design-tools/embedded-software/sdk.html>. [Último acceso: 20 Abril 2019].
- [23] XILINX, «XILINX», 4 ABRIL 2018. [En línea]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf. [Último acceso: 27 MAYO 2019].
- [24] Wikipedia, «Wikipedia», 23 Octubre 2018. [En línea]. Available: <https://es.wikipedia.org/wiki/Polling>. [Último acceso: 11 Junio 2019].

- [25] Wikipedia, «Wikipedia,» 31 Mayo 2019. [En línea]. Available: <https://es.wikipedia.org/wiki/Interrupci%C3%B3n>. [Último acceso: 11 Junio 2019].
- [26] XILINX, «XILINX,» 5 Diciembre 2018. [En línea]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axis_infrastructure_ip_suite/v1_1/pg085-axi4stream-infrastructure.pdf. [Último acceso: 20 Junio 2019].
- [27] N. Ahmad, L. M. Wei y M. H. Jabbar, «Advanced Encryption Standard with Galois Counter Mode using Field Programmable Gate Array,» *Journal of Physics: Conference Series (Institute of Physics Publishing, vol 1019)*, 2018.