UNIVERSIDAD DE SEVILLA – CSIC
Escuela Técnica Superior de Ingeniería Informática
Departamento de Tecnología Electrónica

Instituto de Microelectrónica de Sevilla

TESIS DOCTORAL

IMPLEMENTACIONES VLSI DE CIFRADORES DE FLUJO TRIVIUM DE BAJO CONSUMO

Memoria presentada por:

JOSÉ MIGUEL MORA GUTIÉRREZ

para optar al grado de Doctor por la
Universidad de Sevilla

Directores:
MANUEL VALENCIA BARRERO Y CARLOS JESÚS JIMÉNEZ FERNÁNDEZ

Sevilla – 2017

IMPLEMENTACIONES VLSI DE CIFRADORES DE FLUJO TRIVIUM DE BAJO CONSUMO

Memoria presentada por:

JOSÉ MIGUEL MORA GUTIÉRREZ

para optar al grado de Doctor por la
Universidad de Sevilla

Directores:

MANUEL VALENCIA BARRERO CARLOS JESÚS JIMÉNEZ FERNÁNDEZ

Sevilla – 2017

Agradecimientos

A mis directores de Tesis, Manolo y Carlos por sus aportaciones a este trabajo de investigación, resaltando su implicación en el desarrollo del mismo, sus consejos y el apoyo constante recibido. La entrega y dedicación de Carlos desde el primer día hicieron posible la realización de esta Tesis. A Antonio por darme la oportunidad de iniciar este camino.

Siempre pensé que finalizar el doctorado iba a ser una labor difícil, debido a mi dedicación casi exclusiva en el Instituto de Microelectrónica de Sevilla. El camino no ha sido fácil como muchos ya conocen o han sufrido, y después de tantas horas no entregadas a mi familia y gracias a su comprensión y paciencia he podido llegar hasta aquí. El doctorado me lo imaginé como la realización de alguna de mis rutas de montaña, que tanto me gusta realizar, donde se comienza con muchas ganas, se planifica el camino minuciosamente, se busca información del trayecto, se prepara todo lo necesario para el viaje y luego cuando comienzas el itinerario ves que el camino no es tal como te lo habías imaginado, que el cansancio se va apoderando de ti conforme avanzas, que no llevas todo lo necesario, que a veces no encuentras las marcas del camino, que la duración no era la estimada, y otros tantos detalles que van haciendo efecto en tu estado anímico e incluso a veces, te hacen dudar de si llegarás al destino. Al final, se ha llegado y el camino ha merecido la pena.

No quiero olvidarme de mis compañeros del IMSE a quienes les agradezco todas sus aportaciones. A Miguel en el trabajo de mantenimiento del *Agilent 93000*, a Miguel Ángel en la realización de las placas de test y a Alberto por las tareas de soporte.

A mi familia

Resumen

En toda comunicación de datos se hace necesario proteger y garantizar de una manera adecuada la información enviada a través de cualquier tipo de red para evitar que un observador no autorizado pueda acceder o cambiar la información o suplantar identidades. Estas comunicaciones, cada vez más, se realizan entre dispositivos portables, cuyo funcionamiento con baterías y con limitados recursos de computación imponen importantes condicionantes a los mecanismos de cifrado y autenticación. La necesidad cada vez mayor de hacer segura y privada esta información transmitida entre dispositivos electrónicos portables e inalámbricos está haciendo que el uso de algoritmos criptográficos de bajo consumo de potencia y de recursos (*lightweight cryptography*) sea muy común hoy en día, y con un futuro muy prometedor sobre todo en el escenario de Internet de las Cosas (IoT: *Internet of Things*).

En esta Tesis se presenta el diseño, implementación y caracterización de cifradores de flujo de bajo consumo de potencia. Para la selección de estos cifradores, nos hemos centrado en las propuestas de cifradores de flujo (*stream ciphers*) que el Proyecto *eSTREAM* de la Red Europea de Excelencia en Criptología, *ECRYPT* (*European Network of Excellence in Cryptology*) seleccionó como las que presentaban mejores prestaciones en sus implementaciones hardware. Entre ellos se eligió al cifrador Trivium para proponer arquitecturas eficientes en términos de consumo de potencia aplicando la técnica de paralelización en diferentes diseños a nivel lógico. La aplicación de esta técnica y la elección del cifrador Trivium han sido tomadas tras una revisión del estado del arte sobre las implementaciones hardware dentro de los algoritmos criptográficos con bajo consumo de recursos y del consumo de potencia.

Se han realizado dos propuestas de cifradores de flujo Trivium con arquitecturas eficientes para bajo consumo denominadas MPLP (*Mixed Parallel Low Power*) y FPLP (*Full Parallel Low Power)*. Estas propuestas han sido diseñadas y simuladas con uno, dos, ocho y dieciséis bits a la salida para obtener resultados de recursos, área y consumo de potencia. Estos resultados se han comparado con los obtenidos para la versión estándar con los mismos bits de salida. Se han utilizado para estos diseños distintas librerías de celdas estándar de diferentes tecnologías de fabricación de circuitos integrados y de dispositivos programables, FPGA. Como

tecnologías ASIC se han usado tecnologías submicrónicas y nanométricas, en concreto, 350 nm, 180 nm, 130 nm, y 90 nm. En FPGAs se han utilizado las familias de dispositivos de *Xilinx Spartan-3E* y *Virtex-5*. Finalmente, para la verificación experimental, se ha implementado, mandado a fabricar y testado un circuito integrado (ASIC) en una tecnología de 90 nm, que contiene las propuestas de los cifradores Trivium con y sin técnicas de bajo consumo. Estas propuestas incluyen cifradores con salidas de uno, dos, ocho y dieciséis bits, y se han obtenido resultados experimentales de consumo de potencia y otras medidas con el equipo de test de señal mixta *Agilent 93000.*

Como resultado del trabajo realizado se puede sintetizar y destacar como conclusión final que las propuestas de diseño Trivium presentadas mejoran el consumo de potencia de la versión estándar en los casos de uno, dos y ocho bits a la salida. No es así en el caso de dieciséis bits. Con respecto a los recursos empleados, estos crecen significativamente con el aumento del número de bits de salida, aunque las versiones de bajo consumo MPLP y FPLP mantienen una buena relación en los recursos empleados y en el área en las propuestas de un bit y dos bits.

Todas las medidas realizadas, tanto por simulación como experimentalmente, avalan los datos de mejora en el consumo, siendo los casos de uno y dos bits de los cifradores Trivium FPLP y MPLP los que mejores resultados ofrecen en su conjunto, mejora que es extraordinariamente importante en el caso de un bit donde se alcanza el 50% manteniendo prácticamente los mismos costes en recursos y prestaciones en velocidad de operación.

Índice

[1] Los apéndices solo aparecen en la edición electrónica (pdf).

1. INTRODUCCIÓN

El uso cada vez más extenso que en la sociedad actual se hace de las Tecnologías de la Información y la Comunicación (TIC), hace de la seguridad y privacidad un reto permanente [1]. Las TIC se extienden en un amplio abanico de aplicaciones, desde sistemas informáticos de grandes prestaciones hasta sensores corporales, pasando por equipos médicos, ordenadores personales y telefonía por nombrar algunas de ellas.

En toda comunicación de datos se hace necesario proteger y garantizar de una manera adecuada la información enviada para evitar que un observador no autorizado pueda acceder o cambiar la información o suplantar identidades. Cada vez más, estas comunicaciones se realizan entre dispositivos electrónicos portables, cuyo funcionamiento a base de baterías y sus limitados recursos de computación imponen fuertes condicionantes a los mecanismos de cifrado y autenticación. En particular, el uso de algoritmos criptográficos en sistemas portátiles e inalámbricos de muy bajo consumo se ha convertido en algo muy común hoy en día y presenta un futuro muy prometedor en el escenario de Internet de las Cosas (IoT: *Internet of Things*) [2, 3]. Esto hace que la elección de algoritmos criptográficos de bajo consumo de recursos (LWC: *lightweight cryptography*) [4, 5] y la aplicación de técnicas que minimicen el consumo de potencia sea un objetivo fundamental de diseño.

Por otro lado, la microelectrónica constituye un motor ineludible en las aplicaciones de las TIC y de IoT y lo es tanto desde la perspectiva de la tecnología que soporta esas aplicaciones como por el hecho de ser una materia sujeta a un continuo desarrollo que produce innovación de las propias aplicaciones. Además, en nuestro caso, la pertenencia al IMSE (Instituto de Microelectrónica de Sevilla) hace que este campo de investigación, el de la microelectrónica, sea inherente a nuestro trabajo y a nuestra investigación desarrollada en el Instituto.

Los circuitos criptográficos se relacionan necesariamente con la tecnología en la que son implementados. Las técnicas de diseño microelectrónicas deben suministrar mecanismos que consigan no solo la implementación de los circuitos criptográficos mediante hardware específico, sino también que esas implementaciones sean lo más eficientes posibles.

Esta relación abre un campo de exploración que se ha desarrollado como tema de investigación de esta Tesis. El objetivo de esta es aplicar una técnica de bajo consumo para reducir el consumo de potencia en cifradores de flujo, diseñarlos, implementarlos en tecnologías microelectrónicas y caracterizarlos para, especialmente, medir su consumo de potencia y la reducción obtenida. Se ha diseñado un ASIC que ha incluido todos los diseños de los cifradores de flujo Trivium propuestos. También se ha realizado el test y la caracterización experimental de todos los diseños realizados.

1.1. Presentación

En este documento se presenta el trabajo de investigación que será defendido como Tesis Doctoral. Se ha desarrollado en el marco del Programa de Doctorado en Ingeniería Informática bajo la línea de Informática Industrial y Tecnología Electrónica dentro del grupo de investigación de la Junta de Andalucía (TIC-180) y en el Instituto de Microelectrónica de Sevilla (IMSE, del Centro Nacional de Microelectrónica, CNM-CSIC, Universidad de Sevilla). El trabajo se ha ejecutado en el marco de varios proyectos de investigación del grupo TIC-180, financiados en convocatorias nacionales y regionales. Son estos los proyectos: Cripto-Bio:" Diseño Microelectrónico para Autenticación Cripto-Biométrica", Junta de Andalucía, (TEC2008-3674), CITIES: "Circuitos integrados para transmisión de información especialmente segura", Ministerio de Ciencia e Innovación (TEC2010-16870) y CESAR: "Circuitos microlectrónicos seguros frente a ataques laterales", Ministerio de Economía y Competitividad (TEC2013-45523-R).

Esta Tesis Doctoral se encuadra dentro de la necesidad de ofrecer soluciones hardware para la realización de cifrado y descifrado seguro en aplicaciones en las que haya pocos recursos hardware y las restricciones en el consumo sean importantes. De entre las diferentes propuestas de algoritmos criptográficos hemos seleccionado las iniciativas de la Unión Europea que han mostrado un alto interés por el hardware criptográfico en sus últimos Programas Marco. En concreto, nos hemos fijado en el Proyecto eSTREAM de la Red Europea de Excelencia en Criptología, ECRYPT (*European Network of Excellence in Cryptology*) que seleccionó algunos algoritmos de cifrado de flujo (*stream ciphers*) que podían presentar buenas prestaciones en sus implementaciones hardware y software. La operación de los cifradores de flujo consiste en mezclar el contenido del mensaje que se quiere proteger, con una secuencia pseudo-aleatoria generada por un algoritmo criptográfico bajo el control de una clave secreta, y un valor de inicialización. El cifrador Trivium, surgido de dicha propuesta, es un excelente candidato que busca el equilibrio entre simplicidad, seguridad y velocidad. De aquí que lo hayamos elegido como el cifrador de flujo objeto de investigación de esta Tesis.

De otro lado, durante toda su historia la progresión de la microelectrónica ha sido extraordinaria. El continuo escalado de las tecnologías ha propiciado incrementos en las densidades de integración de los circuitos integrados y en la frecuencia de reloj. Esto ha generado un aumento de las prestaciones de los circuitos y de su funcionalidad. Sin embargo, la miniaturización de las tecnologías ha hecho que los diseñadores de circuitos se enfrenten con nuevos retos, uno de ellos es el consumo de potencia, que resulta de gran interés para una variedad de aplicaciones de sistemas portables e inalámbricos (RFID, *smart-cards*, dispositivos médicos, sensores, etc).

Un consumo excesivo en estos dispositivos electrónicos limita la portabilidad (disminución de la vida efectiva de las baterías) y la fiabilidad (estrés térmico debido al aumento de temperatura). A medida que se añaden más características a un producto electrónico portable, su consumo de energía aumenta y la duración de la batería se reduce, lo que puede requerir de una batería más grande, más pesada o de tiempo de vida más corto entre cargas.

Un consumo de energía alto también puede dar lugar a temperaturas excesivamente altas durante el funcionamiento del dispositivo. Esto puede ocasionar que el encapsulado del circuito integrado ya no sea el adecuado y tenga que cambiarse. Por ejemplo, si el chip lleva un encapsulado de plástico se debería cambiar a un encapsulado cerámico, con mejor disipación térmica, pero con un sobrecoste añadido. En otros casos, para mejorar el funcionamiento del producto electrónico habría que añadirle disipadores de calor, complejos y costosos, e incluso sistemas de refrigeración. Temperaturas de funcionamiento más altas también reducen la fiabilidad debido a la electromigración [6] y otros mecanismos de fallos relacionados con el exceso de temperatura [7].

Hoy en día, el diseño e implementación de circuitos integrados en tecnologías VLSI (*Very Large Scale Integration*) plantea numerosos desafíos y conlleva tal complejidad que resulta imprescindible el uso de metodologías y herramientas de ayuda al diseño para su desarrollo. Esta complejidad no solo es debida a la funcionalidad de los circuitos actuales, sino también a otros aspectos relacionados con ellos, como son las tecnologías de fabricación, prestaciones, el consumo, la testabilidad, la fiabilidad, etc. Ligadas a las tecnologías de fabricación se debe de tener en cuenta la facilidad de acceso a ellas y en qué términos, pues muchas de las tecnologías presentan diferencias en cuanto al coste de su uso, librería de celdas disponibles (digitales y analógicas), modelos de simulación a nivel de transistor y/o a nivel lógico para sus celdas, número de integraciones durante el año, soporte, documentación de la tecnología, etc.

El proceso de diseño de las diferentes propuestas de cifradores de flujo Trivium se lleva a cabo haciendo uso de unas herramientas y entornos de diseño CAD que facilitan el camino entre las ideas de diseño y las tecnologías de implementación de

circuitos integrados. En concreto, el proceso seguido para el diseño e implementación hardware de las diferentes propuestas de cifradores de flujo Trivium con arquitecturas eficientes para bajo consumo parte de la descripción de las arquitecturas a partir de un lenguaje de descripción hardware (HDL: *Hardware Description Language*). Las ventajas de estos lenguajes son considerables en cuanto a la posibilidad de explorar diversas alternativas de un mismo diseño, de verificar su comportamiento funcional a un nivel de abstracción más alto y de independizar las descripciones frente a las tecnologías de fabricación. Los restantes pasos del proceso de diseño son la traslación de la descripción hardware a una descripción a nivel de celdas lógicas de una librería tecnológica específica (síntesis lógica) y su verificación tanto funcional como temporal. Por último, se tiene una fase de implementación física donde se realiza el emplazamiento y conexionado de todas las celdas lógicas y se llega al *layout*.

Para el desarrollo de los trabajos de esta Tesis se han utilizado diferentes herramientas CAD [8] en todo el flujo de diseño. En particular, se ha usado *Synopsys Design Vision* [9] para la síntesis de las descripciones VHDL [10, 11] y *Cadence Design Framework II* [8] para el diseño a nivel de esquemáticos. Para las verificaciones lógicas y temporales se han usado los simuladores *ModelSim SE 6.6d* de *Mentor Graphics* [12] y *NCsim* de *Cadence* [13], así como simuladores analógicos y de señal mixta como *Spectre* y *SpectreVerilog* [14]. Para la implementación física se ha utilizado la herramienta *Encounter Digital Implementation System RTL-to-GDSII* de *Cadence* [15] con la que se ha llevado a cabo la etapa de emplazamiento y conexionado. Finalmente se ha usado la herramienta *Virtuoso* [16] dentro del entorno de *Cadence* para el conexionado y emplazamiento final y *Calibre* de *Mentor Graphics* [17] para realizar las verificaciones físicas del *layout*, antes de su envío a fabricación.

Finalmente, el ASIC fabricado ha sido testado y caracterizado en el laboratorio. Esta tarea a veces conlleva un gran coste económico y siempre tiene un coste elevado de tiempo y esfuerzo. En nuestro caso, para la caracterización y test de los circuitos integrados en el laboratorio, se ha utilizado el equipo de prueba de señal mixta *Agilent 93000*, el cual nos ha permitido aumentar la flexibilidad y versatilidad de las diferentes tareas que se han tenido que llevar a cabo, además de haber ayudado en la reproducibilidad de las pruebas de caracterización del circuito, acortando considerablemente el tiempo de testado.

1.2. Objetivos

En esta Tesis el diseño de circuitos criptográficos se estudia desde una perspectiva microelectrónica en el nivel lógico y su implementación será como circuito integrado VLSI usando una de las tecnologías de integración accesibles desde el IMSE/US. También se han realizado implementaciones en FPGA (*Field-Programmable Gate Array*) del fabricante *Xilinx*.

Para lograr el propósito de esta Tesis, es necesario alcanzar los siguientes objetivos del trabajo de investigación:

- Objetivos de documentación:
  - Establecer el estado del arte en que se encuentra este campo de estudio mediante la presentación de la información recopilada más relevante aparecida sobre el tema que ocupa la Tesis.
- Objetivos de diseño:
  - Elegir un cifrador entre los finalistas de la iniciativa eSTREAM.
  - Aplicar técnicas de reducción de consumo para obtener cifradores de bajo consumo.
  - Diseñar cifradores de bajo consumo con salida para bases de diferentes valores (bases 1, 2, 8 y 16.).
- Objetivos de implementación en las tecnologías disponibles:
  - Implementar los cifradores en varias familias de FPGAs. Analizar sus prestaciones.
  - Implementar los cifradores en tecnologías ASIC de muy baja longitud de integración (submicrónica y nanométricas).
  - Diseñar y enviar a fabricación un prototipo ASIC con las propuestas de los cifradores de bajo consumo.
- Objetivos del test:
  - Diseñar la estrategia de test de los circuitos integrados en el ASIC.
  - Diseñar e implementar la placa de test y configurar el equipo de pruebas.
  - Realizar la verificación funcional de los prototipos.
  - Caracterizar el comportamiento de los circuitos implementados variando los parámetros de la tecnología utilizada.
  - Medir el consumo de cada uno de los cifradores integrados.

1.3. Contenido

La organización de este documento es de la siguiente forma. La memoria de esta Tesis se ha dividido en seis capítulos. Tras el presente capítulo de introducción, el Capítulo 2 presenta el estado del arte en las líneas vinculadas al trabajo de investigación desarrollado. En particular, se describe la iniciativa europea del Proyecto eSTREAM y los cifradores de flujo seleccionados, centrándonos en el cifrador Trivium. Además se introducen los detalles previos e imprescindibles sobre los mecanismos de disipación de potencia en circuitos digitales CMOS y las opciones tecnológicas y de diseño para su reducción.

En el Capítulo 3 se presenta la técnica de paralelización que ha sido aplicada a los cifradores Trivium para reducir su consumo. Para ello se analiza el consumo de potencia de los registros de desplazamiento y se presentan y analizan datos de simulaciones lógicas y eléctricas en un registro de desplazamiento simple y también en otro paralelizado de 8 bits. Posteriormente se presentan las arquitecturas propuestas de los cifradores de flujo Trivium de bajo consumo de potencia junto con un análisis de área y consumo en diseños con diferentes tecnologías ASIC y FPGA.

A continuación el Capítulo 4 presenta la descripción y el diseño de un prototipo ASIC en el que se incluyen las propuestas de los cifradores de bajo consumo desarrollados en este trabajo de investigación. Se presentan también diversos aspectos relacionados con su funcionamiento e implementación, para acabar analizando los datos de área y consumo de todas las propuestas en las diferentes etapas del diseño.

El Capítulo 5 presenta otro de los grandes objetivos de esta tesis, el test del prototipo ASIC que incluye las propuestas de los diferentes cifradores de bajo consumo en el laboratorio. El test ha sido realizado con el equipo de test *Agilent 93000*, del que se comentan las principales características. Se presenta el plan de test desarrollado y los principales resultados obtenidos en el test de los prototipos.

Finalmente el Capítulo 6 recoge las principales conclusiones y la Bibliografía completa la memoria de esta Tesis.

Además, esta memoria contiene tres anexos. El primero incluye los códigos VHDL de los cifradores Trivium, el anexo segundo contiene *scripts* y ficheros utilizados durante el flujo de diseño así como algunos ficheros de *testbench* usados en las diversas simulaciones. Finalmente, el anexo tercero amplía la información relativa a los datos obtenidos en el test de las diferentes muestras. Estos tres anexos solo se incluyen en formato electrónico (dentro del CD), no apareciendo en la versión en papel.

2. CIFRADORES DE FLUJO Y CONSUMO DE POTENCIA

La necesidad cada vez mayor de hacer segura y privada la información transmitida entre diversos sistemas electrónicos está haciendo que el uso de dispositivos criptográficos en sistemas portátiles e inalámbricos de muy bajo consumo de recursos sea muy común hoy en día y con un futuro muy prometedor en el escenario de Internet de las Cosas. Esto nos lleva a pensar que la elección de algoritmos criptográficos con bajo consumo de potencia y de bajo consumo de recursos (LWC), junto con la aplicación de técnicas que reduzcan el consumo de potencia sean un objetivo fundamental para el diseño.

En la criptografía moderna [18, 19] se tiene dos métodos de cifrado de la información, los sistemas de criptografía simétricos y los asimétricos. En los primeros se utiliza la misma clave para codificar y decodificar. Esto conlleva a que el emisor y el receptor tengan la misma clave y si esta por algún motivo se envía debe de hacerse por un canal seguro. Sin embargo en los sistemas asimétricos se tiene una clave para la codificación y otra para la decodificación de la información. En este caso no es necesario enviar la clave pues cada receptor tiene la suya propia.

Dentro de los algoritmos criptográficos con bajo consumo de recursos [20-22] destacan los cifradores de clave simétrica o privada, los cuales a su vez se subdividen en dos grupos fundamentales, según el número de bits que cifren:

- Cifradores de bloque (*block ciphers*), los cuales procesan la información en bloques relativamente grandes que pueden ser 64, 128 o 256 bits dependiendo del método utilizado. Cuanto mayor sea el tamaño del bloque, más seguro es el sistema de cifrado, aunque se aumenta la complejidad de los algoritmos. Además, siempre aplican la misma función a la información por lo que se dice que no poseen memoria [5]. Algunos de los cifradores propuestos toman como base para mejorar sus prestaciones al cifrador AES (*Advanced Encryption Standard*) del NIST (*National Institute of Standards and Technology*) [4, 5, 22]. Entre los cifradores de bloques para LWC se destacan a DESL y DESLXL [23], CLEFIA [24], PRESENT [5, 25] y SIMON [26].

- Cifradores de flujo (*stream ciphers*), cifran la información bit a bit, utilizando registros de desplazamiento con realimentación no lineal. No se esperan a completar un bloque. Se dice que tienen memoria o estado interno y por lo tanto el cifrado depende del contenido de este estado interno además de la información y de la clave. Algunos de los propuestos surgen del proyecto eSTREAM [27-29] del que se hablará en el siguiente apartado. Entre los cifradores de flujo para LWC se destacan a Grain [30], Trivium [31], y Mickey [29].

Los cifradores de flujo suelen ser más rápidos que los cifradores de bloque, sobre todo en hardware, y requieren de una circuitería más sencilla. Son también más adecuados para situaciones de tiempo real en las que la latencia es reducida.

Desde el punto de vista funcional, los cifradores de flujo resultan ideales en sistemas hardware que necesiten cifrar en modo continuo mientras que los cifradores en bloque son más cómodos para implementaciones en software, ya que pueden trabajar sobre un grupo de bits con bloques grandes de datos.

La idea principal del cifrado de flujo se basa en generar una secuencia criptográfica binaria larga y no previsible a partir de una clave elegida de forma aleatoria. El cifrado de flujo es cómodo, sencillo y rápido.

En el procesamiento de cifrado de flujo [18], esquema de la Figura 2.1, el emisor transmite el mensaje haciendo uso de una clave secreta y un algoritmo público. Se genera una secuencia pseudoaleatoria, cuyos elementos (c_i) se mezclan mediante la operación XOR con los correspondientes bits del mensaje (m_i), dando lugar a los bits de la información cifrada, (z_i). Esta secuencia cifrada es enviada a través de un canal público sin protección (inseguro). El receptor, con la misma clave y el mismo algoritmo determinístico, genera la misma secuencia cifrante (c_i), que se mezclan mediante la operación XOR con la secuencia cifrada recibida (z_i), dando lugar a los mismos bits del mensaje original, (m_i). La clave es la única información secreta que tienen que compartir emisor y receptor ya que el algoritmo es público.

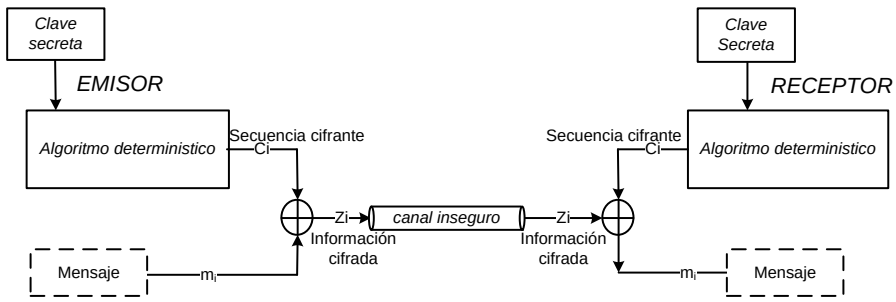


Figura 2.1 Esquema de cifrado de flujo.

A continuación se van a presentar las propuestas surgidas en el ámbito europeo para estimular y avanzar en la búsqueda de algoritmos de cifrado de flujo eficiente y compacto para uso general con la convocatoria eSTREAM, centrándonos en el cifrador Trivium. Posteriormente se hace una breve presentación de las fuentes de disipación de potencia que afectan principalmente a los circuitos digitales CMOS y los principios claves para su reducción.

2.1. Proyecto eSTREAM

En Noviembre del 2004 la Red Europea de Excelencia en Criptología, ECRYPT inició el proyecto eSTREAM con el propósito explícito de dinamizar y avanzar en la búsqueda de algoritmos de cifrado de flujo eficientes y compactos para uso general. Toda la información correspondiente a este proyecto puede encontrarse en [27].

La intención de la convocatoria eSTREAM fue la de estimular trabajos en el área del cifrado de flujo y fomentar su uso ya que en los últimos años este tipo de cifrador había sufrido una constante pérdida de interés. Se pretendía seleccionar un algoritmo de cifrado de flujo que tuviese una amplia difusión y uso generalizado.

Como parte de esta iniciativa, se investigaron nuevos cifradores de flujo que podrían convertirse en adecuados para una adopción generalizada. La convocatoria finalizó en 2008 aunque aún mantiene su actividad. Una nueva revisión de los perfiles ha sido expuesta en el 2012. Los cifradores fueron divididos en dos perfiles, dependiendo de la aplicación prevista:

- Perfil 1: cifradores de flujo para aplicaciones software con altas prestaciones de salida de secuencia cifrante.
- Perfil 2: cifradores de flujo para aplicaciones hardware con limitaciones en lo que respecta a memoria utilizada, número de puertas y consumo de potencia. En este caso la longitud de la clave tenía un mínimo de 80 bits.

La convocatoria eSTREAM fue muy apoyada y seguida por la comunidad criptográfica internacional y, tras seis meses, recibieron propuestas procedentes de más de cien grupos de criptografía de diversas nacionalidades y se seleccionaron 34 algoritmos de cifrado en flujo. Una comisión formada por expertos de reconocido prestigio internacional se encargó de evaluar los algoritmos presentados. Las características de cada una de estas propuestas pueden encontrarse con más detalle en [27, 29].

Los criterios más importantes exigidos a estas propuestas tenían relación con la seguridad, implementación y mercado.

- Seguridad: cualquier ataque criptoanalítico de recuperación de la clave tiene que ser al menos tan costoso como la búsqueda exhaustiva de ella.

- Implementación: una longitud de clave más un vector de inicialización de entre 64-128 bits. Cualquiera de las propuestas tenía que tener unas prestaciones superiores a las del AES, estándar criptográfico de cifrado de bloque.
- Mercado: flexibilidad, eficiencia y posibilidad de un uso generalizado a nivel mundial.

Tras una primera y segunda fase de evaluación, dieciséis propuestas pasaron a la tercera fase, ocho orientadas a su implementación software y otras ocho a su implementación hardware.

Finalmente, la Comisión encargada de seleccionar un candidato de cada perfil decidió un conjunto de cuatro algoritmos de implementación software, perfil 1 (SW) y tres de implementación hardware, perfil 2 (HW), que son los que aparecen en la Tabla 2.1.

De esta manera el proyecto eSTREAM finalmente no se redujo a proporcionar un único candidato de cada tipo, sino que se seleccionó un conjunto de varios para su uso y difusión. Uno de los más aceptados en la comunidad criptográfica es el cifrador Trivium, que es sobre el que hemos trabajado en esta Tesis.

Tabla 2.1 Algoritmos ganadores en la fase final.

Perfil 1 (Software)	**Perfil 2 (Hardware)**
HC-128	Grain v1
Rabbit	MICKEY v2
Salsa20/12	Trivium
SOSEMANUK	

2.2. Cifrador de flujo Trivium

El algoritmo Trivium fue desarrollado por C. De Cannière y B. Preneel [31] como un generador de secuencia cifrante orientado al hardware. Se trata de un cifrador de flujo síncrono que busca el equilibrio entre simplicidad, seguridad y velocidad. Para un análisis teórico más profundo se pueden consultar las referencias [28, 29, 31].

El cifrador Trivium está orientado al hardware y pretende ser: compacto en entornos con restricciones en el número de puertas lógicas, eficiente en implementaciones con recursos computacionales limitados y rápido en aplicaciones que requieran una alta velocidad de cifrado. Este algoritmo reúne todas estas características y, hasta el momento, es el más rápido de todos los presentados a la convocatoria eSTREAM.

A día de hoy, se considera inmune a los ataques del tipo: correlación, suposición y determinación y ataques algebraicos conocidos [32]. Esto tiene todavía mayor mérito teniendo en cuenta que, dada su simplicidad y velocidad, el cifrador Trivium ha estado y seguirá estando en el punto de mira de la comunidad criptoanalítica internacional.

Casi todos los cifradores de flujo tienen dos parámetros de entrada: una clave secreta, *key* y un vector de inicialización, *IV* que proporciona el estado inicial del cifrador. El primero se utiliza en todos los sistemas de cifrado simétrico mientras que el segundo, sirve de aleatoriedad y debe tomar un nuevo valor para cada sesión de cifrado. Es importante tener en cuenta que el vector de inicialización no tiene que ser mantenido en secreto, simplemente debe cambiar para cada sesión. En caso contrario, la información cifrada se vuelve altamente determinista.

El generador Trivium está basado en tres registros de desplazamiento, tiene una clave secreta de 80 bits y un vector de inicialización de otros 80 bits, pudiendo generar hasta 2^{64} bits consecutivos de secuencia cifrante (*key stream*).

Como en muchos de los cifradores de flujo síncrono, el proceso de funcionamiento consta de dos fases, la fase de inicialización y la fase de desplazamiento. En la fase de inicialización se carga el estado interno del cifrador con los valores de la clave secreta y del vector de estado inicial. En la fase de desplazamiento el estado interno es actualizado para generar la secuencia cifrante.

El cifrador Trivium contiene un registro de estado interno de 288 bits, dividido en tres registros de 93, 84 y 111 bits respectivamente. El primer registro incluye desde los bits 92 al 0, el segundo desde los bits 176 al 93, y el tercero, los restantes bits, desde el 287 al 177. Cada bloque está conectado con el bloque contiguo dando lugar a un registro circular. A su vez, cada bloque se puede considerar como un registro de desplazamiento no lineal puesto que la realimentación incluye no solo operaciones XOR sino también operaciones AND.

La generación de la secuencia se realiza mediante un proceso iterativo que extrae los valores de 15 bits del registro de estado interno y los utiliza para actualizar otros 3 bits (*t1*, *t2*, *t3*) que son reintroducidos en los biestables 93, 0 y 177, respectivamente. De aquí surgen los valores de *k1*, *k2* y *k3* con los que, finalmente, se calcula el correspondiente bit z_i de secuencia cifrante. Los bits de estado son posteriormente rotados y el proceso se repite hasta generar los 2^{64} posibles bits de secuencia de salida correspondientes a una misma clave.

Un esquema hardware general del cifrador Trivium aparece representado en la Figura 2.2.

La fase de inicialización conlleva la carga de una clave de 80 bits y un vector de inicialización de 80 bits en el registro de estado interno, dejando los bits restantes a

0, excepto los correspondientes a los bits del registro de estado 285, 286 y 287 que se cargan con nivel lógico 1.

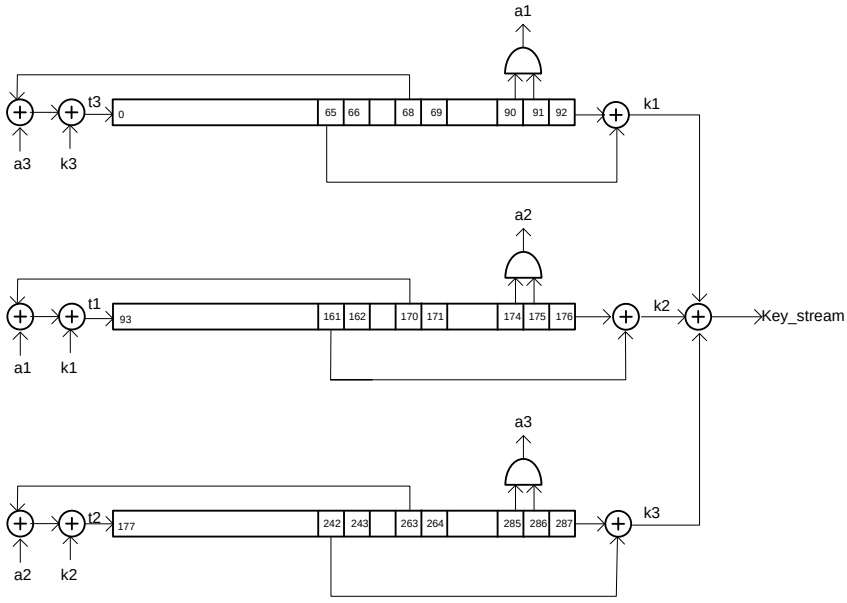


Figura 2.2 Esquema hardware general del cifrador Trivium.

El funcionamiento puede describirse en VHDL [10, 11] tal y como se describe a continuación.

- Registro de estado 1: `state(92 downto 0) <= X"0" & key;`
- Registro de estado 2: `state1(76 downto 93) <= X"0" & IV;`
- Registro de estado 3: `state(287 downto 177) <= "111" & X"0";`

Posteriormente a esta fase se calculan los valores de las señales k_i y a_i como:

- `k1 <= state(65) xor state(92);`
- `a1 <= state(90) and state(91);`
- `k2 <= state(161) xor state(176);`
- `a2 <= state(174) and state(175);`
- `k3 <= state(242) xor state(287);`
- `a3 <= state(285) and state(286);`

Los bits t_i introducidos en cada uno de los registros de estado (registros de desplazamiento no lineal) del Trivium son generados mediante operaciones XOR y AND con las señales (k_i, a_i) previamente calculadas.

- `t1 <= k1 xor a1 xor state(170);`
- `t2 <= k2 xor a2 xor state(263);`
- `t3 <= k3 xor a3 xor state(68);`

El registro de estado interno es rotado 4 veces completamente, es decir 1152 desplazamientos (4x288), antes de obtener un bit válido de secuencia cifrante.

La salida de la secuencia cifrante o *key stream* (z_i) se obtiene como una XOR de las señales k_i.

- `key_stream <= k1 xor k2 xor k3;`

El desplazamiento del registro de estado interno tiene como objetivo aleatorizar adecuadamente el cifrador y así asegurar que la secuencia cifrante no esté correlacionada con la clave y del estado de inicialización.

Nótese que el número que se repite sistemáticamente en este generador es el número 3. Primero se calculan 3 variables (t_i) para determinar el correspondiente bit de salida, después se actualizan esos 3 parámetros y finalmente se produce la rotación de los 3 bloques. De ahí, a decir de los autores, el nombre de Trivium.

2.2.1. Trivium multi-bit

La descripción del cifrador Trivium que se ha realizado en el apartado anterior genera un bit en cada ciclo de reloj. Sin embargo el algoritmo del cifrador Trivium permite realizar implementaciones que generen hasta 64 bits en cada ciclo de reloj [31]. Estas implementaciones mantienen el mismo número de bits en el registro de estado interno aunque incrementan los elementos de realimentación y los bits de desplazamiento en el modo de operación.

Las implementaciones paralelas consiguen una mayor capacidad de generación de bits de secuencia cifrante pero a costa de un incremento de coste de potencia y de una disminución de la frecuencia máxima de operación. En las implementaciones paralelas también se ven afectados el número de ciclos necesarios para generar una secuencia cifrante válida. Si en el caso de un solo bit el número de ciclos es de 1152, para el de 64 bits el número de ciclos se reduce a tan solo 18.

Un esquema de una implementación hardware de un cifrador Trivium de *m* bits se muestra en la Figura 2.3 donde se multiplica también por un factor *m* el número de sumas lógicas y productos. En este caso los registros internos son cargados dependiendo del número de bits a la salida (*m*) y se pueden expresar en código VHDL, teniendo en cuenta que *n=m-1*, como:

- Registro de estado 1: `state(92 downto 0) <= state((90-n) downto 0) & t3(n downto 0);`
- Registro de estado 2: `state(176 downto 93) <= state((174-n) downto (92-n)) & t1(n downto 0);`
- Registro de estado 3: `state(287 downto 177) <= state((285-n) downto (176-n)) & t2(n downto 0);`

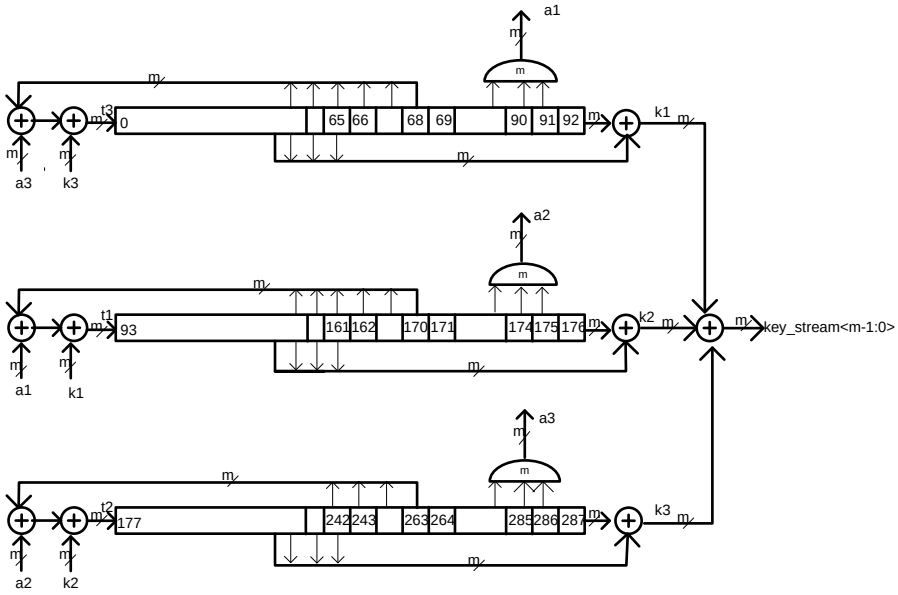


Figura 2.3 Esquema hardware general del cifrador Trivium multi-bit.

De igual manera las operaciones de realimentación vienen también condicionadas por el número de bits a la salida y se pueden expresar en código VHDL como:

- ```
  G1: for i in 0 to n generate
  k1(i) <= state1(65-i) XOR state1(92-i);
  end generate G1;
  ```
- ```
  G2: for i in 0 to n generate
  k2(i) <= state1(161-i) XOR state1(176-i);
  end generate G2;
  ```
- ```
  G3: for i in 0 to n generate
  k3(i) <= state1(241-i) XOR state1(286-i);
  end generate G3;
  ```
- ```
  G4: for i in 0 to n generate
  t1(i) <= k1(i) XOR ((state1(90-i) AND state1(91-i))
  XOR state1(70-i));
  end generate G4;
  ```
- ```
  G5: for i in 0 to n generate
  t2(i) <= k2(i) XOR ((state1(174-i) AND state1(175-i))
  XOR state1(263-i));
  end generate G5;
  ```
- ```
  G6: for i in 0 to n generate
  t3(i) <= k3(i) XOR ((state1(285-i) AND state1(286-i))
  XOR state1(68-i));
  end generate G6;
  ```

La salida de secuencia cifrante multi-bit vendrá dada por la operación XOR de las señales *k1(i)*, *k2(i)* y *k3(i)*, estando descrita la operación con código VHDL:

```
G7: for i in 0 to n generate
key_stream(i) <= (k1(i) XOR k2(i) XOR k3(i));
end generate G7;
```

2.2.2. Aspectos sobre las implementaciones de los cifradoresTrivium

En la Tabla 2.2 se muestra la complejidad en el número de celdas y transistores las implementaciones de cifradores Trivium de 1 hasta 64 bits [31]. Es habitual en algunos diseños y referencias tomar la celda NAND como patrón para realizar una medida del número equivalente de puertas o transistores. En estos casos se toma para realizar una estimación de transistores la celda patrón NAND de 2 entradas, la cual tiene 4 transistores en su implementación. A veces se da también el área en función de esta celda.

Tabla 2.2 Número de puertas en implementaciones Trivium de 1 a 64 bits.

Componentes	**Nº transistores**	**1bit**	**2bits**	**8bits**	**16bits**	**32bits**	**64bits**
Flip-flops	26	288	288	288	288	288	288
AND	6	3	6	24	48	96	192
XOR	12	11	22	88	176	352	704
Nº transistores	-	7638	7788	8688	9888	12288	17088

La eficiencia del cifrador Trivium en hardware ha sido evaluada en la literatura con diferentes tecnologías CMOS y FPGAs. En tecnologías CMOS se han reportado datos del consumo en 350 nm [33], y 130 nm [33-36]. También se han reportado en ellos datos de frecuencia máxima, bits procesados por ciclo de reloj, velocidad de procesamiento, transferencia máxima datos (*throughput*) y área.

En [34, 35] se reporta un estudio muy completo de los candidatos en hardware en la fase II del Proyecto eSTREAM, con datos de frecuencia máxima, área, corriente, consumo total y otros parámetros así como con diferentes métricas de prestaciones según la aplicación final del cifrador. También se analiza el cifrador Trivium con salidas multi-bit, desde 1 bit a 64 bit. En [36] se reportan datos del consumo de potencia así como del camino crítico de retardo y área del Trivium con salida de 1 bit a varias frecuencias de operación. En [37] se analizan diferentes estructuras implementadas en una tecnología de 250 nm y se dan datos de área, frecuencia máxima y *throughput*.

En [33] se emplea el uso de otro estilo de lógica, en este caso SABL (*sense amplifier base logic*) para lograr en este caso una mayor resistencia a ataques DPA (*Differential*

Power Analysis) en el cifrador Trivium [38]. Se dan datos de consumo con lógica CMOS y SABL en dos tecnologías diferentes, 350 nm y 130 nm [33].

Las referencias al empleo de alguna técnica para disminuir el consumo de potencia son también muy escasas, destacando [39] donde se aplica la técnica de *clock gating* para disminuir la potencia pero sin dar datos sobre la reducción frente a la versión sin *clock gating*.

En la Tabla 2.3 se resumen los datos ofrecidos por las referencias más destacadas relativas a área y consumo en tecnologías ASIC. En la tabla se muestran, además, los valores del número de bits a la salida del cifrador Trivium, la tecnología de fabricación, la tensión de alimentación del circuito y la frecuencia de operación.

Tabla 2.3 Datos de área y consumo en tecnologías ASIC para Trivium.

Ref.	Trivium bits salida	Tecnología ASIC (nm)	V_{DD} (V)	Reloj MHz	Área	Consumo
			ASIC			
[33]	1	350	3.3	5	8500 trans.	641 μW
[33]	1	350 SABL	3.3	5	23000 trans.	949 μW
[33]	1	130	1.2	5	8500 trans.	337 μW
[33]	1	130 SABL	1.2	5	23000 trans.	545 μW
[34, 35]	1	130	1.2	10	2580 NAND	175,1 μW
[34]	2	130	1.2	10	2627 NAND	182,7 μW
[34, 35]	4	130	1.2	10	2705 NAND	184,6 μW
[34, 35]	8	130	1.2	10	2952 NAND	203,4μW
[34, 35]	16	130	1.2	10	3185 NAND	214,4 μW
[34, 35]	32	130	1,2	10	3787 NAND	282,5 μW
[34, 35]	64	130	1,2	10	4921 NAND	374,2 μW
[36]	1	130	-	1	15058 μm^2	34,7 μW
[36]	1	130	-	10	15058 μm^2	227 μW
[36]	1	130	-	100	15058 μm^2	2,15 mW
[37]	64	250	-	312	144128 μm^2	-
[39]	16	350	1,5	0,1	3090 NAND	0,68 μA

[33], *dato de área en número de transistores.*

[33], *dato de área en número de transistores.*

[34, 35], *datos de post-layout. Se toma el flip-flop tipo D con enable como 8 puertas NAND.*

[36], *para calcular el número de puertas NAND multiplicar el área en μm^2 por 0,193.*

[37], *celda NAND2 de 23,76 μm^2, celda XOR2 de 55,44 μm^2, y flip-flop con scan de 205,92 μm^2.*

La comparación de implementaciones en diversas tecnologías de fabricación ASIC se hace difícil y complicada pues aunque sean a veces tecnologías del mismo tamaño, los parámetros internos en la librería de celdas estándar pueden ser muy distintos al provenir de fabricantes diferentes. Las referencias mostradas no incorporan en ningún caso técnicas para la reducción del consumo de potencia excepto la ya comentada [39]. El dato de área se da a veces como el número de

puertas equivalentes, dado por el área de la puerta NAND de dos entradas.

Entre otros datos reflejados en la Tabla 2.3 podemos destacar, en primer lugar, que las tecnologías ASIC usadas son submicrónicas pero no nanométricas (hay datos de 130 nm a 350 nm).

Los datos en FPGAS son también escasos y con poca información sobre el consumo de potencia. En la Tabla 2.4 se resumen los datos ofrecidos por las referencias más destacadas relativas a área y consumo en tecnologías FPGA. La mayoría de los autores trabajan con *Xilinx* [40], que es el mayor fabricante de dispositivos FPGA, pero es en [41, 42] y sobre *Altera* [43] donde se han publicado datos diversificando las salidas y aportando resultados de potencia, mientras que en otras dispositivos de las familias de *Xilinx* los datos de potencia son más bien escasos o nulos.

Tabla 2.4 Datos de área y consumo en tecnologías FPGA para Trivium.

Ref.	Trivium bits salida	Fabricante FPGA	V_{DD} (V)	Reloj MHz	Área	Consumo
			FPGA			
[41]	1	*Altera*	-	295	393 LE	382 mW
[41]	2	*Altera*	-	290	368 LE	381 mW
[41]	4	*Altera*	-	300	364 LE	393 mW
[41]	8	*Altera*	-	350	380 LE	435 mW
[41]	16	*Altera*	-	290	424 LE	457 mW
[41]	32	*Altera*	-	280	518 LE	540 mW
[41]	64	*Altera*	-	245	710 LE	670 mW
[42]	1	*Altera*	-	400	39 ALUTs	398 mW
[44]	1	*Xilinx*	-	201	188 CLBs	-
[44]	2	*Xilinx*	-	202	189 CLBs	-
[44]	4	*Xilinx*	-	203	199 CLBs	-
[44]	8	*Xilinx*	-	193	199 CLBs	-
[44]	16	*Xilinx*	-	191	227 CLBs	-
[44]	32	*Xilinx*	-	202	264 CLBs	-
[44]	64	*Xilinx*	-	190	388 CLBs	-
[45]	1	*Xilinx*	-	207-	41 Slices	-
[46]	1	*Xilinx*	-	240-	50 Slices	-
[46]	64	*Xilinx*	-	211-	344 Slices	-

[41], *Altera Cyclone EP1C20F400C6.*

[42], *Altera Stratix III EP3SE50F484C2.*

[44], *Xilinx Spartan 3, xc3s400fg320-5. No hay datos de potencia.*

[45], *Xilinx Virtex II. No hay datos de potencia.*

[45], *Xilinx Spartan II. No hay datos de potencia, ni frecuencia.*

[46], *Xilinx Spartan 3, xc3s50-5pq208. No hay datos de potencia.*

[46], *Xilinx Spartan 3, xc3s400-5fg320. No hay datos de potencia.*

En [41] se presenta un estudio de implementaciones hardware en dispositivos *Altera* de la familia *Cyclone* con datos de consumo de potencia para Trivium con salidas desde 1 bit hasta 64 bit, además del área consumida y otros parámetros. En [42] se usa otra familia de *Altera, Stratix III,* y se dan también datos de consumo de potencia para el cifrador Trivium de un solo bit. Se comparan también parámetros como la frecuencia máxima, área y *throughput* con otros cifradores pero no se hace esta comparación con el consumo de potencia. En [44] se tienen datos en dispositivos de la familia *Spartan 3* de *Xilinx* con medidas de frecuencia de reloj, área, y otros parámetros pero sin aportar datos del consumo de potencia.

El caso Trivium mas estudiado es el de salida de un bit, aunque también se han estudiado los cifradores multi-bit. En [34, 35] puede verse como se incrementa el área al subir el número de bit de salida. Las frecuencias del reloj abarcan para las tecnologías ASICs rangos desde 100 KHz [34] hasta los 312 MHz en [37], mientras que los estudios en FPGAs presentan en general frecuencias más elevadas (entre 190-350 MHz). El dato de área es difícil de comparar en ASICs, pues unos autores los dan en número de puertas NAND, otros en μm^2 y otros en números de transistores. En las FPGAs se dan según la familia de dispositivo utilizada, en número de *Slices* y bloques lógicos programables (CLBS: *configurable logic blocks)* en el caso de *Xilinx* o en número de elementos lógicos (LE: *Logic Element*) o ALUT (*Adaptive Look Up Table*) en el caso de *Altera*.

La estructura interna de los CLBs difiere en las distintas familias de *Xilinx*, aunque todos ellos presentan ciertas semejanzas en cuanto a que un CLB en estas familias contiene dos *Slices* que tienen generadores de funciones o tablas de verdad (LUTs: *look-up tables*), multiplexores dedicados y lógica de acarreo. Cada *Slice* de las familias Spartan-3E contiene cuatro LUTs de cuatro entradas, lógica de acarreo y dos *flip-flops*.

En el caso de las familias *Virtex-5* cada *Slice* contiene también cuatro LUTS pero a diferencia de la *Spartan-3E* el número de entradas y *flip-flops* es mayor; seis entradas y cuatro *flip-flops*.

En el caso de las familias de *Altera* el módulo lógico básico, denominado LE, está formado por una *LUT* de 4 entradas, un registro programable, entrada y salida de acarreo, entrada y salida de registro y entradas de control. En el caso de familias más complejas se tiene el módulo lógico ALUT con un máximo de ocho entradas, encargado de realizar funciones combinacionales, aritméticas y de desplazamiento de datos.

Por último, destacar el crecimiento importante del consumo conforme crece el número de bits de salida mostrado en las referencias [34, 35] y [41], llegando hasta el 74% de aumento en el caso de la versión de 64 bits en las tecnologías ASICs y algo menor en los dispositivos FPGAs, 46% en el caso de 64 bits a la salida, si bien

el consumo en estos últimos dispositivos es de un orden superior pues se tienen mW en vez de μW como en las tecnologías ASICs.

Debido a las características intrínsecas de estos dispositivos (reconfigurabilidad), los diseños en FPGA tienen consumos de potencia dinámica mayores que los ASICs. En estudios realizados [47] se sugiere que de media un diseño en un dispositivo FPGA tiene unas 14 veces más consumo dinámico que en un ASIC cuando el circuito contiene solo lógica. Sin embargo cuando el diseño utiliza macro-bloques dentro de la FPGA como memorias, multiplicadores, entre otros, la diferencia entre el consumo dinámico de los dos dispositivos se reduce. Como resultado, se puede concluir que los dispositivos FPGA son menos eficientes en términos de disipación de potencia cuando se comparan con los ASICs. Para el caso del consumo debido a las corrientes de fugas, también los dispositivos FPGAs tienen una disipación de potencia que va a ser mayor respecto al del ASIC debido a que para soportar la reconfigurabilidad los dispositivos FPGAs tienen más transistores para llevar a cabo la misma funcionalidad. Según estudios realizados [47] se tiene un factor de 5,4 veces con respecto al del ASIC.

Es difícil extraer conclusiones a partir de los datos publicados bien porque no existen [37, 44-46], bien por las diferentes frecuencias, tensión de alimentación o tecnologías utilizadas. Nuestras aportaciones a los resultados de consumo se harán en dispositivos de *Xilinx*, donde no se han encontrado referencias con datos del consumo de potencia y en varias tecnologías ASIC.

2.3. Disipación de potencia en circuitos digitales CMOS

Existe bastante documentación tanto en libros [47-59], como en artículos [60-69], relativa al consumo de potencia en circuitos digitales CMOS, sus causas y posibles métodos de reducción. Este apartado pretende únicamente dar una visión general de los conceptos básicos en el consumo de potencia de los diseños digitales CMOS, por lo que solo se van a presentar las ideas más necesarias. Solamente en el caso del consumo de potencia dinámico se hace un análisis más detallado ya que es la componente dominante y sobre la que se va a centrar el estudio de la reducción en el consumo de potencia de los cifradores Trivium.

La disipación de potencia en circuitos digitales tiene dos componentes fundamentales, una componente de potencia dinámica y otra estática. La disipación de potencia dinámica se produce cuando los datos están conmutando, es decir, cuando el circuito está realizando alguna operación con dichos datos. La disipación de potencia estática sin embargo ocurre cuando el circuito está inactivo, donde los datos se mantienen en un valor constante, sin cambios.

Estas dos componentes fundamentales de disipación de potencia en circuitos

digitales CMOS son debidas principalmente a:

- Consumo de potencia dinámica
  - Debida a la carga y descarga de las capacidades del circuito (P_{SWT}), también denominada de conmutación.
  - Debida a la corriente de cortocircuito de los transistores (P_{SC}).
- Consumo de potencia estático debido a las corrientes de fugas (*leakage*) cuando se aplica tensión al dispositivo (P_{fugas}).

El consumo de potencia total en cualquier circuito digital se puede expresar por la siguiente ecuación:

$$P_{total} = P_{dinámica} + P_{estática} = P_{SC} + P_{SWT} + P_{fugas} \quad (2.1)$$

A continuación se va a analizar cada una de estas componentes del consumo de potencia.

2.3.1. Consumo de potencia dinámico.

Como se recoge en la ecuación (2.1) la potencia dinámica $P_{dinámica}$ es la suma de la de cortocircuito (P_{SC}) y la de conmutación (que aquí representaremos como P_{SWT}). Esta división corresponde a las causas del consumo. Sin embargo en las herramientas de diseño lógico se contemplan desde otra perspectiva, en particular, considerando dónde se consume, si es en el interior o en el exterior de la celda. La parte interna (P_{INT}) recoge todo el consumo en el interior de la celda y corresponde al consumo debido al cortocircuito (P_{SC}) y al consumo debido a la conmutación en las capacidades parásitas que aparecen entre los terminales de los transistores de la celda. Esto es, P_{INT} incluye una parte de la potencia de conmutación total, P_{SWT}, ya que incorpora la potencia de conmutación consumida en los nodos internos de las celdas digitales. Todos estos consumos ligados a P_{INT} son caracterizados por el fabricante de la celda, que los incorpora en el modelo tecnológico de la celda, y no dependen de cómo se va a utilizar en el circuito. La parte externa es debida a la conmutación de la capacidad de carga de la celda. En la capacidad de carga se tiene en cuenta tanto las capacidades de entrada de las celdas a las que se conecta como las de las líneas de interconexión. Como es un consumo debido a la carga y descarga de condensadores durante su conmutación también es denominada potencia de conmutación P_{SW} y se diferencia de P_{SWT} en que no incluye las aportaciones de las capacidades internas de las celdas. Esta visión es también recogida en la ecuación (2.2), en su última expresión.

$$P_{dinámica} = P_{SC} + P_{SWT} = P_{INT} + P_{SW} \quad (2.2)$$

Con más precisión, la estimación de potencia dinámica suma, para todas las celdas, las dos contribuciones -potencia interna (P_{INT}) y potencia de conmutación (P_{SW} o

externa)- tal como recoge la ecuación (2.3), donde el índice *m=1,…M* se refiere a todas las celdas digitales instanciadas en el circuito y *n=1,…N* a todas las interconexiones.

$$P_{dinámica} = \sum_{m=1}^{M} P_{INT\ m} + \sum_{n=1}^{N} P_{SW\ n} \tag{2.3}$$

El valor de cada $P_{INT\ m}$ se toma de los datos tecnológicos suministrados por el fabricante para esa celda. Por otra parte, la capacidad externa es computada de las capacidades de los pines de entrada conectados a la salida de la celda (*fanout*) y las capacidades de las interconexiones, todas ellas calculadas de los modelos dados por el fabricante en la librería tecnológica.

La potencia dinámica de conmutación consumida en una puerta cuya tensión de alimentación es V_{DD}, cuya capacidad equivalente de carga/descarga es C_L, que opera a una frecuencia media f_{CLK} y que tiene un factor de actividad de conmutación *a*, es la indicada en la ecuación (2.4) [49]:

$$P_{SW} = a f_{CLK} V_{DD}^2 C_L \tag{2.4}$$

Esta es la expresión en la que se basan las herramientas digitales para estimar el consumo externo de cada celda $P_{SW\ n}$, donde la alimentación y frecuencia (V_{DD} y f_{CLK}) son parámetros generales de operación del circuito, la capacidad de carga es propia del conexionado de cada celda ($C_{L\ n}$) y la actividad de conmutación ([58], [65], [70]) se particulariza a cada celda según los resultados obtenidos mediante simulación (a_n). Así, resulta la expresión de P_{SW} dada en la ecuación (2.5):

$$P_{sw} = f_{CLK} V_{DD}^2 \sum_{n=1}^{N} a_n\ C_{L\ n} \tag{2.5}$$

Una expresión análoga (ecuación (2.6)) es utilizada para estimar la potencia interna debida a la conmutación, P_{INT_SW}. La principal diferencia con la anterior es que las capacidades son propias de la celda y han sido caracterizadas y modeladas por el fabricante y su modelo está incluido en la librería de celdas que proporciona la tecnología.

$$P_{INT\_SW} = f_{CLK} V_{DD}^2 \sum_{m=1}^{M} a_m\ C_{int\ m} \tag{2.6}$$

De la ecuación 2.5 y 2.6 se pueden extraer todos los factores que influyen en el cálculo del consumo dinámico causado por la carga y descarga de las capacidades internas y externas. Se pueden enumerar como:

- Tensión de la fuente de alimentación. Su dependencia es cuadrática.
- La capacidad de carga de los nodos del circuito y las capacidades internas.
- La frecuencia de reloj en circuitos síncronos.
- La actividad de conmutación.
- El número de nodos.

La otra componente de P_{INT} es debida al consumo de cortocircuito, P_{INT_SC}. Una expresión aceptable para la potencia de cortocircuito [68, 71] de un inversor CMOS

es dada por la ecuación (2.7):

$$P_{SC} = \frac{K'(\frac{W}{L})}{12}(V_{DD} - 2V_t)^3 \tau_{in} a f_{CLK} \quad (2.7)$$

Donde *K* se define como el factor de ganancia y τ el tiempo de subida o bajada (se consideran iguales) del transistor.

Puede observarse que P_{SC} depende de la implementación microelectrónica concreta y de la tecnología de integración de la celda (es función de *K, W/L, Vt*), de la pendiente de la señal de entrada (τ_{in}), de la frecuencia de operación f_{CLK} y de la actividad de conmutación *a*. Como es obligado, para el circuito completo habrá que contabilizar las potencias de cortocircuito de todas las celdas (ecuación. 2.8):

$$P_{INT\_SC} = \sum_{m=1}^{M} P_{SC\ m} \quad (2.8)$$

De forma similar a las otras componentes, se tiene incluido el modelo de consumo de la celda implementada en la librería tecnológica lo cual, junto a los datos de operación (V_{DD}, f_{CLK}) y de los patrones de test (*a*, τ_{in}), permite completar la estimación de P_{INT}.

Con la miniaturización de la tecnología los transistores van siendo más pequeños y por tanto también su tensión de alimentación así como la tensión umbral. Esto hace que el consumo por cortocircuito sea menos significativo que el debido al consumo dinámico. Su contribución aproximada se puede acercar al 5-10% [59] o incluso el 20% en el caso de circuitos de baja potencia y alta frecuencia al trabajar con transistores de baja tensión umbral.

2.3.2. Consumo de potencia estático.

Los componentes de la potencia estática son importantes cuando los circuitos están en reposo, es decir sus entradas y valores internos mantienen valores de tensión constantes. Por ejemplo cuando no hay actividad en los circuitos aunque estén polarizados.

El consumo de potencia estática es debido a la corriente de fugas, que tiene tres componentes: la corriente de las uniones *pn* polarizadas en inversa (formadas por los drenadores/surtidores y los pozos/sustratos), la corriente sub-umbral de los transistores y por la corriente a través del óxido de puerta por efecto túnel.

Estas corrientes crecen conformen disminuye las dimensiones geométricas de los transistores (profundidad del óxido, longitud de la puerta, etc) y comienzan a hacerse significativas en las tecnologías fuertemente submicrónicas. Por tanto, la miniaturización de la escala de los dispositivos CMOS ha hecho que este término tenga cada vez más importancia [56] y que no pueda ser considerado despreciable como anteriormente ocurría en las escalas micrométricas [57]. Recientes trabajos

sugieren que la potencia estática constituirá el 50% del total del consumo en tecnologías de 65 nm y menores [47].

El consumo estático de un circuito depende fundamentalmente del número de transistores que tenga el circuito y del tamaño de puerta de la tecnología y no de las conmutaciones de sus transistores.

En esta Tesis se va a trabajar con tecnologías por encima o iguales a 90 nm por lo que el término de las corrientes de fugas no va a ser tenido en cuenta.

2.4. Reducción del consumo de potencia en circuitos digitales

La reducción de los tamaños de las tecnologías y el auge de los dispositivos alimentados por baterías con comunicaciones inalámbricas (*WiFi* ó *Bluetooth*) han favorecido la necesidad de introducir técnicas de reducción de potencia en circuitos CMOS micro y nanométricos en sus diferentes etapas del proceso de diseño.

Algunas de estas técnicas se basan en la dependencia que tiene la potencia con los parámetros tecnológicos y otras son inherentes al diseño (capacidad, polarización, frecuencia, lógica empleada, etc). La tecnología y el diseño son parámetros que afectan a todas las componentes del consumo de potencia [57] como se muestra en la Tabla 2.5:

Tabla 2.5 Contribución y dependencia del consumo de potencia.

Contribución	Dependencia
P_{swT}	Tecnología, Diseño.
P_{sc}	Diseño.
P_{fugas}	Tecnología, Diseño.

Hacer un análisis exhaustivo de las diferentes técnicas sería imposible. Solamente se presenta un breve resumen de algunas de las más representativas entre las dirigidas a las técnicas de diseño para la reducción del consumo [47-59].

Los siguientes apartados discuten medidas de diseño y de parámetros del modelo tecnológico de la librería de celdas, que pueden reducir los diferentes consumos de potencia.

2.4.1. Opciones de diseño para reducir el consumo

Desde el punto de vista del parámetro que se ve afectado, las técnicas de reducción del consumo de potencia se pueden agrupar en las siguientes categorías.

- Actuación sobre la tensión.
  - Escalado de tensión y reducción de la tensión umbral.
  - Inhabilitación de la alimentación *(power gating)*.
- Actuación sobre el reloj y/o su frecuencia.
  - Disminución de la frecuencia de reloj.
  - Inhabilitación del reloj (*clock gating)*.
  - Estilo de la lógica.
- Reducción actividad de conmutación.
  - Inhabilitación del reloj (*clock gating)*.
  - Estilo de la lógica.
  - Inhabilitación de los datos (*operand isolation*).
- Reducción de las capacidades.
  - Estilo de la lógica.
  - A nivel de implementación física (*layout*).

2.4.1.1. Escalado de tensión y reducción de la tensión de umbral

La reducción de la tensión de polarización del circuito es una solución atractiva para reducir el consumo de potencia puesto que la potencia de conmutación y la de cortocircuito tienen una dependencia cuadrática o cúbica con V_{DD}. Sin embargo la reducción de la tensión de polarización conlleva la penalización en la respuesta temporal del circuito. La penalización por retraso puede mitigarse reduciendo la tensión umbral de los transistores aunque entonces las corrientes de fugas (sub-umbral) se incrementarán exponencialmente [55]. Por tanto es importante seleccionar una apropiada tensión umbral y de alimentación.

Otra manera de mejorar las prestaciones es hacer una arquitectura con escalado de tensión de alimentación y de la tensión umbral. En el interior del circuito se podrían usar dos o más tensiones de alimentación considerando siempre el tiempo que puede llevar al circuito ajustarse a los nuevos valores de tensión. Asimismo en los procesos tecnológicos más actuales, el uso de al menos dos tensiones de sub-umbral es común. Una tensión umbral baja es usada para las partes con retraso crítico y una más alta para el resto. Un compromiso aceptable para el consumo de potencia debido a los retrasos y a las corrientes de fugas sería utilizar transistores pequeños con valores bajos de tensión umbral (Vt) y transistores más grandes con Vt más alta, según sean las necesidades de velocidad de operación.

Otra técnica, denominada *power gating,* añade transistores (normalmente de tensión umbral alta) entre las alimentaciones y ciertas partes del circuito con el objetivo de evitar que la tensión de alimentación llegue a partes del circuito que no son utilizadas durante periodos de tiempo grandes, consiguiendo reducir el consumo de potencia, sobre todo el debido a las corrientes de fugas. También puede haber

efectos negativos en su modo de funcionamiento normal debido a la resistencia extra de los transistores añadidos, que afectan a sus prestaciones temporales (pueden reducirse) o al propio consumo (picos de corriente en las conmutaciones de estos transistores).

2.4.1.2. Actuación sobre el reloj y/o su frecuencia

La reducción de la frecuencia de reloj no es tan beneficiosa como la reducción en tensión. Sin embargo, muchos circuitos digitales de hoy en día tienen diferentes dominios de consumo donde las señales de reloj son silenciadas a los bloques de circuitería que no se están usando en ese momento. Esto también se denomina como *clock gating*.

La técnica de *clock gating* [37] es ampliamente utilizada para reducir el consumo de potencia dinámica en circuitos digitales. El concepto clave consiste en detener el reloj del *flip-flop* cuando se detecta que no hay cambio en los datos afectados por el reloj [63-64], disminuyendo el consumo de energía mediante la reducción de las conmutaciones de la señal de reloj. Así, en lugar de permitir que el reloj conmute continuamente a través de todos los caminos de entrada de reloj de los registros, se va cortando de tal manera que solo llegue a los registros que verdaderamente necesiten cambiar de valor en cada ciclo de reloj. Es necesario añadir algunos nuevos bloques en el diseño para conseguir la parada del reloj, lo que conlleva añadir consumo adicional de potencia. El uso de esta técnica implica una cuidadosa generación de los árboles de reloj durante la síntesis y la implementación física del circuito. Aunque es una técnica madura que se puede llevar a cabo de forma casi automática en el proceso de síntesis por parte de la herramienta CAD usada, su aplicación conlleva en muchos casos un estudio detallado anterior de la arquitectura del sistema y de los bloques a los que podría afectar la aplicación de ésta técnica si la técnica no se usa adecuadamente. La inserción de lógica adicional en la red de distribución de reloj puede traer consigo un aumento en vez de una disminución de la potencia consumida.

2.4.1.3. Reducción de la actividad de conmutación.

La actividad de conmutación es una medida del número de transiciones internas en las señales de un circuito. Este parámetro depende de la secuencia de las señales de entrada y de la operación que realiza el circuito [70]. Además puede contribuir a otros parámetros como la testabilidad y fiabilidad del circuito.

Las técnicas basadas en la reducción de la actividad de conmutación son tan eficientes como las anteriores de reducción de la frecuencia de reloj. Algunas de ellas ya han sido comentadas, como es el caso de la técnica de *clock gating*. Otras opciones son utilizar diferentes estilos de lógica y capacidad de carga como en el caso de las familias lógicas con transistores de paso [50], [61-62]. En *flip-flops* y

registros, las capacidades de los nodos de reloj son muy importantes, ya que la señal de reloj tiene una actividad de conmutación apreciable. Así pues, *flip-flops* con un mínimo número de transistores a los que llegue la señal de reloj han sido también propuestos en la bibliografía [33] y también con diseños optimizados para bajo consumo de potencia [72].

La técnica de *operand isolation* como su nombre indica aísla las señales de entrada de un sub-módulo del circuito para prevenir que las señales se propaguen y generen un consumo indeseado debido al aumento de la actividad de conmutación. Se puede aplicar cuando se tienen circuitos o parte de ellos que no realizan operaciones durante un tiempo determinado con sus entradas o no son necesarias. Se utilizan *latches*, multiplexores u otras celdas para prevenir que las señales de entrada se propaguen.

Otra técnica que se puede aplicar para disminuir la actividad de conmutación es la paralelización [51-55], en la cual se particiona un diseño con frecuencia de reloj *f* en *N* bloques en paralelo con frecuencia *f/N* y un multiplexor controlado por la frecuencia *f*. Cada bloque puede operar a frecuencias N veces menor, disminuyendo por tanto su consumo.

Se aplica sobre todo a memorias, registros de datos en paralelo y registros de desplazamiento. La Figura 2.4 muestra un esquema en el caso de que la paralelización sea doble (*N*=2):

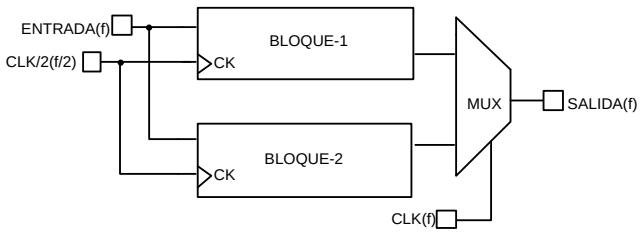


Figura 2.4 Técnica de paralelización para **N**=2.

2.4.1.4. Reducción de la capacidad de carga

Existen diversas técnicas [50-52, 62] para reducir la capacidad de carga en circuitos integrados dependiendo del nivel de abstracción donde se aplique. A nivel lógico y de *layout* la reducción de la capacidad de carga se puede conseguir con el dimensionado de los transistores y la minimización de las interconexiones en metal o la selección de una familia lógica determinada. En cuanto al dimensionado de los transistores hay que tomar ciertas precauciones debido a que dispositivos con dimensiones mínimas pueden no tener balanceados los retrasos de subida y bajada produciendo un consumo mayor de corriente y una degradación de su

funcionamiento.

El uso de lógica dinámica frente a lógica estática reduce también las capacidades parásitas (menos transistores) y el número de transiciones espúreas (*glitches*). Sin embargo, los circuitos construidos con lógica dinámica incrementan su actividad de conmutación debido a la precarga y tienen un consumo debido al reloj. La lógica dinámica está también libre de corrientes de cortocircuito. Alternativas como la lógica con transistores de paso tienen la ventaja de utilizar menos transistores en algunas implementaciones de funciones lógicas reduciendo por tanto la capacidad parásita, pero con penalizaciones, entre otras, como la pérdida de robustez en el diseño.

2.4.2. Relación entre la tecnología y el consumo de potencia

En la mayor parte de los casos el diseño microelectrónico de un circuito digital CMOS sigue una metodología *semi-custom*, estando compuesto por celdas de una librería tecnológica y otras macro celdas como, por ejemplo, memorias y multiplicadores. La medida del consumo de potencia interno y de fugas viene condicionada por los datos del modelo de la celda incluido en la librería tecnológica. Esta información es descrita usando un estándar de modelado para librería tal como *Liberty* [73].

Los modelos de potencia de la librería de celdas son por tanto fundamentales para el cálculo del consumo de potencia y son, en la mayoría de los casos, utilizados junto con simulaciones lógicas del diseño. A continuación vamos a enumerar los principales parámetros que afectan a la estimación y al consumo de potencia de las celdas de librería visto desde un punto de vista interno y desde un punto de vista de las corrientes de fuga [50].

2.4.2.1. Parámetros de dependencia del consumo de potencia interno

El consumo de potencia interno depende de forma cuadrática de la tensión de polarización de la celda, siendo su dependencia cuadrática, aunque también existen otros parámetros tecnológicos (no descritos en el fichero de las celdas de librería) que afectan al consumo de potencia:

a) Tensión umbral *Vt*: Celdas con mayor tensión umbral tienen menor consumo de potencia interna (mejora de P_{sc}). Para una tecnología específica se puede conseguir hasta un 20% de ahorro cambiando a una celda equivalente con una tensión umbral más alta.

b) Esquina (*corner*) del proceso: El consumo interno es mayor para el *corner* del proceso más rápido. Para la misma tensión de polarización y temperatura, la potencia interna en el caso de la esquina del proceso más

rápida puede ser del orden del 10% mayor que las condiciones del proceso nominal.

c) Temperatura: La potencia interna normalmente se incrementa con la temperatura debido a las transiciones más lentas de la señales a temperaturas más altas. El incremento con la temperatura es despreciable para un proceso lento, con celdas con tensión umbral alta y tensión de polarización baja, mientras que es más significativo para un proceso rápido, con celdas con tensión umbral baja y tensiones de polarización altas. Bajo condiciones típicas (proceso típico, tensión umbral estándar, y tensión de polarización nominal), la disipación de potencia interna a altas temperaturas (~125°C) puede ser del orden de un 10% mayor que la potencia interna disipada a temperaturas bajas (~-40°C).

d) Longitud del canal: Como en el caso de la tensión umbral, celdas con dispositivos de canal largo tienen menor consumo de potencia interno que celdas con dispositivos de canal corto.

Además de estos parámetros hay otros en la librería de celdas que también influyen para el consumo de potencia como el *fanout*. Celdas con un alto *fanout* tienen mayor consumo de potencia interna que las de menor *fanout*. Como un diseño usa una gran variedad de tipo de celdas, nos centraremos en los parámetros listados anteriormente para describir la variación del consumo interno para un diseño. Como idea general, para un diseño dado, la potencia interna es en general más alta con el proceso rápido, máxima tensión de polarización y máxima temperatura.

2.4.2.2. Parámetros de dependencia de la corriente de fugas

Los principales parámetros tecnológicos, que afectan a las corrientes de fugas son:

a) Tensión umbral *Vt*: Usando diferentes tensiones umbrales se modifica la corriente sub-umbral. Celdas con mayores tensiones umbrales tienen menores corrientes de fugas pero son más lentas. Similarmente, celdas con valores bajos de tensión umbral tienen valores mayores de corrientes de fugas y son más rápidas. La contribución de la corriente por efecto túnel no cambia significativamente al cambiar la tensión umbral sino al cambiar la profundidad de la capa de óxido.

b) *Fanout*: Celdas con valores elevados de *fanout* tienen una corriente de fugas mayor pero proporcionan mayor velocidad que las celdas con menor *fanout*, cuando se conectan muchas celdas.

c) Temperatura: La corriente sub-umbral del dispositivo MOS tiene una fuerte dependencia no lineal con respecto a la temperatura. En la mayoría de los procesos tecnológicos, la corriente sub-umbral puede multiplicarse

por diez o veinte o incluso más cuando la temperatura sube de 25°C a 125°C. La contribución debida a la corriente por efecto túnel tiene una menor variación con respecto a la temperatura o a la tensión de umbral del dispositivo. A altas temperaturas la corriente sub-umbral sigue siendo la contribución dominante frente a la corriente de fugas.

d) Longitud del canal: La mayoría de las librerías de celdas estándar se diseñan con la mínima longitud de canal que soporta la tecnología. Sin embargo, la corriente de fugas puede reducirse incrementando la longitud de canal del dispositivo MOS. La contrapartida es que los dispositivos con canales más largos tienen peor comportamiento temporal.

e) Proceso tecnológico y *corner*: Aunque la corriente por efecto túnel es despreciable para procesos mayores de 100 nm, tiene un efecto no despreciable para tecnologías de 65 nm y menores. La corriente de fugas también tiene una fuerte dependencia con los *corners* del proceso, siendo mayor para el *corner* rápido frente al típico. Similarmente, la corriente de fugas para el *corner* lento es mucho más pequeña que para el típico.

2.5. Conclusiones

En este capítulo se ha presentado como el uso cada vez más extendido de dispositivos criptográficos en sistemas portables e inalámbricos hace que el consumo de potencia y los recursos lógicos sean un factor crítico del diseño. Esto va a dar lugar a que la aplicación de mecanismos que minimicen el consumo y los recursos en estos sistemas sea un factor muy a tener en cuenta como objetivo de diseño.

Dentro de los algoritmos criptográficos con bajo consumo de recursos se destacan los cifradores de clave simétrica, los cuales se subdividen en cifradores de bloque y de flujo. Se eligen los cifradores flujo frente a los cifradores de bloques por su arquitectura más sencilla para una implementación hardware y su rapidez en aplicaciones que lo precisen. Dentro de éstos se escoge el cifrador Trivium propuesto en la convocatoria eSTREAM por la Red Europea de Excelencia en Criptografía como una buena opción para el diseño de bajo consumo y recursos por su simplicidad, seguridad y velocidad.

El cifrador de flujo Trivium permite realizar implementaciones que generan varios bits en su salida. Estas implementaciones han sido evaluadas en recursos consumidos y potencia a través de las referencias de la bibliografía con diferentes tecnologías y dispositivos, haciendo constar la escasez de referencias al empleo de técnicas de reducción de consumo de potencia en estos cifradores.

En este capítulo se ha hecho una breve presentación de los mecanismos de disipación de potencia que afectan a los circuitos digitales CMOS y los principios claves para su reducción. Se ha presentado también como el consumo de potencia dinámico es la componente dominante en este tipo de circuitos, siendo los factores que más influyen en este consumo la tensión de polarización de los dispositivos, la capacidad de carga de los nodos del circuito, las capacidades internas, la frecuencia de reloj, la actividad de conmutación y el número de nodos. Estos parámetros van a venir determinados por la tecnología y las técnicas de diseño empleadas. En cuanto a las técnicas de diseño para reducir estos parámetros se han citado el escalado de la tensión de alimentación, el control sobre el reloj y su frecuencia, el estilo de la lógica, la inhabilitación de los datos o el estilo de la implementación física. Mientras que para las opciones tecnológicas se tiene el escalado de la tensión umbral de las celdas de librería, el uso de diferentes esquinas del proceso tecnológico, la temperatura, el *fanout* de las celdas de librería y la elección de diferentes procesos tecnológicos.

Finalmente nuestro estudio se va a centrar en mejorar el consumo dinámico mediante la aplicación de técnicas a nivel de diseño que son válidas en tecnologías ASIC y FPGA, a las que se pueden sumar técnicas tecnológicas. Estas técnicas se van a aplicar para proponen arquitecturas eficientes en consumo de potencia para los cifradores de flujo Trivium que se verá en el próximo capítulo.

3. DISEÑO DE CIFRADORES TRIVIUM DE BAJO CONSUMO

El uso cada vez más extendido de dispositivos criptográficos en sistemas portables e inalámbricos [74, 75] donde el consumo de potencia es un factor crítico, da lugar a que la aplicación de mecanismos que minimicen el consumo en estos sistemas sea un factor muy a tener en cuenta como objetivo de diseño. En el Capítulo anterior seleccionamos el cifrador de flujo Trivium como un algoritmo criptográfico de bajo consumo de recursos muy adecuado para este tipo de sistemas. En este Capítulo se abordan las técnicas más apropiadas para minimizar el consumo de potencia en el diseño de cifradores Trivium, centrándonos en el caso de salida de secuencia cifrante de un solo bit, que es el más referenciado, aunque también se estudiarán los casos multi-bit.

En cuanto a técnicas para la reducción de potencia, este trabajo aporta soluciones para mejorar el consumo dinámico de los cifradores de flujo Trivium a nivel lógico. Como se justificará en el siguiente apartado, la técnica de paralelización [55, 66, 76], se presenta como la mejor opción ya que la arquitectura interna de éste, basada en registros de desplazamiento, es muy adecuada para la aplicación de dicha técnica.

En el primer apartado se estudia como ejemplo base, el consumo de potencia en registros de desplazamiento (simple y paralelizado) con simulaciones eléctricas y a nivel lógico. El segundo apartado presenta la implementación de los cifradores Trivium de bajo consumo de un bit y con salida multi-bit. En el tercer apartado se hace un análisis de área y consumo de potencia en los cifradores utilizando una tecnología ASIC. El cuarto apartado analiza los cifradores en tres tecnologías de fabricación submicrónicas. Y por último en el quinto se implementan los cifradores de un solo bit en dispositivos programables FPGA, mostrando datos de los recursos utilizados y su consumo de potencia.

3.1. Cálculo del consumo en registros de desplazamiento

Los cifradores de flujo Trivium, que serán presentados en el apartado 3.2, están compuestos por algo menos de trescientos biestables y en torno a la decena de puertas de las que la mayor parte son XOR. Los biestables están conectados formando un registro de desplazamiento no lineal que actúa como registro de estado [31]. De aquí que la práctica totalidad del consumo se produzca en este registro y, por tanto, las técnicas de reducción de consumo a emplear son las que resultan más eficientes para los biestables. En este trabajo los biestables (y las puertas lógicas) a utilizar son los que suministran las *foundries* en sus librerías de celdas o los dispositivos programables FPGA, por lo que el esfuerzo de diseño para bajo consumo se centra en el diseño lógico del cifrador Trivium y no en el de las celdas.

Las dos técnicas que mejor se adaptan para reducir el consumo en biestables son las de paralelización [51, 55, 62] y la de *clock gating* [63, 64, 77]. Ambas técnicas fueron comentadas en el capítulo anterior dentro del apartado de la reducción de la actividad de conmutación. El concepto clave de la técnica de *clock gating* consiste en detener el reloj de un biestable (o grupo de biestables) cuando éste (éstos) no ha (han) de cambiar su actual estado almacenado. Así, en lugar de permitir que el reloj conmute continuamente a través de todos los caminos y en todas las entradas de reloj de los biestables, se anula la señal de reloj que excita a los biestables que no cambian de estado y solo se deja pasar a los registros que verdaderamente necesiten cambiar de valor en ese ciclo de reloj. Con ello se disminuye el consumo de energía dinámica ya que se eliminan las conmutaciones "improductivas" de la señal de reloj y de los correspondientes biestables al mismo tiempo que se mantiene la funcionalidad del circuito.

La aplicación de esta técnica no tiene un coste cero ya que es necesario añadir cierto número de componentes que incrementan el área ocupada y, por tanto, el número de dispositivos que consumen energía, pudiendo afectar negativamente también a la frecuencia máxima de operación. Los nuevos componentes son necesarios en dos sentidos: uno, para incorporar el circuito que detiene o deja pasar la señal de reloj, el cual debe ser cuidadosamente diseñado para no causar problemas temporales (tipo *glitch* o de pérdida de la sincronización), y, otro, para determinar cuándo se tiene que detener y cuándo dejar pasar dicha señal, función que depende de la operación del circuito y de los biestables a los que se desee detener el reloj.

Como regla general, la aplicación de la técnica de *clock gating* se hace más eficaz para los circuitos, o bloques de circuitos, cuyos registros no tienen actividad de conmutación durante períodos de tiempo significativos. Así, por ejemplo, en [51] se estudia cómo aplicarlo a los contadores para reducir el ruido de conmutación (junto al consumo).

Se muestra en esa Tesis que dicha técnica es más eficaz en los contadores cuando se aplica a un buen número de los bits más significativos de cuenta, porque estos bits más significativos se mantienen sin cambios mientras que los (n) bits menos significativos realizan un ciclo completo de cuenta (2^n ciclos de reloj). Así, realizar la función que controla el *clock gating* es fácil (simple y de poco coste), el reloj es detenido para un número significativo de biestables y durante un tiempo suficientemente grande, características todas ellas que hacen que las ventajas que se obtienen al parar el reloj superen los costes introducidos con los nuevos componentes añadidos. Cuando se consigue dicha ventaja resulta eficiente la aplicación de esta técnica. Sin embargo, en nuestro caso, la técnica no se considera eficiente debido a que en los cifradores de flujo los biestables están operando continuamente ya que su función es la de un registro de desplazamiento, con lo que no es fácil encontrar grupos de bits que 'operen' menos tiempo que los demás. Además, los datos que se introducen en los biestables son pseudoaleatorios y dependen de la clave y del vector de inicialización y no de la propia estructura del circuito, lo que hace compleja la función de detección de cuándo parar el reloj. De esta forma, la inclusión de lógica adicional para realizar el adecuado control del reloj penalizará finalmente el consumo, además de que puede provocar problemas de temporización, por lo que podemos concluir que esta técnica no es eficiente para el diseño de Trivium de bajo consumo.

De acuerdo con el análisis anterior, en esta Tesis se ha elegido la paralelización como la técnica óptima de reducción de consumo para aplicar en los cifradores Trivium.

Por otra parte, un aspecto previo pero muy relevante para investigar diseños de Trivium de bajo consumo es la validación de los procedimientos de medida de potencia que se han de utilizar. Las medidas más fiables se obtendrán mediante simulación. Se dispone de dos formas de medir el consumo que corresponden a las simulaciones de carácter eléctrico y las de carácter lógico.

En el nivel lógico, en el que se ubica este trabajo, el consumo se medirá preferentemente mediante simulaciones de carácter lógico, ya que éstas son mucho más rápidas que las eléctricas con lo que permiten simular circuitos más complejos en tiempos razonables. El análisis de potencia a nivel lógico requiere conocer los modelos de consumo "internos" de las celdas usadas, que los diseñadores tienen disponibles en todas las tecnologías, junto a la topología del circuito y la consiguiente determinación de cargas "externas" a cada celda. Esta información, junto a la actividad de conmutación causada con la operación del circuito, permite estimar el consumo de potencia.

Sin embargo, las medidas proporcionadas por las simulaciones lógicas tienen un menor grado de exactitud que las obtenidas mediante simulaciones de carácter eléctrico. La simulación eléctrica se realiza a nivel de transistor y conduce a

resultados mucho más próximos al consumo real. De hecho, puede dar un valor muy preciso del consumo total puesto que se puede conocer la tensión y la intensidad de corriente de la fuente en todo momento. Para llevar a cabo simulaciones eléctricas hay que disponer de una tecnología que proporcione al diseñador los modelos a nivel de transistor de las celdas de librería estándar. No todas las tecnologías accesibles por nosotros (que lo hacemos a través de *Europractice*) proporcionan estos datos, lo cual impide optar por la simulación eléctrica como método general de medida.

Una de las tecnologías disponibles en nuestro entorno es AMS 350 nm la cual, aunque ya no está entre las tecnologías más avanzadas, todavía mantiene plenamente su utilidad y proporciona los modelos lógico y eléctrico de las celdas, lo que nos permite hacer el doble estudio del consumo de los diseños. Utilizando esa tecnología, en este apartado vamos a medir el consumo de potencia de un registro de desplazamiento de 8 bits, similar al registro nuclear del Trivium, para así tener un conocimiento profundo de su consumo y de las relaciones entre los datos obtenidos a partir de simulaciones lógicas y eléctricas.

Como primer paso, consideremos el elemento básico más utilizado para la realización hardware de los registros de desplazamiento que es el *flip-flop* o biestable tipo D activo por flanco. En AMS de 350 nm este *flip-flop* comprende un biestable asíncrono (*latch*) maestro en cascada con un biestable asíncrono (*latch*) esclavo. Se ha analizado el consumo y picos de corriente mediante simulaciones eléctricas con *Spectre* en *Cadence* usando valores típicos de la tecnología de 350 nm, una frecuencia de reloj de 25 MHz y polarización V_{DD} de 3,3 V.

Los dos *latches* están activos durante las fases de reloj opuestas, lo que significa que los biestables tienen picos de corriente en ambos flancos de reloj. En la Figura 3.1 se muestran las formas de onda de la simulación eléctrica, en concreto, la entrada de reloj (*CK*), la entrada de datos (*D*) y la salida de datos del biestable (*DOUT*) junto con la corriente de la fuente de polarización *Idd*.

El valor de los picos de *Idd* depende de si hay o no cambios en el valor de estado almacenado y, por tanto, en la salida de datos. Cuando la entrada no cambia se obtienen picos de corriente de aproximadamente 120 μA, tanto si permanece al valor lógico cero como si es a uno, mientras que si existen cambios los picos de corriente aumentan hasta aproximadamente 400 μA.

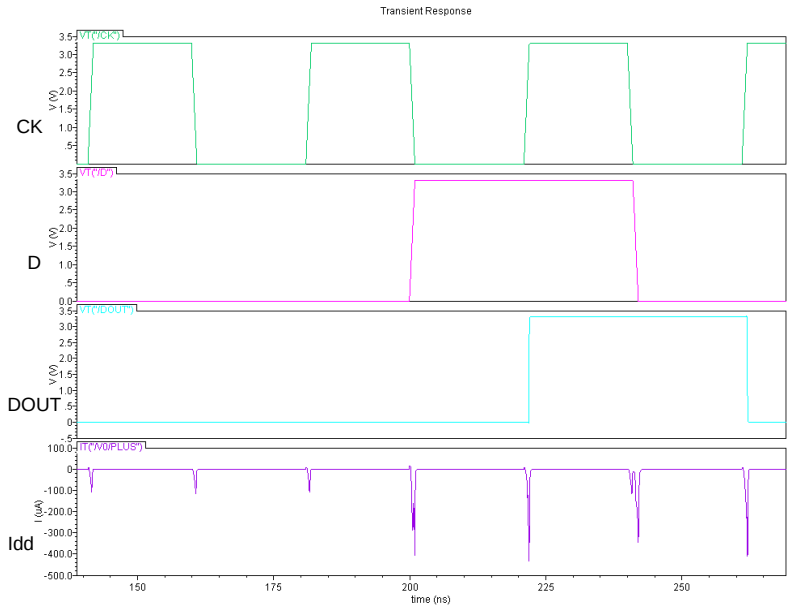


Figura 3.1 Simulación eléctrica de un flip-flop tipo D mostrando los picos de corriente en la fuente de polarización producidos con los flancos de reloj.

3.1.1. Consumo de potencia con simulaciones eléctricas

El consumo de potencia depende de los cambios en la lógica interna que se generen con los cambios de las entradas. En este apartado se presentan los resultados del análisis del consumo de potencia de un registro de desplazamiento de ocho bits "en el escenario de peor caso", esto es, con un conjunto de patrones que maximiza el número de cambios, estando por tanto los biestables cambiando su estado en cada ciclo de reloj.

El diseño del registro de desplazamiento de ocho bits con biestables síncronos tipo D disparado con flanco de subida y con entrada de habilitación de reloj (celda *DEFC1* de la librería tecnológica AMS 350 nm) a nivel de esquemático realizado con la herramienta DFWII de *Cadence* se muestra en la Figura 3.2.

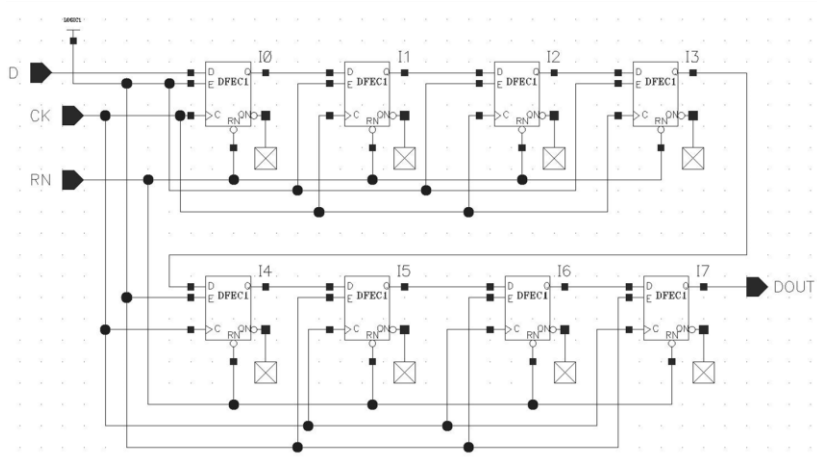


Figura 3.2 Esquemático diseñado con la herramienta DFWII de un registro de desplazamiento de 8 bits.

Se ha analizado el consumo de potencia de ese registro mediante simulaciones eléctricas con *Spectre* en *Cadence* usando valores típicos de la tecnología, una frecuencia de reloj de 25 MHz y polarización V_{DD} de 3,3 V. La Figura 3.4 muestra las formas de onda de la simulación eléctrica donde se muestra la entrada de reloj (*CK*), la entrada de datos (*D*) y la salida de datos del registro de desplazamiento (*DOUT*) junto con la corriente de la fuente de polarización I_{dd}.

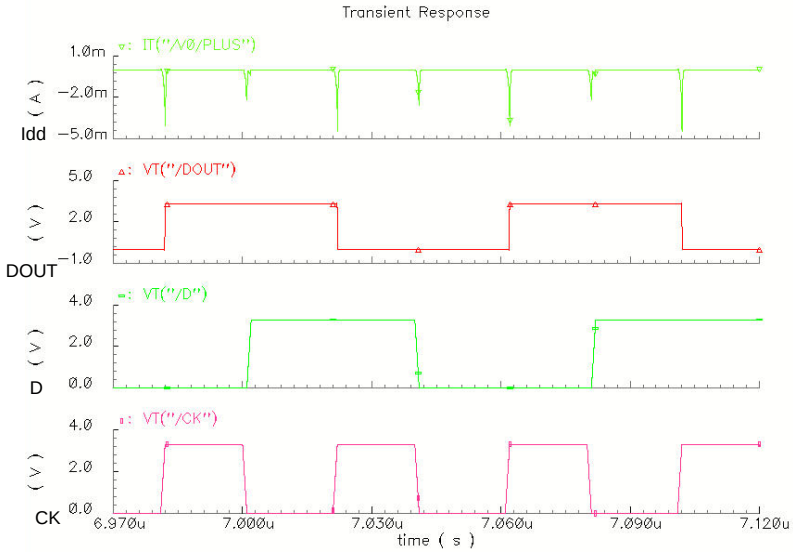


Figura 3.3 Formas de onda y consumo de corriente en el registro de desplazamiento de 8 bits con una simulación eléctrica.

Existe un pico de consumo de potencia con ambos flancos de reloj y cada vez que hay un cambio en el dato del biestable, siendo los picos de corriente mayores en el flanco de subida (4,45 mA) que en los de bajada (2,1 mA). En el escenario de peor caso, con los biestables conmutando cada ciclo de reloj se tiene una corriente promedio $I_{dd}(avg)$ de 62,97 µA. Por tanto, el consumo de potencia promedio vendrá como:

$$P_{avg} \sim I_{dd}(avg) * V_{DD} = 62{,}97\ \mu A * 3{,}3\ V = 207{,}8\ \mu W \tag{3.1}$$

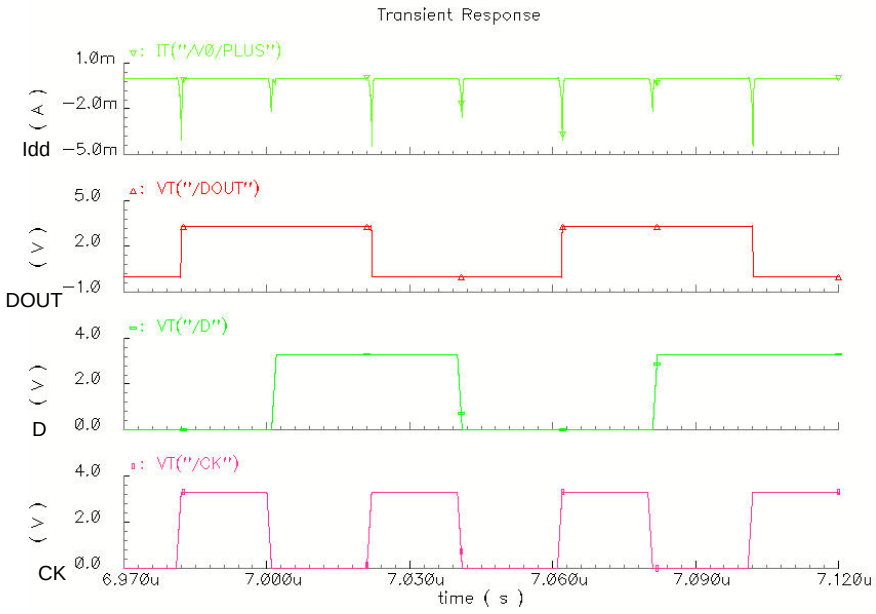


Figura 3.4 Formas de onda y consumo de corriente en el registro de desplazamiento de 8 bits con una simulación eléctrica.

A este registro de ocho bits se le ha aplicado la técnica de paralelización, la cual divide en dos registros el registro de desplazamiento. Cada nuevo registro tiene la mitad de bits que el original y operará a una frecuencia mitad, aunque manteniendo la velocidad del flujo de datos. Durante la operación de desplazamiento, los bits que van llegando en los ciclos pares son almacenados en un registro y los que llegan en los ciclos impares se almacenan en el otro registro. Llamamos por tanto al primero de los registros como par y al segundo como impar.

La Figura 3.5 muestra un esquemático de este registro de desplazamiento paralelizado, obtenido también con DFWII. Se debe añadir alguna circuitería adicional como es un biestable para generar el reloj de frecuencia mitad y un multiplexor a la salida para seleccionar el dato de salida del registro par o impar. El registro impar se sincroniza con el flanco de subida del reloj de frecuencia mitad,

mientras que el registro par se sincroniza con el flanco de bajada. La salida se elige de entre los dos bits más significativos de los registros de desplazamiento con un multiplexor que va conmutando con la señal de reloj.

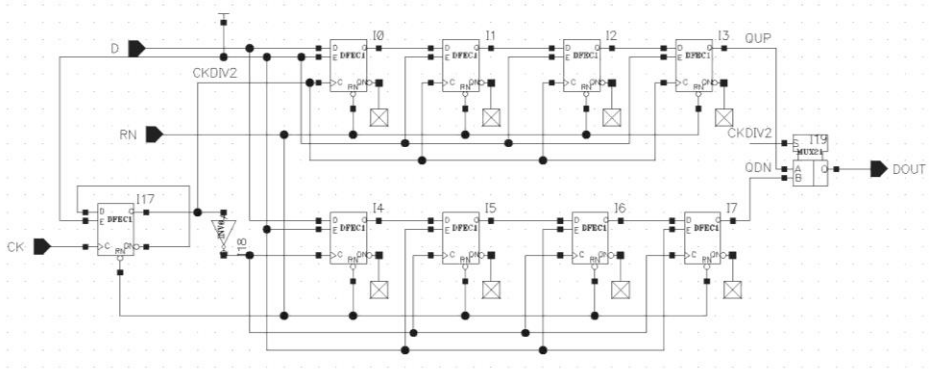


Figura 3.5 Esquemático diseñado con la herramienta DFWII de un registro paralelizado de desplazamiento de 8 bits.

Como ya se ha comentado en el apartado anterior, para obtener el peor caso (máximo consumo de potencia) se tiene que producir un cambio en todos los biestables del registro de desplazamiento en cada ciclo de reloj del biestable. Para conseguir esto se necesita un conjunto de patrones de entrada que vayan cambiando cada dos ciclos del reloj de entrada. Esto hará que todos los biestables de los registros paralelizados de desplazamiento cambien en cada ciclo de su reloj y produzcan un consumo máximo. Si se utilizaran los patrones del registro de desplazamiento estándar los biestables no cambiarían, ya que uno de los registros se cargaría con 0´s y el otro con 1´s y no se alcanzaría el valor máximo de consumo.

Se ha analizado el consumo de potencia de este registro mediante simulaciones eléctricas con *Spectre* en *Cadence*. La Figura 3.6 muestra las formas de onda generadas tras una simulación eléctrica con el simulador. Se muestra la señal de entrada de datos (*D*) cambiando cada ciclo de reloj (*CKDIV2*), la señal de salida (*DOUT*), el reloj de entrada (*CK*), el reloj divido por dos que ataca a los biestables del registro de desplazamiento (*CKDIV2*) y la corriente que circula por la fuente de polarización (I_{dd}).

Aplicando el conjunto de patrones de máximo consumo, sobre un intervalo de tiempo T suficientemente amplio (en nuestro caso de 8 ms), con una polarización de 3,3 V, se mide una corriente promedio en la fuente de polarización de 39,9 µA y se obtiene un valor de consumo de potencia promedio de 131,7 µW.

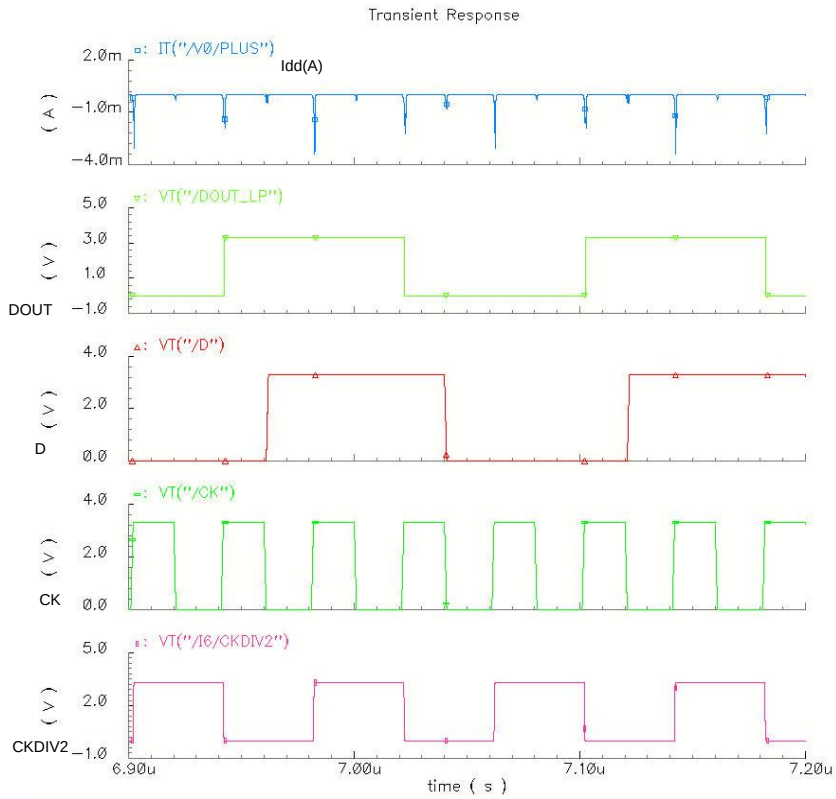


Figura 3.6 Formas de onda y consumo de corriente en el registro de desplazamiento paralelo de 8 bits.

Los resultados de potencia para estos patrones de peor caso son presentados en la Tabla 3.1, en la que se ha incluido, además, la columna "Mejora" para apreciar el porcentaje de ahorro del coste de la versión paralelizada frente al coste de la versión estándar. Se aprecia cómo se reduce en un 36% el consumo en el registro paralelizado frente al estándar. Los picos de corriente también se reducen respecto a la versión estándar como consecuencia de la disminución en ambos flancos de reloj de los picos de corriente. Por ejemplo el pico de corriente máximo en el flanco de subida se reduce en un 24% y en el de bajada lo hace en un 81%.

Tabla 3.1 Datos de potencia y corriente con simulaciones eléctricas y con patrones de caso peor.

Simulador Spectre	**Registro Standard**	**Registro Paralelizado**	**Mejora***
Potencia media	207,8 µW	131,7 µW	36%
Pico max. corriente	4,45 mA	3,38 mA	24%
Pico flanco bajada	2,1 mA	0,4 mA	81%

*$(Mejora = \%(P_{estandar} - P_{paralelizado})/P_{estandar})$.

Si se utilizan los patrones de peor caso para el registro de desplazamiento estándar, la reducción en el consumo es mayor ya que dichos patrones no generan el caso peor de consumo en el registro paralelizado pues no todos los biestables conmutan en cada ciclo de reloj.

Los resultados de potencia con estos patrones se presentan en la Tabla 3.2 donde también se ha incluido además, la columna "Mejora". El consumo del registro paralelizado es de 76,23 μW lo que supone una mayor reducción en el consumo, llegando al 63% frente al 36% del escenario del caso peor.

Tabla 3.2 Datos de potencia y corriente con simulaciones eléctricas y patrones utilizados en la simulación del registro de desplazamiento estándar.

Simulador Spectre	Registro Standard	Registro Paralelizado	Mejora*
Potencia media	207,8 μW	76,23 μW	63%
Pico max. corriente	4,45 mA	1,37 mA	69%

*$(Mejora = \%(P_{estandar} - P_{paralelizado})/P_{estandar})$.

Estos resultados junto con los de la Tabla 3.1 nos dan valores entre 30-60% de mejora en el consumo de los registros de desplazamiento de 8 bits, siendo también del mismo orden y muy apreciables la reducción de los picos de corriente con el flanco de subida y bajada del reloj.

3.1.2. Consumo de potencia con simulaciones a nivel lógico.

Este apartado presenta y compara los resultados del consumo de potencia obtenidos para el mismo registro de desplazamiento de ocho bits en su forma estándar y aplicando la técnica de paralelización, pero ahora las medidas han sido realizadas a nivel lógico. En este nivel la medida de la potencia dinámica por simulación maneja la actividad producida por los patrones de entradas y el modelo del consumo del circuito, dividido en dos partes, las de las celdas usadas y las cargas de cada celda obtenidas tras la configuración del circuito que incorporan las conexiones y el *fanout*.

El consumo de potencia se ha realizado con la herramienta de *Synopsys* usando un fichero de actividad de conmutación en formato saif (*switching activity interchange format*). Este fichero se ha generado mediante simulaciones lógicas con el simulador *Modelsim* de *Mentor Graphics*, utilizando un reloj de la misma frecuencia que en las simulaciones eléctricas (25 MHz). Los modelos de capacidad y consumo de las celdas de librería e interconexiones se toman de la propia librería tecnológica proporcionada por el propio fabricante.

Se ha analizado el consumo de potencia de los registros de desplazamiento

estándar y paralelizado utilizando los mismos patrones que en las simulaciones eléctricas. Los resultados del informe de potencia dado por *Synopsys* se muestran en la Tabla 3.3 junto con el porcentaje de mejora. Puede observarse que estos resultados siguen las mismas pautas que los obtenidos con la simulación eléctrica, aunque los valores concretos difieren.

En efecto, los datos resultantes en el caso lógico para el caso de peor consumo indican que la potencia dinámica del registro de desplazamiento con la técnica de paralelización disminuye entre un 24% respecto al estándar, pauta igual a la mostrada con la simulación eléctrica aunque con una mejora mayor (36%). El informe de *Synopsys* nos permite saber que esto es debido principalmente a que la potencia interna es reducida en el caso paralelo, en un 33% aproximadamente. Sin embargo el consumo debido a las conmutaciones en las interconexiones aumenta un 58% como consecuencia del ligero incremento en el número de interconexiones. Por otra parte y aunque no es objeto de estudio en esta Tesis, el informe indica que también aumenta ligeramente la corriente de fugas (cuyo valor, en todo caso es 4 órdenes de magnitud inferior al del consumo dinámico). Estos resultados son plenamente coherentes con la teoría del consumo, centrando su reducción en las celdas que operan a mitad de frecuencia (potencia interna) y no en la topología del circuito (potencia externa).

Tabla 3.3 Informe de los consumos de potencia en *Synopsys* con simulaciones a nivel lógico.

Potencia @25MHz V=3.3V	**Registro Estándar**	**Registro Paralelizado Caso peor**	**Mejora***	**Registro Paralelizado**	**Mejora***
Interna (celdas)	161 μW	107,1 μW	33,4%	78,4 μW	51,3%
Conmutación (interconexiones)	19 μW	30,1 μW	-58,4%	21,2 μW	-11,5%
Dinámica (total)	180 μW	137,2 μW	23,7%	99,6 μW	44,6%
Fugas (celdas)	2,9 nW	3,4 nW	-17%	3,4 nW	-17%

*$(Mejora = \%(P_{estandar} - P_{paralelo})/P_{estandar})$. *Valores negativos indican un incremento del dato.*

Cuando se comparan los valores concretos de los datos obtenidos con cada tipo de simulación se aprecian diferencias en los resultados. Por ejemplo, para el caso de peor consumo, en el caso del registro estándar la simulación eléctrica indica un consumo de 207,8 μW mientras que *Synopsys* da un valor de 180 μW, lo que, asumiendo como correcto el valor eléctrico, supone una desviación del 13% en el valor lógico. Esa inexactitud también ocurre en las medidas de consumo del paralelizado, que ahora es de un 4% (137,2 μW frente a 131,7 μW) en el caso de peor consumo o 23% en el caso de no serlo (99,6 μW frente a 76,23 μW).

El estudio de análisis de peor caso que hemos presentado en esta sección permite validar los procesos de simulación eléctrica y lógica para medir el consumo. Por

otra parte, ambas medidas de simulación apuntan de igual modo a la importante reducción en el consumo que introduce la paralelización y que en el registro de 8 bits chequeado supone obtener mejoras de alrededor del 30-40% en el consumo de potencia.

3.2. Diseño de cifradores Trivium de bajo consumo

El cifrador de flujo Trívium contiene un registro de estado con un total de 288 bits además de un número pequeño de puertas lógicas, un esquema hardware general del cifrador aparece representado en la Figura 3.7 (este esquema fue ya presentado en el Capítulo 2). La mayor parte del consumo se disipa en los biestables de su registro por lo que la aplicación de la paralelización se presenta como la técnica óptima que va a permitir reducir de una manera efectiva el consumo de potencia dinámico. La paralelización reduce la actividad de conmutación de los biestables, siendo la disminución dependiente, del flujo de datos que circulan por el cifrador.

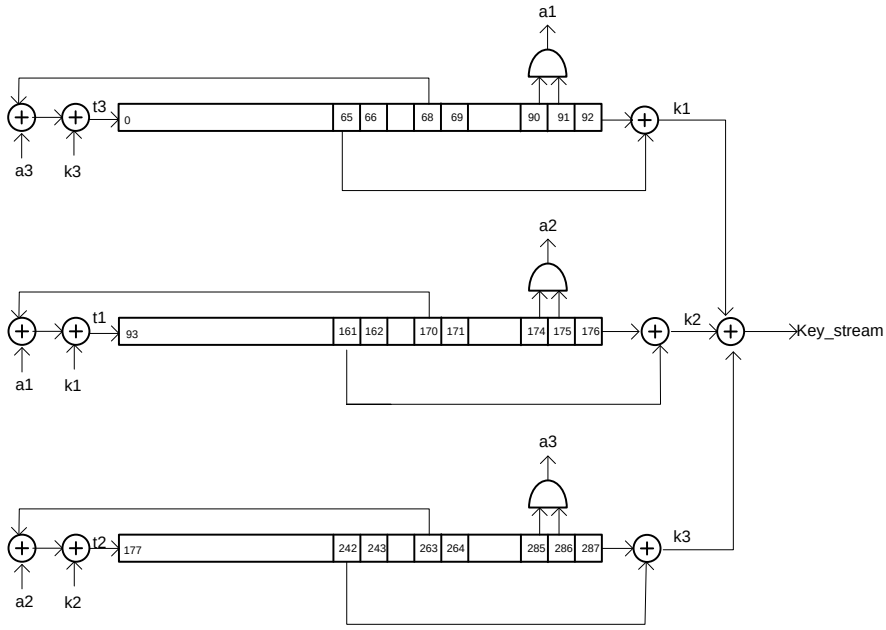


Figura 3.7 Esquema hardware general del cifrador Trivium.

En la implementación hardware del Trivium de un solo bit, el registro de estado se divide en tres registros de desplazamiento con realimentación lineal (LFSR) cuyas entradas de realimentación son los bits 0, 93 y 177. En cada uno de estos tres registros hay un grupo de biestables que configuran un registro de desplazamiento simple, similar salvo en el número de bits al estudiado en el apartado anterior, en el que se puede aplicar directamente la técnica de paralelización. Esta aplicación da

lugar a nuestra propuesta MPLP (*Mixed Parallel Low Power*) que se presenta en el epígrafe 3.2.1.

Por otro lado, la técnica puede ser aplicada también a todos los biestables que configuran cada uno de los registros de desplazamiento, a los bits 0 a 92 por una parte, a los bits 93 a 176 por otra y a los bits 177 a 287 por último. Esto añade complejidad y más puertas lógicas al circuito total. Esta aplicación da lugar a nuestra propuesta FPLP (*Full Parallel Low Power*) que se presenta en el epígrafe 3.2.2.

Por último, las peculiaridades de los Trivium con salidas multi-bit se tratan en el epígrafe 3.2.3.

3.2.1. Trivium MPLP

La propuesta de Trivium MPLP consiste en paralelizar directamente cada uno de los tres segmentos de registros de desplazamiento simples cuyas salidas no estén envueltas en operaciones combinacionales de realimentación. Con ello, de los 288 bits que forman el registro de estado se puede aplicar paralelización a 196 bits.

Esta paralelización requiere de pocas modificaciones hardware en cada registro de desplazamiento del cifrador Trívium, como se muestra en la Figura 3.8.

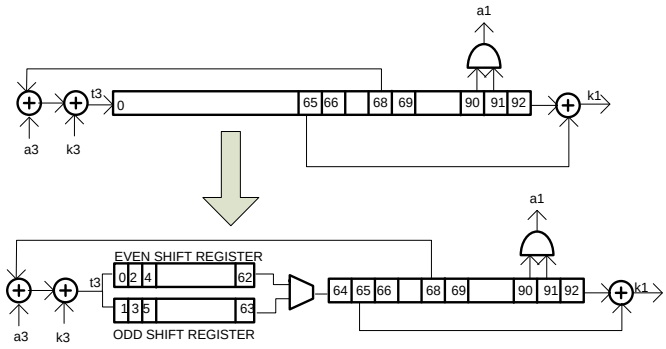


Figura 3.8 Esquema del registro LFSR aplicando la técnica de paralelización.

En efecto, consideremos por ejemplo la primera cadena (la que comienza en el bit 0). Los bits que van del 0 al 64 forman un segmento simple de desplazamiento, que no participan en la realimentación. Como es un número impar, se aplica directamente la técnica de paralelización a todos menos al último, con lo que quedaría paralelizado el grupo del 0 al 63, quedando los restantes, del 64 al 92, tal como estaban en la cadena Trivium original. El segmento del 0 al 63 queda dividido en dos registros, denominados par e impar, donde se alojan los bits 0, 2,...62 (pares) y 1, 3,...63 (impares) del segmento de registro original.

Cada uno de estos registros usa un reloj de frecuencia mitad al original. Además se añade un multiplexor que, según el valor de la señal de reloj, recoge como salida una de las de cada registro de desplazamiento desdoblado, la par (62) o la impar (63). La salida del multiplexor se conecta con el siguiente bit (el 64) para la cadena del flujo de datos del registro de desplazamiento original.

La aplicación de esta técnica a los tres segmentos da lugar a la primera propuesta de cifrador Trivium de bajo consumo, denominada MPLP y que es mostrada en la Figura 3.9.

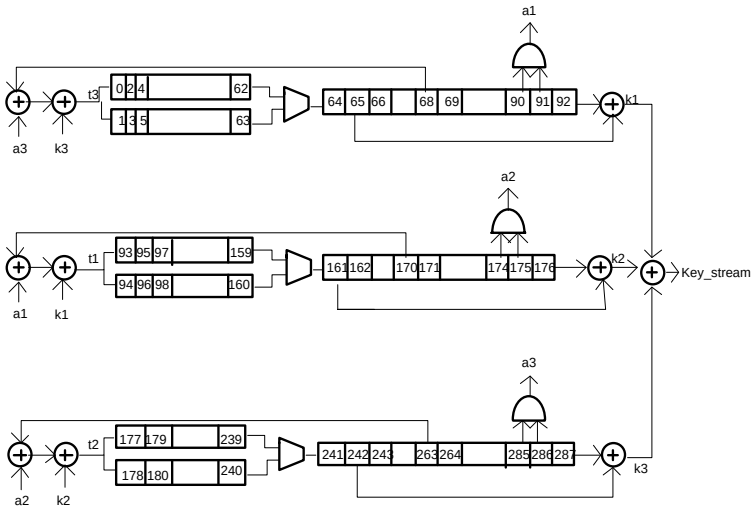


Figura 3.9 Esquema de la version de bajo consumo Trivium MPLP.

Se debe hacer notar que el registro par resultante de la partición parcial de cada uno de los registros se carga con el flanco de subida del reloj mientras que el registro impar lo hace con el flanco de bajada. El reloj de estos registros es de frecuencia mitad a la de los registros que no se encuentran paralelizados. La frecuencia de operación del Trivium no se ve afectada.

Por último, cabe decir que en este diseño no se recoge la forma de cargar la clave y el vector de inicialización, aspecto que será tratado posteriormente, aunque podemos anticipar que se realizará mediante carga en paralelo.

3.2.2. Trivium FPLP

La paralelización puede extenderse a todos los bits de cada uno de los tres registros de desplazamiento tengan o no realimentaciones. Esta nueva propuesta de cifrador de bajo consumo, que requiere mayores modificaciones del hardware, es

denominada FPLP y se muestra en la Figura 3.10.

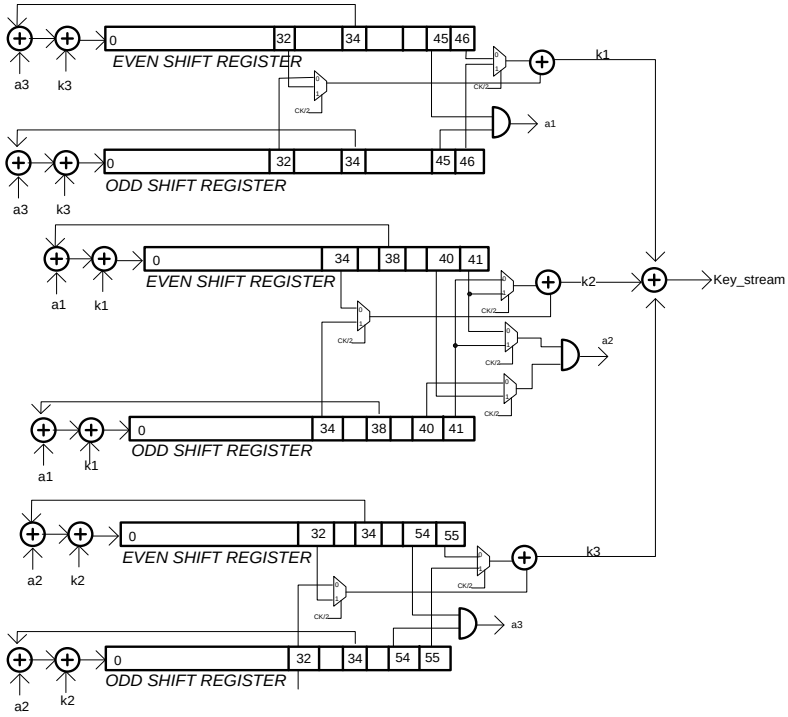


Figura 3.10 Esquema de la version de bajo consumo Trivium FPLP.

En esta segunda propuesta de cifrador Trivium de bajo consumo cada uno de los registros de estado se divide en parejas de registros, par e impar, con el mismo número de bits y con un reloj de frecuencia mitad. También se añade un multiplexor a la salida de las parejas de registros que selecciona el bit adecuado en cada ciclo de reloj. En la Figura 3.10 se indica también la longitud de cada uno de los registros.

Las mayores modificaciones en el Trivium FPLP provienen de la necesidad de generar correctamente los bits de realimentación en todos los ciclos de reloj. En efecto, la generación de los bits de entrada en cada uno de los registros y la salida de la secuencia cifrante dependen del contenido de ciertos bits del registro. La localización de estos bits también depende de si el ciclo de reloj es par o impar. En un caso el bit que se escoge será el del registro par y en el otro del registro impar. Para que esta selección sea la adecuada, se debe añadir más lógica combinacional controlada por el reloj.

Con respecto al Trivium estándar, el FPLP posee solo algunas celdas combinacionales adicionales, por lo que el incremento de recursos es prácticamente

despreciable. Por otra parte, su velocidad de operación prácticamente no se ve afectada. Comentarios similares al del epígrafe anterior cabe hacer sobre la carga inicial de la clave y del vector de inicialización.

3.2.3. Cifradores Trivium con salida multi-bit

Como ya se detalló en el Capítulo 2, además de la versión de un bit, es posible realizar el cifrador Trivium con salida multi-bit. En relación a los cambios en el hardware, recordemos que el registro de estado se mantiene con el mismo número de bits mientras que el número de señales utilizadas en las realimentaciones dependen del número de bits a la salida. Las operaciones de realimentación vienen condicionadas por este valor, que en todo caso resultan más complejas que el de único bit, con el consiguiente aumento en el número de celdas combinacionales.

En la Tabla 3.4 se muestra la complejidad de las implementaciones de Trivium de 1, 2, 8 y 16 bits. Se han incluido tanto el número de celdas básicas (biestables y puertas lógicas) como el número de transistores equivalentes (una AND de 2 entradas se implementa como una NAND, con 4 transistores, seguido de un inversor, con 2 transistores, lo que le da un total de 6 transistores). En la tabla se aprecia el aumento tan significativo en el número de celdas y puertas de las implementaciones de uno hasta dieciséis bits [29].

Tabla 3.4 Número de puertas en las implementaciones de los Trivium estándar multi-bit de 1 a 16 bits de salida.

Componentes	Nº transistores	1-bit	2 bits	8-bits	16-bits
Biestables	26	288	288	288	288
AND	6	3	6	24	48
XOR	12	11	22	88	176
Estimación transistores	-	7638	7788	8688	9888

Las versiones de bajo consumo de potencia, tanto MPLP como FPLP, se pueden adaptar también para cifradores Trivium con salida multi-bit. Siguiendo el ejemplo del cifrador Trivium estándar, se han implementado tres versiones multi-bit con salida de dos, ocho y dieciséis bits.

Un ejemplo del cifrador de baja potencia MPLP para dos bits de salida se representa en la Figura 3.11.

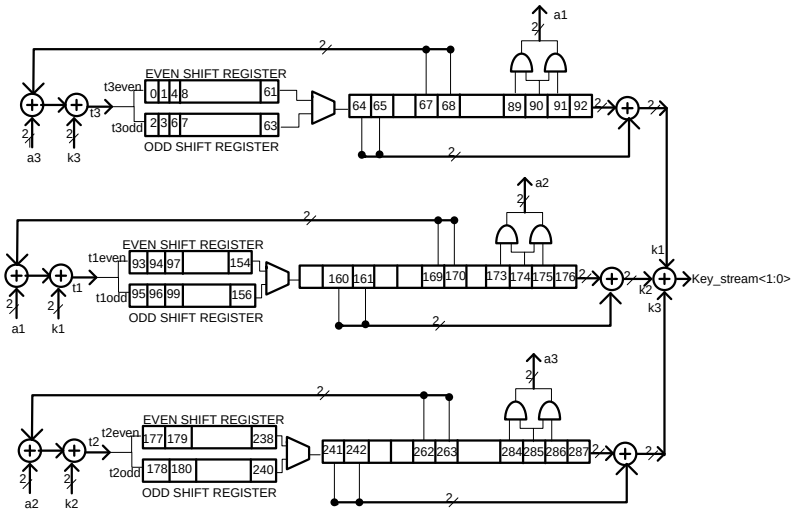


Figura 3.11 Esquema de la versión Trivium MPLP con salida de dos bits.

Si se compara este circuito con el correspondiente de un bit (Figura 3.9) se verá cómo ahora las subfunciones de realimentación son dobles e involucran a un mayor número de bits del registro de estado. Así, por ejemplo, la subfunción "a1" que en el caso de 1 bit era una señal con la operación AND de los bits 90 y 91, en el caso 2-bits son dos señales, con la operación AND de los bits 89 y 90 y la operación AND de los bits 90 y 91. Lo mismo ocurre con las otras señales de las realimentaciones y, por supuesto, también con la salida *key_stream.*

Como ocurre con el caso de un bit, el esquema de la Figura 3.11 resulta más complicado que el MPLP. Esto es debido a que, como el número de celdas combinacionales aumenta, se incrementan los multiplexores que van seleccionando qué bit del registro de estado se tiene que escoger en cada ciclo de reloj para la realimentación hacia el registro par y hacia el impar. Existe por tanto un incremento de estos componentes combinacionales, que se hace mayor conforme se tengan más bits a la salida del cifrador. En el caso de dos bits como se muestra en la Figura 3.12, se necesitarán casi el doble de multiplexores que en el caso de un bit.

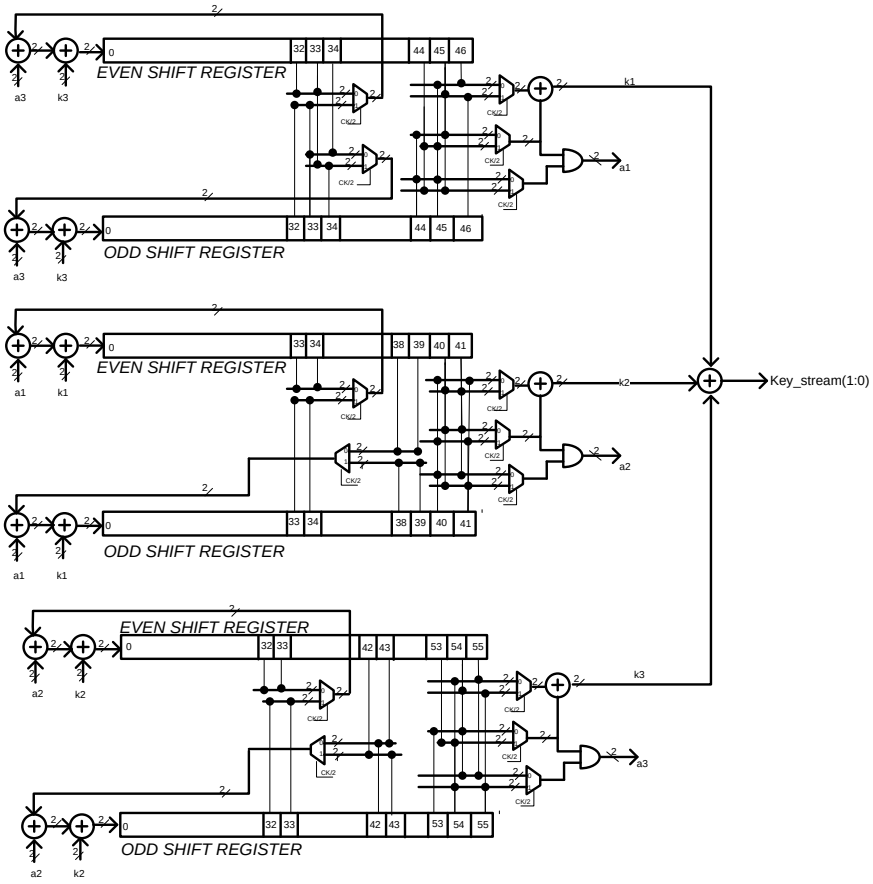


Figura 3.12 Esquema de la versión Trivium FPLP con salida de dos bits.

Para las versiones MPLP y FPLP de ocho y dieciséis bits se hace difícil dibujar la implementación a nivel de esquemático, esquemáticos que, por otra parte, no resultan de ninguna utilidad práctica y por ello no se presentan en esta Memoria de Tesis. Los diseños se han realizado en código VHDL y se presentan en el Apéndice A.

3.3. Análisis de área y consumo en Trivium MPLP y FPLP

En este apartado se presenta la caracterización en detalle, sobre todo en consumo, de las implementaciones de los cifradores Trivium estándar, MPLP y FPLP. Para ello se realizaron simulaciones tanto a nivel lógico, que es la más rápida y usual en el nivel de diseño RT propio de este trabajo, como a nivel eléctrico, que proporciona los valores de consumo más precisos. No se va a hacer hincapié en la

caracterización temporal de las diferentes implementaciones, pero todas ellas cumplen con la restricción de reloj impuestas en la síntesis para el cifrador Trivium.

Al igual que en el epígrafe 3.1, para realizar el análisis de consumo con simulaciones eléctricas se requiere disponer de una tecnología que proporcione los modelos a nivel de transistor de sus celdas de librería. La tecnología escogida fue AMS de 350 nm, ampliamente usada y muy fiable, que proporciona dichos modelos.

La descripción de los cifradores se hizo mediante código VHDL. La descripción RTL descrita fue sintetizada con la herramienta de *Synopsys Design Vision* y verificada con el simulador *ModelSim* de *Mentor Graphics*. Para la verificación funcional del prototipo se han utilizado como valores de clave y de vector de inicialización algunas de las parejas de valores de entre los descritos en [27].

Los cifradores Trivium implementados en este Capítulo llevan incluidos la lógica de carga de la clave y del vector de inicialización que se realiza de forma paralela. Esto lleva implícito la inclusión de la lógica necesaria para realizar esta operación y la de inicialización de los cifradores. Es por ello que el número de puertas va a diferir de la Tabla 3.4 vista anteriormente.

3.3.1. Análisis de área

El informe de recursos de la herramienta de síntesis para la versión estándar y de baja potencia Trivium MPLP se muestra en la Tabla 3.5. En la tabla se ha incluido, además, la columna "Mejora" para mostrar el porcentaje de ahorro del coste de la versión MPLP frente al coste de la versión estándar.

Tabla 3.5 Informe de recursos con *Synopsys* en Trivium y Trivium MPLP.

Synopsys **Informe de área**	**AMS 350nm**		
	TRIVIUM	**TRIVIUM MPLP**	**Mejora***
Nº Celdas	615	546	11%
Numero interconexiones	792	718	9%
Área Combinacional (μm²)	26990	23096	14%
Área No-combinacional (μm²)	99590	106069	-7%
Área Celdas (μm²)	126580	129165	-2%
Área interconexiones (μm²)	17559	17541	-0,1%
Área Total (μm²)	144140	146706	-2%

*$(Mejora = \%(T_{Trivium} - T_{MPLP})/T_{Trivium})$. *Valores negativos indican un empeoramiento del dato.*

Se observa que la herramienta de síntesis utiliza menos celdas de librería para implementar el prototipo Trivium MPLP que el estándar (un 11% menos). Esto es debido a que la herramienta de síntesis utiliza diferentes celdas combinacionales para la descripción del cifrador Trivium MPLP lo cual se manifiesta en que el área

combinacional ocupada es un 14% menor mientras que el área no combinacional sí aumenta en un 6% como consecuencia de la suma de algún biestable adicional como el usado para la división del reloj. La herramienta de síntesis utiliza un número bastante menor de puertas NOR y NAND de dos entradas que para el caso estándar.

Pero estas diferencias no implican una variación significativa del área total en estas dos versiones, ya que difieren en torno al 2%. En cuanto a las conexiones, el Trivium MPLP precisa un 9% menos que el estándar debido a esta reducción en el número de celdas combinacionales y su interconexionado, aunque la superficie ocupada por ellas apenas varía de una a otra versión.

La comparación de recursos entre la versión estándar de Trivium y la de baja potencia FPLP se muestra en la Tabla 3.6. También se añade la columna "Mejora" para mostrar el porcentaje de ahorro del coste de la versión FPLP frente al coste de la versión estándar.

Tabla 3.6 Informe de recursos con *Synopsys* en Trivium y Trivium FPLP.

Synopsys	**AMS 350 μm**		
Informe de área	**TRIVIUM**	**TRIVIUM FPLP**	**Mejora***
Nº celdas	615	748	-22%
Número interconexiones	792	921	-16%
Área combinacional (μm²)	26990	32159	-19%
Área No-combinacional (μm²)	99590	100573	-1%
Área celdas (μm²)	126580	132732	-5%
Área interconexiones (μm²)	17559	19017	-8%
Área Total (μm²)	144140	151750	-5%

*$(Mejora = \%(T_{Trivium} - T_{FPLP})/T_{Trivium})$. *Valores negativos indican un empeoramiento del dato.*

La herramienta de síntesis utiliza un número mayor de celdas de librería para implementar el prototipo Trivium FPLP, un 22% más que el estándar. Si exploramos el informe de celdas generado por la herramienta vemos que se ha utilizado un número diferente de celdas, pero sobre todo llama la atención la gran diferencia en el número de inversores de reloj (celda *CLKIN0*). El informe muestra un uso de 121 celdas *CLKIN0* en la versión FPLP frente a 12 en la versión estándar. Ésta es la razón del mayor aumento en el número de celdas, que queda de manifiesto también en el área combinacional ocupada, un 19% mayor.

Por su parte, el área no combinacional apenas aumenta en un 1% como consecuencia de la suma de un biestable más para la división de reloj. Con todo ello, la versión FPLP tiene una pequeña penalización en cuanto al área de un 5%.

En cuanto a las conexiones, el Trivium FPLP precisa un 16% más que el estándar debido a este aumento en el número de celdas combinacionales y su interconexionado, siendo la superficie ocupada por ellas un 8% mayor.

3.3.2. Análisis del consumo de potencia

A diferencia del registro de desplazamiento simple, ni en el Trivium estándar ni en los diseños MPLP y FPLP que hemos propuesto es factible hacer análisis de peor caso por la inviabilidad de conocer los vectores de inicialización y de claves que proporcionan ese peor caso. Sin embargo la potencia consumida depende de los datos, siendo recomendable que los tests sean varios y lo más objetivos e independientes posibles. Por ello para evaluar el impacto en el consumo de potencia al aplicar la técnica de paralelización, se han realizado simulaciones a nivel *RT* con varias parejas de claves y vectores de inicialización de las propuestas en [27]. Con una de ellas se han comparado las transiciones producidas en los biestables del registro de estado y se ha evaluado la reducción en la actividad de conmutación al aplicar la paralelización.

El resultado para la versión Trivium MPLP con una simulación de 1535 ciclos de reloj es mostrado en la Tabla 3.7. En cada ciclo de reloj se almacenaron los cambios producidos en el registro de estado (288 bits) y se evaluaron también por ciclo de reloj si las transiciones producidas cambiaban el nivel lógico anterior de "0" a "1" o viceversa. Para la versión estándar cambian una media de 138 biestables mientras que en la versión MPLP solo cambian 95 biestables de media. El máximo número de transiciones para la versión estándar es de 158 frente a las 117 para el caso MPLP.

La versión de baja potencia MPLP reduce siempre el número de transiciones en cada ciclo de reloj, lo que implica que el término de consumo de potencia debido al factor de actividad va a ser disminuido al aplicar está técnica. Se puede afirmar que de media se tienen un 31% de reducción en el número de transiciones de nivel alto a bajo y viceversa. Para el número máximo de transiciones se tiene una reducción algo menor, del orden del 26%.

Tabla 3.7 Informe de las transiciones durante una simulación MPLP.

Trivium	Transiciones totales		Reducción Media-Max.	Transiciones 0-1		Transiciones 1-0	
	Media	Max.	%	Media	Max.	Media	Max.
Estándar	138	158	-	69	80	69	78
MPLP	95	117	31%-26%	47	58	47	59

En el caso de la versión Trivium FPLP el resultado es mostrado en la Tabla 3.8. Se obtuvo también un menor número de cambios de media en la versión de bajo consumo frente a la estándar, reduciéndose el número de transiciones en cada ciclo de reloj. Se puede afirmar que de media se tienen un 49% de reducción en el número de transiciones de nivel alto a bajo y viceversa. Para el número máximo de transiciones se tiene una reducción algo menor, del orden del 46%. Esto es, en la

versión FPLP se reducen casi a la mitad el número de transiciones.

Estas reducciones son mayores que las obtenidas en el caso de la versión MPLP lo que nos hace pensar en una mayor reducción de la potencia para la versión FPLP.

Tabla 3.8 Informe de las transiciones durante una simulación FPLP.

Trivium	Transiciones totales		Reducción Media-Max.	Transiciones 0-1		Transiciones 1-0	
	Media	Max.	%	Media	Max.	Media	Max.
Estándar	138	158	-	69	80	69	78
FPLP	70	86	49%-46%	35	43	35	43

La reducción del número de transiciones en cada ciclo de reloj implica que el término de consumo de potencia en los cifradores debido al factor de actividad va a disminuir en un porcentaje de entre el 31% y 49% dependiendo de la propuesta de Trivium de bajo consumo elegida.

A continuación se han sintetizado las diferentes versiones Trivium con la tecnología elegida y simulado funcionalmente a nivel lógico y a nivel eléctrico para su verificación.

3.3.2.1. Consumo de potencia con simulaciones a nivel lógico.

Para el cálculo del consumo de las propuestas Trivium de bajo consumo, una vez sintetizadas en la librería tecnológica de AMS 350 nm, se realizaron simulaciones digitales que proporcionaron los ficheros de actividad del circuito, que posteriormente fueron introducidos en la herramienta propia de *Synopsys* para el cálculo del consumo de potencia.

Con los ficheros de actividad obtenidos tras una simulación de 1700 ciclos de reloj (68 μs) de frecuencia 25 MHz se obtuvieron los resultados de potencia mostrados en la Tabla 3.9 para la propuesta Trivium MPLP y en la Tabla 3.10 para el Trivium FPLP. En ambas tablas se observa que la potencia de perdidas (fugas) es despreciable frente a la dinámica (diferencia de cuatro órdenes de magnitud), por lo que no la consideraremos en adelante. También se ha incluido, además, la columna "Mejora" para mostrar el porcentaje de ahorro del consumo de las versiones de bajo consumo frente al de la versión estándar.

La versión Trivium MPLP tiene algo más de un 30% menos de consumo de potencia dinámica que la estándar lo que concuerda con el análisis del factor de actividad. Esta mejora afecta tanto a la potencia interna como al consumo de las conmutaciones en las interconexiones.

Por otro lado, la versión FPLP tiene un 50% menos de consumo de potencia dinámico como ya se preveía tras el análisis del factor de actividad. Hay sin embargo un incremento considerable (40%) en el consumo de las conmutaciones en

las interconexiones como consecuencia del aumento del número de éstas y del número de celdas combinacionales.

Tabla 3.9 Informe de potencia con *Synopsys* en Trivium y Trivium MPLP.

Potencia@25 MHz-V=3,3 V *Synopsys*	**AMS 350 nm**		
	TRIVIUM	**TRIVIUM MPLP**	**Mejora***
Interna (celdas)	4,26 mW	2,87 mW	33%
Conmutación (interconexiones)	0,67 mW	0,44 mW	34%
Dinámica (total)	4,93 mW	3,24 mW	34%
Fugas (celdas)	128 nW	190 nW	-48%

*$(Mejora = \%(T_{Trivium} - T_{FPLP})/T_{Trivium})$. *Valores negativos indican un empeoramiento del dato.*

Los resultados de las simulaciones lógicas de las versiones de bajo consumo Trivium muestran mejoras generales en el consumo dinámico del orden del 34-50% dependiendo de la versión Trivium elegida, siendo mejor el caso FPLP. Esta versión consigue una disminución del orden del 63% en el consumo de potencia interna debido a la reducción de la actividad de conmutación con la paralelización de los registros de desplazamiento a pesar del aumento de su consumo debido a las capacidades de interconexión y de carga.

Tabla 3.10 Informe de potencia con *Synopsys* en Trivium y Trivium FPLP.

Potencia@25 MHz-V=3,3 V *Synopsys*	**AMS 350 nm**		
	TRIVIUM	**TRIVIUM FPLP**	**Mejora***
Interna (celdas)	4,26 mW	1,54 mW	63%
Conmutación (interconexiones)	0,67 mW	0,94 mW	-40%
Dinámica (total)	4,93 mW	2,48 mW	49,7%
Fugas (celdas)	128 nW	132 nW	-3%

*$(Mejora = \%(T_{Trivium} - T_{FPLP})/T_{Trivium})$. *Valores negativos indican un empeoramiento del dato.*

3.3.2.2. Consumo de potencia con simulaciones a nivel eléctrico

Se han realizado también simulaciones eléctricas para calcular la potencia y corriente consumida. Las simulaciones eléctricas, como ya se ha indicado, tienen la ventaja de ser más precisas pero consumen mayores recursos en tiempo de simulación.

En la Figura 3.13 se representan las formas de onda de la corriente de la fuente de polarización del circuito obtenida con *Spectre* durante 1 μs. La escala de la corriente se tiene en mA. Para apreciar mejor los picos de corriente las gráficas tienen el eje de la corriente adecuado para cada medida, no siendo las mismas en el caso estándar y de bajo consumo. En la Figura 3.13.a que se corresponde al cifrador Trivium estándar, la cota máxima en el eje de la corriente llega hasta -60 mA y en Figura 3.13.b que se corresponde al cifrador de bajo consumo MPLP llega hasta -30 mA.

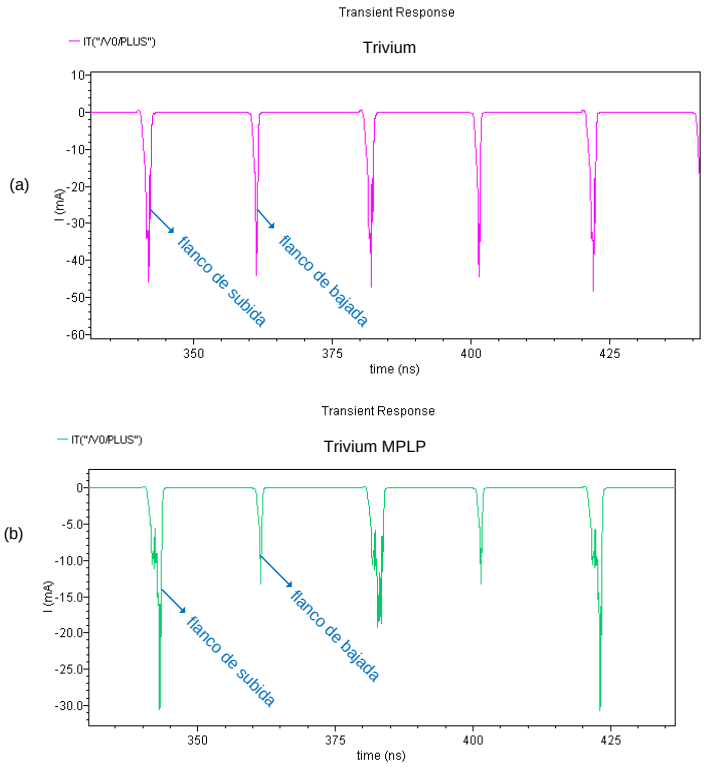


Figura 3.13 Picos de corriente en la fuente de polarización del cifrador Trivium, a) estándar, b) MPLP.

Se aprecia cómo los picos de corriente en ambos flancos de reloj en la versión estándar (Figura 3.13.a) son muy similares, mientras que la versión MPLP (Figura 3.13.b) presenta una reducción importante durante el flanco de subida y una disminución aun mayor con los flancos de bajada, como ya ocurría en las simulaciones eléctricas de los registros de desplazamiento. Recordemos que en la versión MPLP se tienen biestables con el reloj dividido por dos y otros con el reloj principal. Los biestables que utilizan el reloj principal aportan picos de corriente en los flancos de subida y bajada, mientras que los biestables que utilizan el reloj dividido por dos aportan picos de corriente en el flanco de subida del reloj principal solamente.

En la Tabla 3.11 se muestran los datos de potencia media y los valores máximos de los picos de corriente medidos con las simulaciones a nivel eléctrico. Estos resultados muestran que los picos de corriente se reducen aproximadamente un 40% en los flancos de subida y casi un 67% en los de bajada. La potencia media total tiene una disminución del 40% en la versión MPLP frente a la versión estándar, al igual que la corriente.

Tabla 3.11 Informe de potencia y corriente en Trivium estándar y Trivium MPLP.

Potencia/Corriente@25 MHz V=3,3 V	**AMS 350 nm**		
	TRIVIUM	**TRIVIUM MPLP**	**Mejora***
Potencia media	5,0 mW	3 mW	40%
Corriente media	1,5 mA	0,86 mA	40%
Picos de corriente máx. flanco [1]+	52 mA	30,7 mA	41%
Picos de corriente máx. flanco [2]-	43 mA	14 mA	67%

[1]*flanco de subida de reloj.* [2]*flanco de bajada de reloj.*

*$(Mejora = \%(T_{Trivium} - T_{MPLP})/T_{Trivium})$.

De igual modo en la Figura 3.14 se representan las formas de onda de la corriente de la fuente de polarización para la versión Trivium estándar y FPLP. En este caso los picos de corriente en los flancos de subida se reducen un 48% en la versión FPLP (Figura 3.14.b) respecto a la versión estándar (Figura 3.14.a) y casi desaparecen en los flancos de bajada reduciéndose un 83%. Esto es así porque en la versión FPLP todos los biestables utilizan el reloj dividido por lo que el flanco de bajada del reloj apenas tiene efecto.

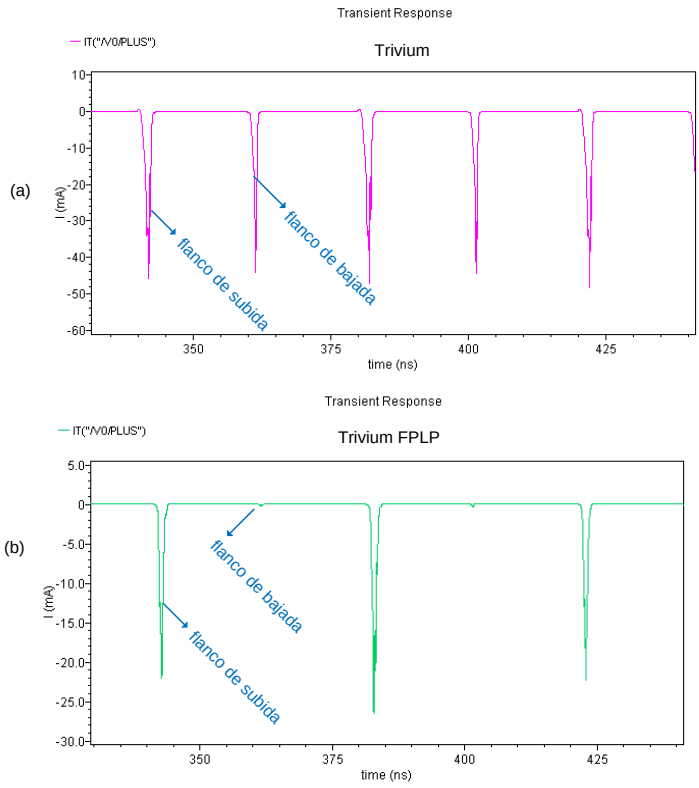


Figura 3.14 Picos de corriente en la fuente de polarización del cifrador Trivium, a) estándar, b) FPLP.

En la Tabla 3.12 se muestran los consumos de las versiones estándar y FPLP así como los valores máximos de los picos obtenidos con la simulación. El consumo medio de potencia disminuye un 66% en la versión FPLP debido a la reducción de los picos de corriente en los flancos de subida y su casi desaparición en los de bajada.

Los datos de consumo de potencia aportados por las simulaciones a nivel eléctrico confirman los de las simulaciones lógicas y demuestran que las versiones de bajo consumo Trivium consiguen reducciones en el consumo de potencia entre el 40-67% dependiendo de la versión de bajo consumo y picos de corriente en los flancos de reloj de menor amplitud. También confirma que la versión FPLP obtiene reducciones mayores.

Tabla 3.12 Informe de potencia y corriente en Trivium y Trivium FPLP.

Potencia@25 MHz	**AMS 350 nm**		
Vcore = 3,3 V	**TRIVIUM**	**TRIVIUM FPLP**	**Mejora***
Potencia media	5 mW	1,64 mW	67%
Corriente media	1,5 mA	0,49 mA	67%
Picos de corriente máx. flanco[1]+	52 mA	27 mA	48%
Picos de corriente máx. flanco[2]-	43 mA	0,5 mA	83%

[1]flanco de subida de reloj. [2]flanco de bajada de reloj.

*$(Mejora = \%(T_{Trivium} - T_{FPLP})/T_{Trivium})$.

Cuando se comparan los porcentajes de mejora de los datos obtenidos con cada tipo de simulación se aprecian similitudes aunque con cierta desviación en las reducciones en el consumo de potencia. Por ejemplo, en el caso del cifrador Trivium MPLP, la simulación eléctrica indica un porcentaje de mejora del 40% frente a un 34% (desviación de 6 puntos) de la simulación lógica. Para el cifrador Trivium FPLP la simulación eléctrica indica un porcentaje de mejora del 67% (desviación de 17 puntos) frente a un 50% de la simulación lógica (desviación de 17 puntos).

3.3.3. Análisis de área y consumo de potencia en Trivium multi-bit

En este trabajo se han desarrollado también cifradores Trivium multi-bit tanto en sus versiones estándar como paralelizadas, para los casos de 2, 8 y 16 bits.

A continuación se presentan los datos del número de celdas, interconexiones y área dados por la herramienta *Synopsys* además del consumo de potencia tras la síntesis de todas las propuestas de los cifradores Trivium con salida uno (Trivium_x1), dos (Trivium_x2), ocho (Trivium_x8) y dieciséis (Trivium_x16) bits.

3.3.3.1. Análisis de área en Trivium multi-bit

Tras la síntesis la Tabla 3.13 muestra el área ocupada, el número de celdas lógicas e interconexiones de cada uno de los cifradores Trivium diseñados. También muestra el área de las celdas combinacionales y no combinacionales (biestables) usados, y el área de las interconexiones. Además para analizar los resultados, se muestra en la Tabla 3.14 los incrementos (%) de cada medida respecto a la versión Trivium de 1 bit. Para apreciar mejor los detalles del informe de la herramienta, los datos se han representado gráficamente en la Figura 3.15.

Tabla 3.13 Área y número de celdas lógicas del informe de post-síntesis.

Prototipos Trivium	Librería: AMS 350					
	Nº celdas	Nº nets	Área Total (μm^2)	Área Combinacional (μm^2)	Área No Combinacional (μm^2)	Área Inter. (μm^2)
Trivium_x1	615	792	144140	26991	99590	17559
Trivium_x1mp	546	718	146706	23096	106069	17541
Trivium_x1fp	748	921	151749	32159	100573	19017
Trivium_x2	795	963	151904	31595	99590	20718
Trivium_x2mp	550	726	148269	23569	106888	17811
Trivium_x2fp	774	953	154954	34616	100573	19764
Trivium_x8	899	1076	162913	40859	99590	22464
Trivium_x8mp	661	863	157817	32014	105796	20007
Trivium_x8fp	1003	1235	177725	50687	102648	24390
Trivium_x16	1034	1224	177402	53053	99590	24759
Trivium_x16mp	753	987	172248	40986	108617	22644
Trivium_x16fp	1321	1593	212243	71235	110255	30753

El primer hecho destacable cuando se analizan los resultados es que los recursos consumidos crecen significativamente con el aumento de bits de salida. Así, como se muestra en la Tabla 3.14, el número de celdas en la versión estándar de 16 bits es un 68% mayor que en la versión de 1 bit, ocurriendo algo similar en la versión MPLP (con un aumento del 38%) y de la versión FPLP (aumento del 77%). Incrementos semejantes ocurren en el número de nodos (55% en estándar, 37% en MPLP, 73% en FPLP) y, aunque con menor medida, en el área total (23% en estándar, 17% en MPLP y 40% en FPLP).

Los cifradores Trivium FPLP tienen más celdas lógicas y más área combinacional que las otras propuestas de cifradores Trivium. Por su parte, el área no combinacional es parecida en todas las versiones excepto en la versión FPLP de 16 bits donde aumenta un 10% debido al aumento en el número de biestables en algunos registros paralelos. Lo más llamativo es el aumento en el área de las celdas combinacionales que aparece en las versiones FPLP de ocho y dieciséis bits, con un aumento del 58% y del 122% respectivamente. Esto es así debido al número de bits

que hay que realimentar y operar lógicamente, lo que hace que aumente en gran medida el número de celdas combinacionales.

Por su parte, la versión Trivium MPLP, que es la que mejor evoluciona con el número de bits de salida, obtiene en la síntesis de *Synopsys* menos celdas combinacionales y menos nodos de interconexión que las otras dos versiones. Además, para el caso multi-bit (sean 2, 8 o 16) es la versión del cifrador Trivium con menos área total.

Tabla 3.14 Porcentajes de área y número de celdas lógicas, referenciados a la version "x1".

Prototipos Trivium	Librería: AMS 350					
	Nº celdas	**Nº nets**	**Área Total**	**Área Combinacional**	**Área No Combinacional**	**Área Inter.**
Trivium_x1	-	-	-	-	-	-
Trivium_x1mp	-	-	-	-	-	-
Trivium_x1fp	-	-	-	-	-	-
Trivium_x2	29%	22%	5%	17%	0%	18%
Trivium_x2mp	1%	1%	1%	2%	1%	2%
Trivium_x2fp	3%	3%	2%	8%	0%	4%
Trivium_x8	46%	36%	13%	51%	0%	28%
Trivium_x8mp	21%	20%	8%	39%	0%	14%
Trivium_x8fp	34%	34%	17%	58%	2%	28%
Trivium_x16	68%	55%	23%	97%	0%	41%
Trivium_x16mp	38%	37%	17%	77%	2%	29%
Trivium_x16fp	77%	73%	40%	122%	10%	62%

*$(\% = \%(T_{estandar} - T_{FPLP-MPLP})/T_{estandar})$.

Los datos de las propuestas Trivium de bajo consumo MPLP y FPLP mostrados en la Tabla 3.14, mantienen una buena relación en el número de celdas e interconexiones frente a la versión estándar en las propuestas de un bit y dos bits a la salida, dando un área muy parecida cuando se comparan entre ellas, pero a partir de ocho y dieciséis bits la diferencia se hace mucho mayor, sobre todo en la versión FPLP de dieciséis bits donde hay una clara diferencia de recursos y área.

Las diferencias entre las diferentes versiones estándar y paralelizadas a nivel del número de celdas e interconexiones y área total son mostradas gráficamente también en la Figura 3.15, con las mismas conclusiones.

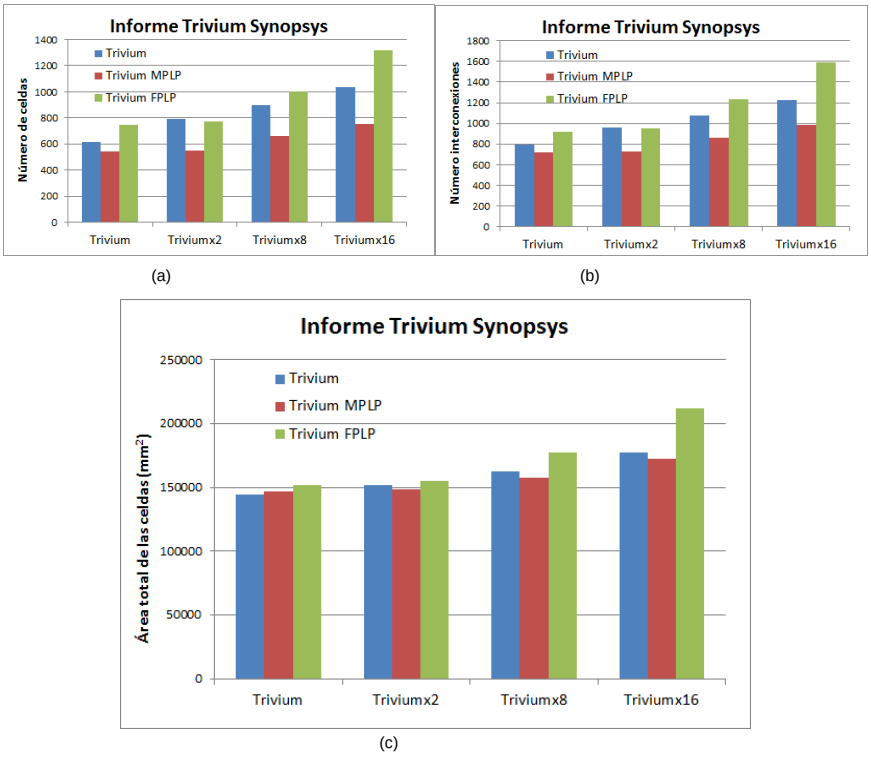


Figura 3.15 Gráfica post-síntesis de las versiones Trivium multi-bit, a) número de celdas, b) número de interconexiones y c) área toral de las celdas.

3.3.3.2. Análisis del consumo de potencia en Trivium multi-bit

Se han obtenido los datos del consumo promedio de potencia con la herramienta de síntesis de *Synopsys,* con la utilización de un fichero con la actividad de conmutación de los nodos del circuito. Para la generación de este fichero de actividad se ha realizado una simulación con la *netlist* a nivel de puertas generada tras la síntesis de las diferentes propuestas de los cifradores Trivium con salida uno, dos, ocho y dieciséis bits.

En la Tabla 3.15 se representan los datos reportados por la herramienta de *Synopsys* del consumo de potencia post-síntesis en condiciones típicas, así como el porcentaje de mejora del consumo respecto a la versión estándar.

Tabla 3.15 Consumo de potencia post-síntesis AMS 350 nm.

Prototipos Trivium	**Vcore=3,3 V-25 MHz-Caso Típico**				
	Consumo de Potencia				
	mW		**nW**	**mW**	**% Mejora Total***
	Interna	***Conmutación***	***Fugas***	***Total***	
Trivium_x1	4,26	0,67	128,77	4,93	-
Trivium_x1mp	2,88	0,44	190,07	3,32	33
Trivium_x1fp	1,54	0,94	131,82	2,48	50
Trivium_x2	4,34	0,85	128,34	5,19	-
Trivium_x2mp	2,57	0,65	198,51	3,22	38
Trivium_x2fp	2,41	0,57	132,37	2,98	43
Trivium_x8	4,62	1,17	133,42	5,79	
Trivium_x8mp	3,98	1,01	192,0	4,99	14
Trivium_x8fp	2,99	1,26	143,02	4,25	27
Trivium_x16	4,80	1,50	140,0	6,40	-
Trivium_x16mp	5,70	1,53	225,89	7,23	-13
Trivium_x16fp	4,64	2,71	163,0	7,35	-15

*$(Mejora = \%(P_{estandar} - P_{FPLP-MPLP})/P_{estandar})$. *Valores negativos indican un empeoramiento del dato*

Para apreciar mejor los detalles del consumo total de potencia, esos datos se representan gráficamente en la Figura 3.16. Como cabía esperar, el consumo crece con el aumento del número de bits de salida.

Los datos del consumo post-síntesis confirman la mejora en el consumo de potencia dinámica total en los cifradores Trivium MPLP y FPLP de 1, 2 y 8 bits de salida. Los casos más favorables en cuanto a la mejora del consumo de potencia se dan en los casos de menor número de bits de salida, siendo los casos de uno y dos bit de los Trivium FPLP y MPLP los que mejores resultados ofrezcan en su conjunto.

La mejora en la potencia total se consigue debido a la fuerte disminución que se produce en el consumo de potencia interno, que disminuye drásticamente en todas las versiones Trivium, aunque conforme se aumenta el número de bits de salida esta reducción se hace algo menor.

La potencia dinámica debida a la conmutación aumenta ligeramente en todos los casos de los cifradores Trivium FPLP mientras que en los MPLP disminuye hasta la versión de ocho bits.

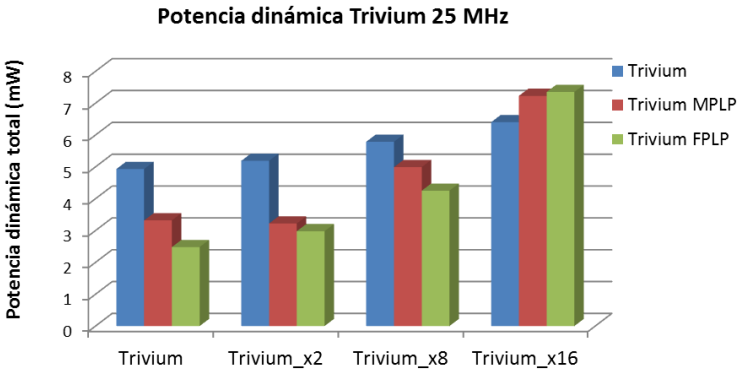


Figura 3.16 Gráfica del consumo de potencia total en cifradores Trivium.

Por último, en la Figura 3.17, se presenta un diagrama de barras con los porcentajes de mejora en los consumos para comparar las versiones MPLP y FPLP. Recordamos que los porcentajes son positivos cuando el consumo es menor que el del estándar y negativo en caso contrario.

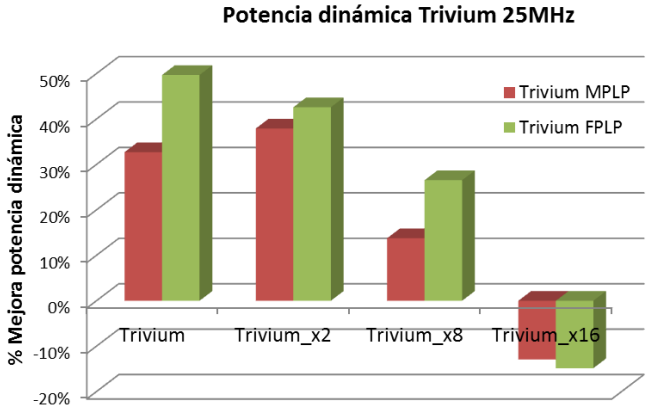


Figura 3.17 Gráfica del porcentaje de mejora en el consumo post-síntesis.

En la Figura 3.17 es inmediato apreciar el menor consumo de todas las propuestas de los cifradores Trivium MPLP y FPLP excepto en el caso de las propuestas de 16 bits a la salida. Las propuestas de bajo consumo Trivium MPLP y FPLP tienen mejoras en el consumo de potencia frente a la versión estándar entre el 27 y el 43% en las versiones con uno y dos bits de salida, siendo algo menor (entre 14-27%) en las versiones de bajo consumo con 8 bits de salida. Sin embargo las versiones de bajo consumo de 16 bits de salida no consiguen mejorar a la versión estándar, lo cual es debido a su mayor número de celdas e interconexiones, teniendo una penalización del 13-15% aproximadamente. La versión FPLP es la propuesta con mejores resultados en consumo de potencia hasta salidas con 8 bits.

3.4. Análisis de área y consumo en diferentes tecnologías ASIC.

La tecnología AMS 350 nm, aunque plenamente vigente, ya está superada por muchas otras tecnologías, algunas de las cuales están disponibles para nosotros. Con el objeto de analizar cómo afecta la elección de tecnologías de diferentes tamaños en las prestaciones de los diseños propuestos en esta Tesis, en este apartado se presentan los resultados de sus implementaciones en tres tecnologías de fabricación fuertemente submicrónicas. En este estudio nos hemos limitado a comparar la mejor versión en consumo (Trivium FPLP) con la versión Trivium estándar.

Para cubrir este objetivo se ha realizado un estudio de área y consumo de potencia con las tecnologías de fabricación de UMC *(United Microelectronics Corporation*) de 180 nm, 130 nm y 90 nm. En estas tecnologías no se disponen de modelos a nivel de transistor con lo cual no es posible realizar simulaciones eléctricas. Solo se han llevado a cabo simulaciones a nivel lógico por lo que los resultados del consumo de potencia son los dados por la herramienta de *Synopsys*.

En la Tabla 3.16 se presenta un informe de área de los Trivium estándar y FPLP. Además, para analizar los resultados en la Tabla 3.17, se muestran los incrementos (%) de consumo respecto a la versión Trivium estándar.

Tabla 3.16 Informe de síntesis *Synopsys* con tecnologías de diferentes tamaños en Trivium estándar (TRIV) y Trivium FPLP.

***Synopsys*@25 MHz**		**180 nm**		**130 nm**		**90 nm**	
Informe		**TRIV**	**FPLP**	**TRIV**	**FPLP**	**TRIV**	**FPLP**
Nº celdas		604	644	608	627	602	626
Nº interconexiones		769	808	775	795	766	796
Área (μm^2)	Área combinacional	2755	2877	2270	2810	1748	2184
	Área no combinacional	4898	5246	8064	7709	5645	5134
	Área total celdas	7653	8123	10334	10519	7392	7318

Con respecto al informe temporal, ambas versiones son muy parecidas y cumplen con la restricción de reloj (25 MHz) impuesta en la síntesis para el cifrador Trivium. Sin embargo las versiones FPLP, aunque la cumplen, son algo más lentas debido al incremento de la lógica combinacional.

Tabla 3.17 Porcentajes de área y número de celdas lógicas, con referencia a la version "x1".

Trivium_x1 Trivium_x1fp	Tecnología UMC				
	Nº celdas	Nº Inter.	Área Combinacional*	Área No Combinacional*	Área Total*
UMC_180	7%	5%	4%	7%	6%
UMC_130	3%	3%	24%	-4%	2%
UMC_90	4%	4%	25%	-9%	-1%

*$(\% = \%(T_{estandar} - T_{FPLP-MPLP})/T_{estandar})$ *. *Valores positivos indican empeoramiento del dato*

En cuanto al área, la versión de bajo consumo utiliza siempre más celdas de librería (3-7%) que la versión estándar, aunque el número de celdas disminuye conforme la tecnología se hace más pequeña. Esto es debido a que la herramienta de síntesis elige las celdas de la propia librería suministrada por el fabricante y que pueden no ser las mismas en distintas tecnologías. El área de la lógica de las celdas no combinacionales fluctúa en las tres tecnologías pues en alguna de ellas aumenta el número y en otras disminuye. Esto es debido al tipo de celdas de la librería que contengan cada una de ellas y que no es igual en todos los casos, a pesar de que el número de biestables internos no cambian (solo se añade un biestable en FPLP para la división del reloj). En 130 nm y 90 nm, la herramienta utiliza biestables diferentes en las síntesis de los cifradores Trivium estándar y FPLP, siendo los utilizados en la versión FPLP de menor área. Esto hace que el área no combinacional sea menor en la versión FPLP aun teniendo el mismo número de biestables.

Concluyendo, cuando la tecnología disminuye el tamaño de la puerta, el número de celdas en la versión FPLP disminuye ligeramente: desde un 6% en 180 nm hasta un 4% en 90 nm. Globalmente la penalización en área es de 6% en 180 nm y 2% en 130 nm pero disminuye un 1% cuando la tecnología es de 90 nm debido a que el tamaño de los biestables usados es menor que en la versión estándar.

Se han obtenido datos del consumo promedio de potencia a nivel lógico siguiendo el mismo procedimiento comentado anteriormente. Los datos obtenidos se muestran en la Tabla 3.18.

Puede observarse que el consumo total dinámico va reduciéndose conforme baja la escala de integración de la tecnología. En todos los casos sigue siendo mayor el consumo interno de las celdas que el de las interconexiones y capacidades parásitas. De hecho, el porcentaje de la contribución de las celdas crece frente al de las interconexiones, tanto en la versión estándar (84% para 180 nm, 91% para 130 nm, 93% para 90 nm), como en FPLP (71% para 180 nm, 76% para 130 nm, 82% para 90 nm). Curiosamente, en cada tecnología la versión FPLP consume menos potencia en las celdas internas y consume más en las interconexiones. En efecto, en celdas internas y para la tecnología de 180 nm el consumo de la versión FPLP es un 63% del de la versión estándar, mientras que en las interconexiones el consumo de

la versión FPLP es el 114% del de la versión estándar. Para la tecnología 130 nm estos porcentajes son el 61% y el 163% y para 90 nm los porcentajes son el 75% y el 175%.

El consumo interno de las celdas va decreciendo conforme disminuye el tamaño de la tecnología. Sin embargo el consumo debido a las conmutaciones en las interconexiones aumenta en la versión FPLP como consecuencia del aumento en el número de interconexiones y de los *glitches*. Este aumento es del 13% en 180 nm, del 63% en 130 nm y del 75 % en 90 nm.

Tabla 3.18 Informe de la potencia con *Synopsys* de la versión Trivium (TRIV) y Trivium FPLP con diferentes tamaños de la tecnología.

***Synopsys*@25 MHz Informe de Potencia (μW)**	**180 nm**		**130 nm**		**90 nm**	
	Vcore =1,8 V		**Vcore = 1,2 V**		**Vcore = 1,2 V**	
	TRIV	**FPLP**	**TRIV**	**FPLP**	**TRIV**	**FPLP**
Interna	868	545	216	131	204	154
Conmutación interconexiones	140	159	19	31	16	28
Total Dinámica	1007	703	235	162	219	182
Fugas	0.09	0.1	0.25	0.29	18	19

En la Figura 3.18 se presenta un diagrama de barras con los porcentajes de mejora en los consumos. Cuando se compara el consumo de potencia de las versiones estándar y FPLP en las tres tecnologías, la versión FPLP tiene siempre un menor consumo de potencia que la versión estándar, un 30% menor en 180 nm, un 31% menor en 130 nm y un 17 % menor en 90 nm.

Otro aspecto destacable es el aumento progresivo de la potencia de fugas conforme se disminuye la escala de integración, llegando a ser del orden del consumo de las conmutaciones en las interconexiones en la tecnología de 90 nm, aunque todavía muy por debajo del consumo dinámico total.

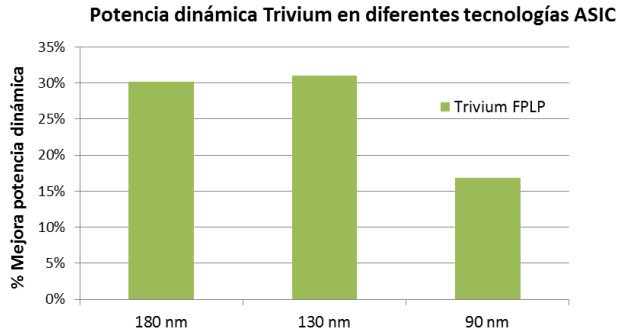


Figura 3.18 Gráfica de la mejora en el consumo de potencia de la versión Trivium FPLP frente a la estandar Trivium con diferentes tamaños de la tecnología.

3.5. Implementaciones en FPGAs

El interés de las pruebas de diseño realizadas sobre FPGAs (*Field Programmable Gate Arrays*) se centra en ver las posibilidades de esta tecnología para la realización de cifradores de flujo Trivium con la arquitectura propuesta, explorando mediante la realización de diversos diseños la evolución en el consumo de recursos y potencia. A diferencia de las realizaciones ASIC, los dispositivos FPGAs cuentan con unos recursos fijos. Estos recursos, en caso de que no se utilicen, permanecen desaprovechados.

En la literatura hay pocas contribuciones que traten el análisis y la reducción del consumo de potencia en cifradores Trivium sobre FPGA. En el Capítulo 2 se dieron datos de algunos de ellas. En [19] se muestra cómo una implementación Trivium en FPGAs de *Altera* tienen aproximadamente 398 mW de consumo interno, mientras que en [15] se tiene 382 mW.

Se ha escogido *Xilinx* como fabricante de los dispositivos programables para nuestras pruebas de consumo. Este fabricante posee diferentes familias de dispositivos adaptadas, cada una de ellas a un rango de aplicaciones. Así por ejemplo, la familia *Spartan 3* [53] posee una serie de dispositivos para aplicaciones en las que se requiera un nivel bajo de complejidad. Los dispositivos de la familia *Virtex* [54] cubren un amplio rango de necesidades que van desde complejidades medias hasta complejidades muy altas.

En nuestro caso los diseños de las versiones del cifrador Trivium de un bit fueron implementados en una familia de bajo coste, *Spartan-3E*, y en otra de altas prestaciones, *Virtex-5*. La familia FPGA *Spartan-3E* fabricada con tecnología de 90 nm, está especialmente dedicada a diseños con un volumen alto de integración y bajo coste. La familia *Virtex-5*, fabricada con tecnología de 65 nm, está dedicada a diseños de altas prestaciones basados en núcleos IP (*Intellectual Property*) y módulos a medida.

El software utilizado para las implementaciones ha sido *Xilinx ISE* 14.7 *Design Suite* con el simulador ISim. Los patrones de test han sido los mismos que se han utilizado para la verificación en tecnologías ASIC. Los resultados de consumo de potencia se han obtenido con la herramienta de *Xilinx Power Analyzer* (XPA) [78], utilizando también ficheros tipo saif (*switching activity interchanging file*) obtenidos de una simulación post-rutado con frecuencia de reloj 25 MHz.

3.5.1. Recursos de lógica utilizados en FPGAs

El recurso principal utilizado para la implementación de la lógica secuencial o combinacional en estos dispositivos es el bloque de lógica configurable también

llamado CLB. Cada CLB se encuentra conectado a una matriz de conmutación a través de la cual se accede a las líneas de conexionado de la FPGA. Para la gestión de los relojes, Xilinx tiene un bloque de hardware dedicado a ello denominado DCM (*Digital Clock Manager*).

La estructura interna de los CLBs difiere en las distintas familias de los dispositivos *Xilinx*, aunque todos ellos presentan ciertas semejanzas en cuanto a que un CLB en estas familias contiene dos *Slices* que se organizan en columnas y no se encuentran directamente conectados entre ellos. Los *Slices* tienen generadores de funciones o tablas de verdad (LUTs), biestables, multiplexores dedicados y lógica de acarreo.

Cada *Slice* de las familias *Spartan-3E* contiene cuatro LUTs de cuatro entradas, lógica de acarreo y dos biestables. Hay también dos buffers tri-estado asociados a cada CLB que pueden ser accesibles por todas las salidas de una CLB. La familia *Spartan-3E* trabaja internamente a 1,2 V.

En el caso de las familias *Virtex-5* cada *Slice* contiene también cuatro LUTS pero a diferencia de la *Spartan-3E* el número de entradas y biestables es mayor, seis entradas y cuatro biestables. La familia *Virtex-5* trabaja internamente a 1,0 V.

Respecto a los recursos utilizados, los informes de la herramienta de *Xilinx* para las versiones estándar y FPLP del cifrador Trivium de 1bit en las FPGAs se muestran en las Tabla 3.19 y Tabla 3.20. Estos informes muestran que las implementaciones de bajo consumo FPLP y MPLP tienen ligeras penalizaciones en comparación con la versión estándar.

Tabla 3.19 Informe de los recursos implementación en *Spartan 3E*.

Utilización de la lógica XC3s500e-fg320	**TRIVIUM**	**TRIVIUM MPLP**	**TRIVIUM FPLP**
Slice Flip Flops	288	288	290
LUT de 4 entradas	455	454	461
Slices ocupadas	229	229	235
BUFGMUXs	1	2	2
DCMs	1	1	1

En el dispositivo *Spartan-3E* no hay diferencias apreciables entre la versión estándar y la de bajo consumo MPLP y solo se aprecian algunas diferencias en la versión FPLP debido al uso de lógica combinacional adicional y mayor número de interconexionado. Esa lógica consistente en un mayor número de multiplexores y celdas combinacionales que se implementan como LUTS. El uso de *Slices* es por tanto algo mayor en la versión FPLP, concretamente un 2,6% de incremento en el número de *Slices* y un 1,32% en el número de LUTs con respecto a las versiones estándar y MPLP.

En el dispositivo *Virtex-5*, véase la Tabla 3.20, los *Slices* ocupados en la versión FPLP

y MPLP son un 4% más que en la versión estándar. El número de LUTs usados en la versión FPLP se incrementa un 1,3% en comparación con la estándar. A pesar de estas pequeñas diferencias de recursos, el incremento en lógica en nuestras propuestas de bajo consumo es prácticamente despreciable, como ocurría en las tecnologías ASIC.

Tabla 3.20 Informe de los recursos implementación en *Virtex-5*.

Utilización de la lógica XC5vlx20t-2ff323	**TRIVIUM**	**TRIVIUM MPLP**	**TRIVIUM FPLP**
Slice Flip Flops	288	288	290
Slices LUTs	292	294	296
Slices ocupadas	74	77	77
LUT Flip Flop	292	294	298
BUFG/BUFGCTRLs	1	2	2
DCM ADV	1	1	1

3.5.2. Consumo de potencia dinámico en FPGAs

El informe de consumo de potencia presenta la potencia dinámica para cada recurso usado en el diseño. Las Tabla 3.21 y Tabla 3.22 muestran los resultados del consumo de potencia en los dispositivos *Spartan-3E* y *Virtex-5*. La información en estas tablas está agrupada por tipo de recursos: consumo de los relojes para el árbol de reloj del diseño, consumo de la lógica para los elementos del *Slice* y consumo de las interconexiones.

El consumo de las entradas y salidas a través de los *pads* de la FPGA no son mostrados en estas tablas debido a que se trata de prototipos de circuitos para aplicaciones embebidas en la FPGA en las que las entradas y salidas del bloque se van a conectar a otros módulos internos de la FPGA con lo que el dato de consumo final se va a referir como bloque IP.

Tabla 3.21 Informe del consumo de potencia dinámica en *Spartan-3E* con XPA.

Potencia (mW)-25MHz XC3s500e-fg320	**TRIVIUM**	**TRIVIUM MPLP**	**TRIVIUM FPLP**	**Mejora*** **MPLP**	**Mejora*** **FPLP**
Reloj	2,05	1,91	1,16	7%	43%
Lógica	1,86	1,22	1,0	34%	46%
Interconexionado	1,2	0,87	0,72	28%	40%
Total	5,11	4,0	2,88	22%	44%

*$(Mejora = \%(P_{estandar} - P_{FPLP-MPLP})/P_{estandar})$.

La principal contribución en el consumo dinámico son las señales/líneas de reloj, siendo mayor el consumo en los dispositivos *Virtex-5* que en las *Spartan-3E*. La implementación FPLP consigue los mejores resultados en la reducción de la contribución del reloj, con un 43% en *Spartan-3E* y un 55% en *Virtex-5*.

Tabla 3.22 Informe del consumo de potencia dinámica en *Virtex-5* con XPA.

Potencia (mW)-25MHz XC5vlx20t-2ff323	**TRIVIUM**	**TRIVIUM MPLP**	**TRIVIUM FPLP**	**Mejora* MPLP**	**Mejora* FPLP**
Reloj	7,65	5,43	3,46	29%	55%
Lógica	1,19	0,84	0,65	29%	45%
Interconexionado	0,54	0,45	0,34	17%	37%
Total	9,38	6,72	4,45	28%	53%

*$(Mejora = \%(P_{estandar} - P_{FPLP-MPLP})/P_{estandar})$.

Cuando se comparan los consumos de potencia de las tres implementaciones, se aprecia que la versión de bajo consumo FPLP obtiene los mejores resultados en cuanto a reducción de potencia.

En la Figura 3.19 se muestran estos resultados mediante un diagrama de barras con los porcentajes de mejora en los consumos para comparar las versiones. El consumo en los dispositivos FPGA *Spartan-3E* es un 44% menor para la versión FPLP comparada con la estándar y esta mejora es incluso mayor con los dispositivos FPGA de la familia *Virtex-5* donde se alcanza una reducción del 53% con mejoras en el consumo del reloj, lógica e interconexionado.

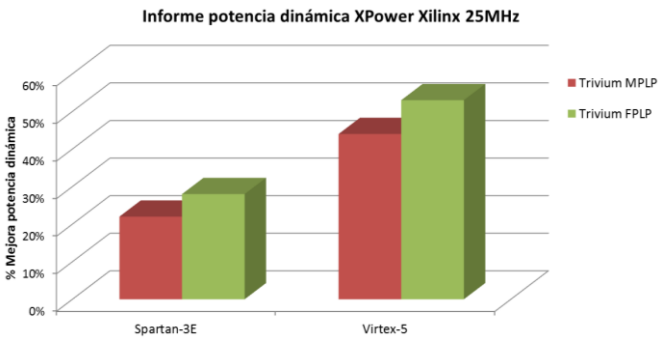


Figura 3.19 Gráfica de la mejora en el consumo de potencia en los Trivium MPLP-FPLP con la herramienta XPA.

Estos datos demuestran que la técnica propuesta de paralelización es también muy eficiente en las implementaciones Trivium en FPGA. En efecto, los resultados obtenidos muestran reducciones entre el 22% y 53% dependiendo de la familia FPGA y versión de Trivium usada, sin una penalización considerable en cuanto a recursos internos utilizados por la FPGA. La versión FPLP mejora entre 25 y 22

puntos el porcentaje de mejora de consumo que logra la versión MPLP.

3.6. Conclusiones

En este capítulo se ha presentado la técnica de paralelización como la mejor opción para la reducción en el consumo de potencia en los cifradores de flujo Trivium ya que su arquitectura interna resulta muy adecuada para su aplicación. Para ello se ha realizado un estudio de consumo en registros de desplazamiento con simulaciones eléctricas y lógicas apuntando en ambos casos a la importante reducción en el consumo que introduce la paralelización.

A continuación se han presentado dos propuestas de cifradores de flujo Trivium con arquitecturas eficientes para bajo consumo aplicando la técnica de paralelización denominadas MPLP y FPLP. Estas propuestas de cifradores de flujo Trivium con y sin arquitecturas de bajo consumo se han implementado con salida de la secuencia cifrante de un bit en una tecnología de 350 nm, para obtener resultados de recursos, área y consumo de potencia a partir de simulaciones lógicas y eléctricas verificando y comparando los resultados obtenidos después de la síntesis.

De los resultados de la síntesis se concluye que los cifradores Trivium FPLP tienen más celdas lógicas y más área combinacional que las otras propuestas de cifradores Trivium. La penalización en área de las versiones de bajo consumo alcanza el 2-5% respecto a la versión estándar.

En cuanto a los datos del consumo de potencia, en los cifradores de flujo de 1 bit de salida, los resultados de las simulaciones lógicas muestran mejoras generales en el consumo dinámico del orden del 34-50% dependiendo de la versión Trivium elegida, siendo mejor el caso FPLP. Los datos de consumo de potencia aportados por las simulaciones a nivel eléctrico confirman los datos de las simulaciones lógicas y demuestran que las versiones de bajo consumo Trivium consiguen importantes reducciones en el consumo de potencia (entre el 40-66%) dependiendo de la versión de bajo consumo y picos de corriente en los flancos de reloj de menor amplitud (entre el 40-48% en los flancos de subida y 67-83% en los de bajada). También confirma que la versión FPLP obtiene reducciones mayores.

Asimismo, se han implementado estas propuestas de cifradores de flujo Trivium con y sin arquitecturas de bajo consumo con salida de la secuencia cifrante de más de un bit, en concreto dos, ocho y dieciséis bits, en la misma tecnología de 350 nm, para obtener resultados de recursos, área y consumo de potencia a partir de simulaciones lógicas, verificando y comparando los resultados obtenidos después de la síntesis. Se obtiene que los recursos consumidos crecen significativamente con el aumento de bits de salida. Así, por ejemplo, el aumento de área en la versión

estándar de 16 bits es un 23% en estándar, un 17% para la versión MPLP y un 40% para la versión FPLP.

Las propuestas Trivium de bajo consumo MPLP y FPLP mantienen una buena relación en el número de celdas e interconexiones frente a la versión estándar en las propuestas de un bit y dos bits a la salida, pero a partir de ocho y dieciséis bits la diferencia se hace mucho mayor, sobre todo en la versión FPLP de dieciséis bits donde hay una clara diferencia de recursos y área.

Los cifradores Trivium FPLP tienen más celdas lógicas y más área combinacional que las otras propuestas de cifradores Trivium. Por su parte, la versión Trivium MPLP, es la que mejor evoluciona cuando aumenta el número de bits de salida.

En cuanto al consumo tras la síntesis, los datos son coincidentes en verificar que las versiones de bajo consumo Trivium tienen siempre menor consumo que la estándar excepto para las versiones de 16 bits. Las propuestas de bajo consumo Trivium MPLP y FPLP tienen mejoras en el consumo de potencia frente a la versión estándar entre el 27 y el 43% en las versiones con 1 y 2 bits de salida, siendo algo menor (entre 14-27%) en las versiones de bajo consumo con 8 bits de salida. Sin embargo las versiones de bajo consumo de 16 bits de salida no consiguen mejorar el consumo de la versión estándar, lo cual es debido a su mayor número de celdas e interconexiones, teniendo una penalización del 13-15% aproximadamente.

Por otro lado, se ha estudiado el consumo total dinámico frente a la escala de integración, mediante la síntesis y simulación del cifrador de flujo Trivium sin y con arquitecturas de bajo consumo (versión FPLP) de 1 bit de salida en tecnologías de 180 nm, 130 nm y 90 nm para obtener resultados de área y consumo de potencia a partir de simulaciones lógicas. Cuando se compara el consumo de potencia de las versiones de bajo consumo en estas tres tecnologías la versión FPLP tiene siempre un menor consumo de potencia que la versión estándar, un 30% menor en 180 nm, un 31% menor en 130 nm y un 17 % menor en 90 nm.

Finalmente se han implementado en dispositivos programables FPGA de Xilinx (familias *Spartan-3E* y *Virtex-5*) versiones estándar, MPLP y FPLP de los cifradores de flujo Trivium con salida de la secuencia cifrante de un bit y se ha obtenido resultados de los recursos utilizados y del consumo de potencia. Los resultados obtenidos muestran reducciones entre el 22% y 53% dependiendo de la familia FPGA y versión del cifrador Trivium usado, sin una penalización considerable en cuanto a recursos internos utilizados por la FPGA (menor del 3%).

La implementación FPLP consigue los mejores resultados, con un 44% de reducción del consumo en *Spartan-3E* y un 53% en *Virtex-5*, mientras que la implementación MPLP obtiene un 22% en *Spartan-3E* y un 28% en *Virtex-5*.

4. INTEGRACIÓN DE UN PROTOTIPO ASIC

Una vez que se han verificado y analizado funcionalmente en diferentes tecnologías las propuestas Trivium de bajo consumo se presenta el trabajo realizado para la integración de estas propuestas en un circuito integrado, al cual hemos llamado CITIES. Este es un ASIC (*Application-Specific Integrated Circuit*) diseñado para verificar y caracterizar en el laboratorio las versiones de las propuestas del cifrador Trivium con y sin arquitecturas de bajo consumo. Este circuito contiene además otros bloques digitales de funcionalidad diversa orientados para aplicaciones criptográficas que no son objeto de esta tesis, pero que utilizan los cifradores Trivium aquí presentados.

Con este circuito integrado se han buscado varios objetivos. En primer lugar verificar y caracterizar el funcionamiento de las versiones de cifradores de bajo consumo Trivium con las arquitecturas propuestas y compararlas con la estándar. En segundo lugar disponer de cifradores Trivium que puedan ser utilizados en ejemplos prácticos de comunicación criptográfica. Teniendo en cuenta estos objetivos se escogieron las opciones de implementación y encapsulado óptimas para desarrollar el circuito integrado final.

Se eligió la tecnología CMOS de 90 nm de TSMC (*Taiwan Semiconductor Manufacturing Company Limited*) para implementar el prototipo ASIC. Se trata de una tecnología ampliamente utilizada con éxito en otros proyectos con implementaciones de circuitos integrados como por ejemplo los proyectos CriptoBio [79] y MOBY-DIC [80]. Además tiene un buen compromiso entre prestaciones y precio de integración.

A continuación en el primer apartado se van a describir los aspectos más importantes de la descripción del prototipo ASIC CITIES que son englobados bajo el módulo llamado *uut_bt*. El segundo apartado trata del flujo de diseño digital empleado para la realización del ASIC. En el tercer apartado se presentan las tareas de la etapa de diseño lógico junto con los resultados de área y consumo de potencia obtenidos en esta etapa, mientras que en el apartado cuarto se llevan a cabo las de la implementación física (*layout*) del circuito integrado CITIES en su conjunto.

Finalmente en el quinto y último apartado se estudia el consumo de potencia tras la finalización de la implementación física.

4.1. Descripción del prototipo ASIC CITIES

El circuito CITIES integra como ya se ha comentado las versiones estándar y de bajo consumo del cifrador Trivium bajo el módulo *uut_bt*, además de otros bloques (no descritos en esta tesis) para aplicaciones de comunicación criptográfica y análisis de fallos que utilizan los cifradores Trivium propuestos en esta tesis.

El módulo de más alta jerarquía que representa al circuito CITIES se presenta en la Figura 4.1 donde se incluyen solamente las entradas y salidas correspondientes al módulo *uut_bt*.

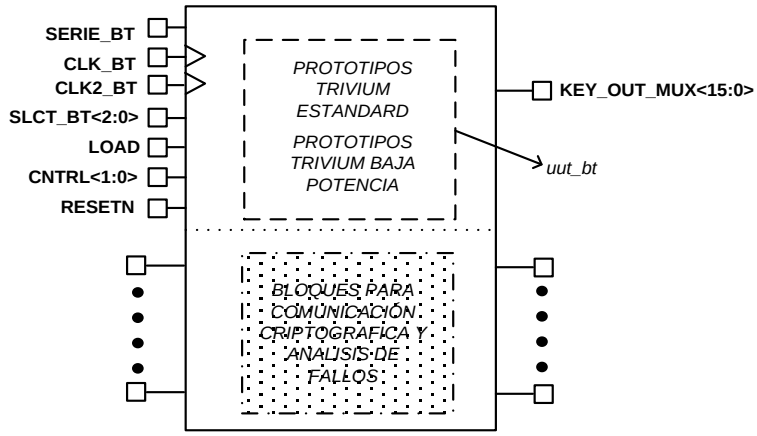


Figura 4.1 Circuito ASIC CITIES.

La descripción de estas entradas y salidas se presenta y describe en la Tabla 4.1. Como ya se ha comentado, solo se describen las relacionadas con el módulo *uut_bt*, que contiene las diferentes versiones del cifrador Trivium.

Tabla 4.1 Entradas/Salidas ASIC CITIES para los cifradores Trivium.

Nombre	Dirección	Descripción y comentarios
RESETN	Entrada	Señal de inicialización del circuito. Activa a nivel bajo.
CNTRL<1:0>	Entrada	Bus de entrada de 2 bits que configura el modo de funcionamiento de los cifradores Trivium.
LOAD	Entrada	Señal de habilitación de la carga en serie. Activa a nivel alto.
CLK_BT	Entrada	Reloj principal de los cifradores Trivium.
CLK2_BT	Entrada	Reloj de carga para la entrada serie (*SERIE_BT*).
SERIE_BT	Entrada	Entrada serie de datos, para la carga de IV<79:0>, KEY<79:0> y habilitadores de reloj (12 bits).
SLCT_BT<2:0>	Entrada	Bus de entrada de 3 bits de multiplexación de las salidas de los cifradores Trivium.
KEY_OUT_MUX<15:0>	Salida	Bus de salida de 16 bits con las salidas de los cifradores Trivium multiplexados por *SLCT_BT*.

El diseño del módulo *uut_bt* se ha particionado en varios módulos con jerarquía propia que se muestran en la Figura 4.2.

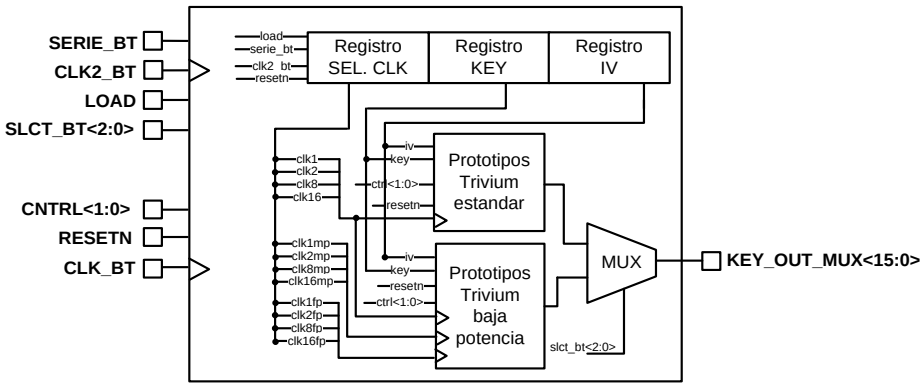


Figura 4.2 Esquema del módulo *uut_bt.*

Por un lado se tiene el conjunto de cifradores Trivium que se van a implementar con las propuestas en su versión estándar y por otro lado los de bajo consumo. En total se han incluido doce versiones de cifradores Trivium en el módulo *uut_bt.*

Un esquema descriptivo del contenido de todas las propuestas Trivium se muestra en la Figura 4.3 correspondiendo a:

- 4 propuestas Trivium en su versión estándar correspondientes a salidas con 1 bit, 2 bits, 8 bits y 16 bits.
- 8 propuestas Trivium en su versión de bajo consumo de potencia correspondiente a la paralelización completa (FPLP) y a la paralelización mixta (MPLP) con salidas de 1 bit, 2 bits, 8 bits y 16 bits.

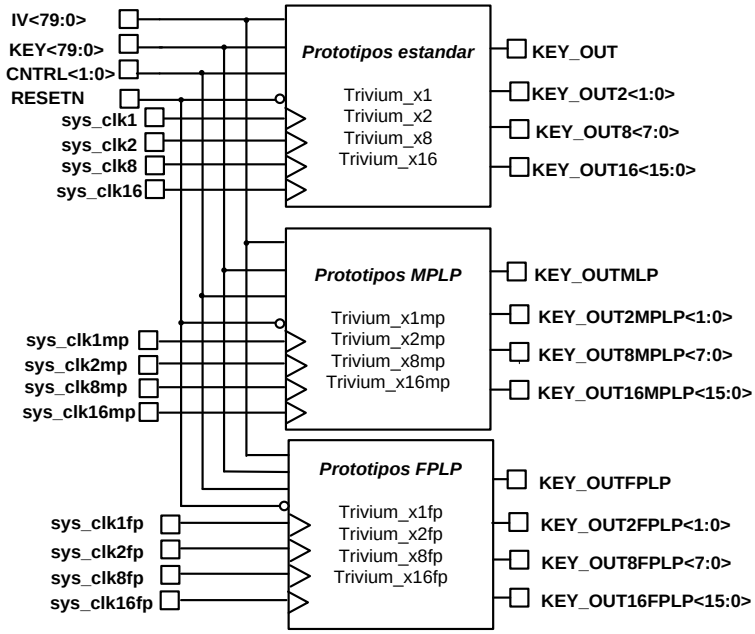


Figura 4.3 Esquema de las diferentes propuestas Trivium.

Para evitar un alto consumo en el número de pines de entrada y salida se decidió implementar la carga de los valores que necesitan los cifradores Trivium, clave y vector de inicialización, de forma serie mediante la señal de entrada *SERIE_BT*. Las entradas de control, *CNTRL<1:0>* son comunes a todos los Triviums. Las salidas de las diferentes versiones Trivium se multiplexan con la entrada de selección denominada *SLCT_BT<2:0>*.

En la Tabla 4.2 se presenta la correspondencia de la salida del circuito integrado *KEY_OUT_MUX<15:0>* con las de las diferentes versiones Trivium, en función de la entrada de selección *SLCT_BT<2:0>*.

Tabla 4.2 Selección de las salidas con las versiones Trivium.

SLCT_BT<2:0>	KEY_OUT_MUX<15:0>
000	Señales procedentes de otros bloques
001	key_out**&**key_outfplp**&**key_outmplp**&**"0000000000000"
010	"00"**&**key_out2**&**key_out2mplp**&**key_out2fplp**&**key_out8mplp
011	key_out8**&**key_out8fplp
100	key_out8fplp**&**key_out8mplp
101	key_out16
110	key_out16mplp
111	key_out16fplp

Los relojes de cada uno de los cifradores implementados se pueden deshabilitar individualmente mediante un registro de control de 12 bits. Con esta opción se ha pretendido tener la opción de activar o no cada uno de los prototipos Trivium y de esta forma posibilitar la medida del consumo de potencia de cada uno de ellos. La palabra de control se carga de forma serie junto con la clave y vector de inicialización.

Un cronograma del mecanismo de carga en serie de los bits de selección de reloj, vector de inicialización y de la llave se puede ver en la Figura 4.4.

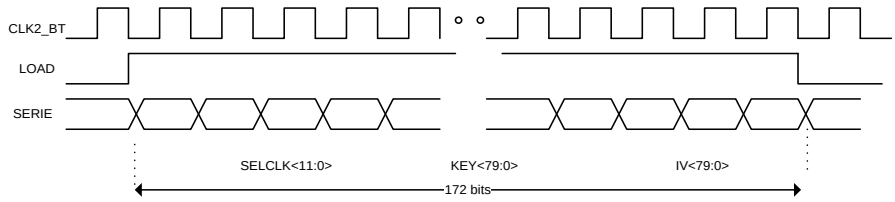


Figura 4.4 Cronograma de la carga serie.

Se tiene un reloj externo *CLK2_BT*, diferente al principal de los cifradores (*CLK_BT*), para realizar esta operación y una señal *LOAD* (activa a nivel alto) para activar la carga. La carga en serie conlleva 172 ciclos de reloj, 80 para la carga de la clave, 80 para el vector de inicialización y 12 para el registro de control de los relojes, siendo el orden de carga como se muestra en el cronograma, primero el vector de inicialización, después la clave y por último la selección de los relojes:

La selección de los diferentes relojes de las versiones Trivium se realiza mediante una palabra de 12 bits del registro serie de entrada. La Tabla 4.3 muestra el contenido de estos bits y el reloj que activa cada uno de estos bits junto con la versión Trivium en la que actúa. Si se quiere activar por ejemplo el reloj de la versión estándar Trivium de 2 bits se deberá cargar en la palabra de selección de relojes (*SELCLK<11:0>)* una palabra tal como <000000001000>, estando el bit

SELCLK<3> a valor lógico 1. Si por el contario se quiere activar todos los relojes se deberá cargar <111111111111>.

Tabla 4.3 Selección de los diferentes relojes de las versiones Trivium.

SELCLK<11:0>	Reloj CLK	Versiones TRIVIUM
SELCLK<0>	sys_clk1	Trivium_x1
SELCLK<1>	sys_clk1fp	Trivium_x1fp
SELCLK<2>	sys_clk1mp	Trivium_x1mp
SELCLK<3>	sys_clk2	Trivium_x2
SELCLK<4>	sys_clk2fp	Trivium_x2fp
SELCLK<5>	sys_clk2mp	Trivium_x2mp
SELCLK<6>	sys_clk8	Trivium_x8
SELCLK<7>	sys_clk8fp	Trivium_x8fp
SELCLK<8>	sys_clk8mp	Trivium_x8mp
SELCLK<9>	sys_clk16	Trivium_x16
SELCLK<10>	sys_clk16fp	Trivium_x16fp
SELCLK<11>	sys_clk16mp	Trivium_x16mp

4.2. Flujo de diseño digital

Para realizar un circuito integrado con perspectiva de éxito es absolutamente necesario planificar y ordenar el ciclo de desarrollo del ASIC. Esto es lo que hace una metodología de diseño. En nuestro caso el circuito integrado se ha implementado siguiendo una metodología de diseño *semi-custom* con herramientas de diseño comerciales tal como se representa en la Figura 4.5.

El proceso comienza con una fase de diseño lógico también llamada *front-end*, basada en el uso de lenguajes de descripción hardware (HDL) para la descripción y especificación del circuito a implementar. Como ya se ha visto en capítulos anteriores el circuito ha sido descrito en VHDL. Tras la verificación de estas descripciones en código a nivel RT, los restantes pasos del *front-end* son la síntesis lógica a nivel de celdas de librería y su verificación funcional y temporal [8].

A continuación se tiene la fase de implementación física o *back-end*, donde se realiza el emplazamiento, la creación de los anillos de alimentación y la generación del árbol o árboles de reloj. Posteriormente se realiza el conexionado de todas las celdas llegando a la implementación física o *layout*. Tras la verificación de las reglas de diseño del fabricante en el *layout*, se genera una descripción post-*layout* en formato *Verilog* y se procede a su verificación funcional y temporal con los parásitos obtenidos de la implementación física empleándose los mismos test que se usaron para la verificación funcional del código VHDL, junto con algunos adicionales dedicados a verificaciones más particulares.

Al final de todo este proceso se obtuvo la implementación física del prototipo CITIES sin el anillo de *pads* de entrada/salida y polarizaciones. En esta tecnología el fabricante proporciona solamente *pads* bidireccionales y su configuración como entrada o salida debe ser hecha por el diseñador. En nuestro caso, estos *pads* de entrada y salida ya fueron configurados e incorporados a la librería de diseño en otros diseños ya fabricados con esta tecnología. Con ellos se generó un anillo eligiendo los *pads* adecuados a cada una de las entradas y salidas, además de las polarizaciones. El conexionado y emplazamiento final con los *pads* se realizó manualmente (*full-custom*). Por último, se realizaron las verificaciones descritas por el fabricante (DRC), además de chequear la similitud entre la descripción a nivel de esquemático del circuito y el *layout* (LVS). Finalmente se generó el fichero *GDSII* para su envío a fabricación.

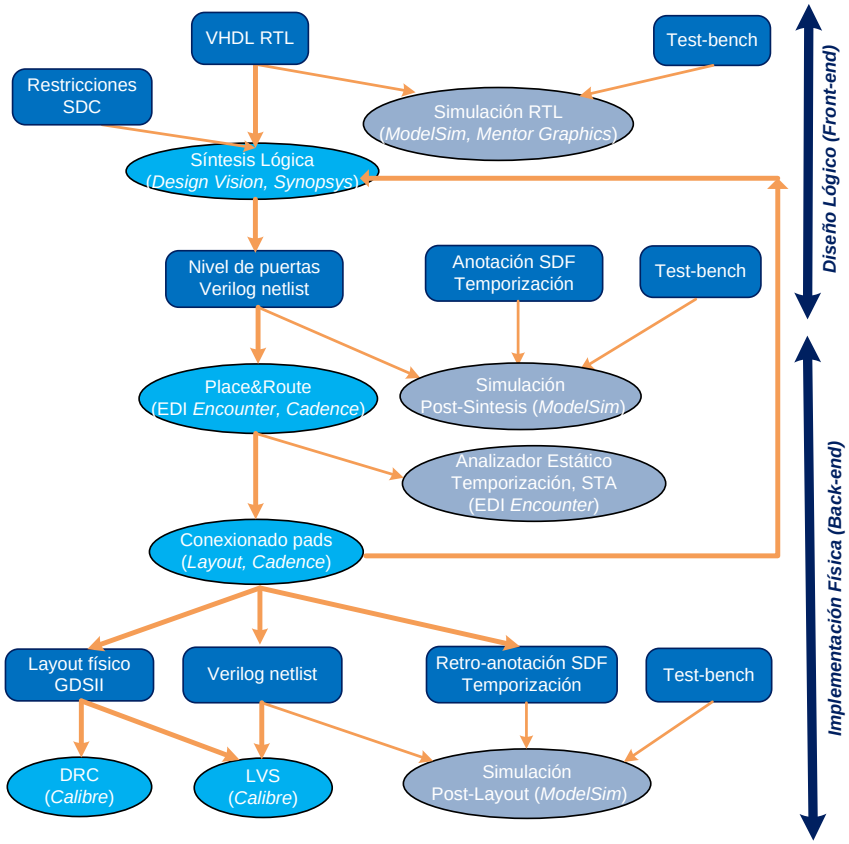


Figura 4.5 Flujo de diseño digital.

Las herramientas de CAD utilizadas en estas etapas han sido, para la simulación RTL, post-síntesis y post-layout, *ModelSim SE 6.6d* de *Mentor Graphics*. El proceso de síntesis lógica se ha realizado con la herramienta *Design Vision G-2012.06-SP4* de *Synopsys*. Durante el proceso de diseño se añadieron algunas restricciones temporales como son la frecuencia de reloj o los retrasos entre algunas señales, entre otras. La descripción del circuito a nivel de puertas (*netlist*) resultante del proceso de síntesis se trasladó a la herramienta *Encounter Digital Implementation (*EDI*) System RTL-to-GDSII* de *Cadence* donde se llevó a cabo la etapa de emplazamiento y conexionado. Dentro del entorno de la herramienta de *Cadence* también se utilizó la herramienta de *layout Virtuoso* para el conexionado final con los *pads*. Para las verificaciones finales (DRC y LVS) se usó *Calibre* de *Mentor Graphics*.

4.3. Diseño lógico

A continuación se presentan las diversas tareas que conforman la etapa de diseño lógico, que engloba la descripción de los diferentes módulos en lenguaje de descripción hardware, la síntesis lógica y su verificación funcional y temporal así como los resultados obtenidos. La descripción en lenguaje VHDL a nivel RT de las diferentes versiones del cifrador Trivium se ha presentado en los Capítulos 2 y 3, las cuales tomamos como punto de partida.

El circuito que ha sido sintetizado incluye todas las arquitecturas descritas en código VHDL que pueden consultarse en el Apéndice A y que describen el funcionamiento de las diferentes propuestas de cifradores Trivium que se han diseñado y caracterizado. Todos ellos han sido incluidos en el módulo de jerarquía mayor (*uut_bt*) comentado previamente en el apartado 4.1.

4.3.1. Síntesis Lógica

Para el proceso de síntesis se ha utilizado un fichero de texto (*script*) conteniendo comandos de la herramienta, que se ejecutan línea por línea, y que se encarga de leer todos los códigos en VHDL, definir los relojes principales y los derivados de ellos, como los divididos por dos de los Trivium de bajo consumo, cargar las librerías de temporización y las condiciones ambientales, realizar la síntesis, guardar los resultados, generar los ficheros de retardos de propagación entre señales (retro-anotaciones) en formato sdf y los informes de área, temporización, potencia, etc. Este fichero se puede ver en el Apéndice B de la Tesis.

El informe de la herramienta de síntesis de *Synopsys* nos muestra los siguientes datos del ASIC CITIES:

- Número de puertos: 48
- Área combinacional: 91.116,25 μm²
- Área Inversores-Buffers: 3099,70 μm²
- Área no combinacional: 160.195,90 μm²
- Área total de celdas: 251.312,14 μm²

El área correspondiente del módulo *uut_bt* es de 89.686,70 μm² (35% del área total).

La Tabla 4.4 muestra el número de celdas de referencia usadas, el área ocupada y el número de celdas lógicas de cada uno de los cifradores Trivium diseñados tras la síntesis. También muestra el área de la lógica secuencial (no combinacional) y de la lógica combinacional. Además para analizar los resultados, se muestra en la Tabla 3.14 los incrementos (%) de cada medida respecto a la versión Trivium de 1 bit correspondiente. Para apreciar mejor los detalles del informe de la herramienta, los datos se han representado también gráficamente en la Figura 4.6.

Tabla 4.4 Área y número de celdas del informe post-síntesis de *Synopsys*.

Propuestas Trivium	Librería: TSMC 90nm				
	Celdas de Referencia	Área (μm²)	Celdas Lógicas	Área No Combinacional (μm²)	Área Combinacional (μm²)
Trivium_x1	18	6360,94	625	4390,24	1970,74
Trivium_x1mp	25	6603,00	665	4474,92	2128,09
Trivium_x1fp	26	6750,47	620	4632,97	2117,51
Trivium_x2	24	6428,72	631	4387,42	2041,30
Trivium_x2mp	28	6646,04	646	4545,48	2100,57
Trivium_x2fp	33	6898,65	656	4628,74	2268,92
Trivium_x8	24	6851,37	723	4367,66	2483,71
Trivium_x8mp	25	7070,81	791	4511,61	2559,21
Trivium_x8fp	38	7712,91	873	4736,69	2976,22
Trivium_x16	26	7425,73	855	4410,00	3015,73
Trivium_x16mp	26	7637,41	979	4494,67	3142,74
Trivium_x16fp	30	8992,87	1147	5063,39	3929,49

Cuando se analizan los resultados se observa que los recursos consumidos crecen significativamente con el aumento de bits de salida, como también se había apreciado en el análisis del Capítulo 3, epígrafe 3.3.3.1. Así, por ejemplo y tal como se muestra en la Tabla 3.14, el número de celdas en la versión estándar de 16 bits es un 37% mayor que en la versión de 1 bit, ocurriendo algo similar en la versión MPLP (con un aumento del 47%) y en la versión FPLP (aumento del 85%). Incrementos semejantes ocurren, aunque con menor medida, en el área total (17% en estándar, 16% en MPLP y 33% en FPLP). El tipo y número de celdas a utilizar en un diseño por la herramienta de síntesis varía de una tecnología a otra. Es por ello que los datos del número de celdas difieran algo de los del Capítulo 3 (tecnología

AMS 350 nm). Sin embargo los porcentajes referidos al área total si son más acordes entre ambas tecnologías. La diferencia mayor se tiene en el área combinacional debido sobre todo a la librería de celdas de cada tecnología.

Tabla 4.5 Porcentajes de área y número de celdas, respecto a la version "x1".

Propuestas Trivium	Librería: TSMC 90			
	Nº celdas	Área Total	Área Combinacional	Área No Combinacional
Trivium_x1	-	-	-	-
Trivium_x1mp	-	-	-	-
Trivium_x1fp	-	-	-	-
Trivium_x2	1%	1%	4%	0%
Trivium_x2mp	-3%	1%	-1%	2%
Trivium_x2fp	6%	2%	7%	0%
Trivium_x8	16%	8%	26%	-1%
Trivium_x8mp	19%	7%	20%	1%
Trivium_x8fp	41%	14%	41%	2%
Trivium_x16	37%	17%	53%	0%
Trivium_x16mp	47%	16%	48%	0%
Trivium_x16fp	85%	33%	86%	9%

*$(\% = \%(M_{estandar} - M_{FPLP-MPLP})/M_{estandar})$.

Los cifradores Trivium FPLP tienen más celdas lógicas y más área combinacional que las otras propuestas de cifradores Trivium. Por su parte, el área no combinacional es parecido en todas las versiones excepto en la versión FPLP de 16 bits (aumenta un 9%) debido al aumento en el número de biestables en algunos registros paralelos par e impar. Lo más llamativo es el aumento en el área de las celdas combinacionales que aparece en las versiones FPLP de ocho y dieciséis bits, con un aumento del 41% y del 86% respectivamente. Esto es así debido al gran número de bits que hay que realimentar y operar lógicamente, lo que hace que aumente en gran medida el número de celdas combinacionales.

Por su parte, la versión Trivium MPLP, que es la que mejor evoluciona con el número de bits de salida, obtiene en la síntesis de Synopsys menos celdas combinacionales que las otras dos versiones. No obstante su área se incrementa, para el caso multi-bit (sea 2, 8 o 16), más que en la versión Trivium estándar.

Las diferencias entre las diferentes versiones estándar y paralelizadas a nivel del área total son mostradas gráficamente también en la Figura 4.6, con las mismas conclusiones.

Las propuestas Trivium de bajo consumo MPLP y FPLP, mantienen una buena relación en el número de celdas frente a la versión estándar en las propuestas de un bit y dos bits a la salida, dando un área muy parecida cuando se comparan entre ellas, pero a partir de ocho y dieciséis bits la diferencia se hace mucho mayor, sobre

todo en la versión FPLP de dieciséis bits donde hay una clara diferencia de recursos y área.

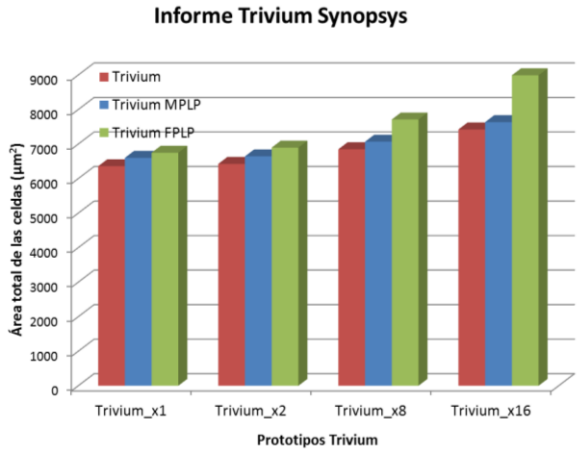


Figura 4.6 Gráfica post-síntesis del área total de las versiones Trivium.

4.3.2. Simulaciones y verificaciones post-síntesis

Las verificaciones cumplen un doble objetivo, asegurar el correcto diseño del circuito a nivel funcional y verificar su correcto funcionamiento temporal. En flujos de diseño de circuitos integrados digitales realizados a partir de descripciones HDL, el mayor esfuerzo de verificación ha de realizarse sobre las propias descripciones HDL a nivel RT. Sobre las siguientes etapas se requiere menores esfuerzos de verificación, debido a que hasta ellas se llega mediante procesos semiautomáticos dentro de entornos de diseño. Sin embargo es importante no saltarse ninguna de estas etapas de simulación en el flujo de diseño.

En esta etapa del flujo de diseño de este prototipo se realizaron en primer lugar simulaciones individuales de los diferentes códigos Trivium por separado y que ya se presentaron en capítulos anteriores. A continuación se llevaron a cabo simulaciones funcionales y temporales del módulo *uut_bt*, que contiene a todas las versiones del cifrador Trivium y los bloques de carga de la clave y vector de inicialización.

Para la generación de los patrones de simulación de los cifradores Trivium se tiene una señal de reloj de 50 MHz y diferentes parejas de valores de la clave (*key*) y del vector de inicialización (*IV*), mostradas en la Tabla 4.6. Estos valores están basados en algunas de las parejas de valores dadas en la propuesta del cifrador Trivium [27].

Tabla 4.6 Parejas de valores de clave e inicialización para Trivium.

Pareja	key-IV	Pareja	key-IV
1	K => X"00000000000000000000" IV=> X"00000000000000000000"	5	K =>X"F6F7F8F9FAFBFCFDFEFF" IV=>X"00000000000000000000"
2	K => X"80000000000000000000" IV=> X"00000000000000000000"	6	K =>X"0A5DB00356A9FC4FA2F5" IV=>X"1F86ED54BB2289F057BE"
3	K =>X"0F62B5085BAE0154A7FA" IV=>X"288FF65DC42B92F960C7"	7	K => X"00000000000000000001" IV=>X"00000000000000000000"
4	K =>X"0102030405060708090A" IV=>X"00000000000000000000"	8	K => X"0A5DB00356A9FC4FA2F5" IV=>X"1F86ED54BB2289F057BE"

Se han llevado a cabo simulaciones en el diseño final estimulando la parte correspondiente al módulo *uut_bt* con diferentes parejas y comprobado que la salida es correcta. En el Apéndice B de la Tesis se presentan los ficheros de test utilizados en la simulación también llamados *testbench* Las simulaciones post-síntesis fueron ejecutadas en tres condiciones de simulación denominadas de peor caso (WC), mejor caso (BC) y típica (TC). La diferencia entre éstas reside en los valores de retardos de las celdas de la librería tecnológica suministrada por el fabricante de la tecnología, en nuestro caso la *foundry* TSMC 90 nm. Los valores de estos tres condiciones se muestran en la Tabla 4.7. Se corresponden con diferentes valores en los parámetros de alimentación y temperatura.

Tabla 4.7 Condiciones caracterización de la tecnología TSMC 90nm.

Nombre	Parámetro	WC	TC	BC	Unidades
VDD_{CORE}	Alimentación celdas estándar	1,08	1,2	1,32	V
VDD_{IO}	Alimentación pads IO	2,25	2,5	2,75	V
T	Temperatura	125	25	0	°C

4.3.3. Datos post-síntesis del consumo de potencia

Se han obtenido los datos del consumo promedio de potencia con la propia herramienta de síntesis de *Synopsys*. Para mejorar la precisión de los resultados se ha incluido un fichero con la actividad de conmutación de los nodos generado por la herramienta de simulación y utilizando el *netlist* a nivel de puertas generado tras la síntesis.

La medida del consumo de potencia es dependiente de los estímulos y es por ello importante hacer una adecuada elección de los patrones de vectores para obtener unos correctos resultados. En este caso se han llevado a cabo simulaciones post-síntesis utilizando unos patrones que activan todos los relojes internos de los bloques Trivium junto con el fichero de retrasos temporales generado con

anterioridad por la herramienta de síntesis. Este fichero de retro-anotaciones contiene la información sobre el retardo de las interconexiones y de las celdas en el caso típico. De la simulación se obtiene un fichero tipo vcd (*value charge dump*) que contiene el registro de todos los cambios de las señales en los nodos del circuito y que se usa para el cálculo del consumo de potencia.

En la Tabla 4.8 se muestran los datos del consumo de potencia post-síntesis, en condiciones típicas, generados por la herramienta de *Synopsys*, así como el porcentaje de mejora del consumo respecto a la versión estándar.

Estos datos de potencia son muy coincidentes aunque algo más optimistas que los obtenidos en el capítulo 3 con datos de otra tecnología de fabricación. Son coincidentes en demostrar que las versiones de bajo consumo Trivium tienen siempre menor consumo que la estándar pero difieren para la versión de dieciséis bits donde los datos obtenidos en el Capítulo 3 no ofrecen mejoras, mientras que con esta tecnología aún se tienen mejoras en el consumo.

Como cabía esperar, el consumo crece con el número de bits de salida. La mejora en la potencia total se consigue debido a la fuerte disminución producida en el consumo de potencia interno (Int.), que disminuye drásticamente en todas las versiones Trivium, aunque conforme aumenta el número de bits de salida esta disminución se hace algo menor.

La potencia dinámica de conmutación (Conm.) va aumentando progresivamente en todos los casos de los cifradores Trivium FPLP mientras que en los MPLP disminuye, salvo en la versión de dieciséis bits.

Tabla 4.8 Consumo de potencia post-sintesis

VDD=1,2 V-50MHz- Caso Típico							
Propuestas Trivium	**Consumo de Potencia**				**% Mejora Total**	**% Mejora Int.**	**% Mejora Conm.**
	µW		**nW**	**µW**			
	Int.	**Conm.**	**Fugas**	**Total**			
Trivium _x1	353	29,9	396,7	384			
Trivium _x1mp	232	20,9	409,8	253	34	34,3	30,1
Trivium _x1fp	178	32,7	418	212	44	49,6	-9,4
Trivium _x2	358	32,0	402,3	390			
Trivium _x2mp	233	22,8	411,4	257	34	34,9	28,8
Trivium _x2fp	184	37,8	427,4	222	43	48,6	-18,1
Trivium _x8	368	40,2	429,1	409			
Trivium _x8mp	276	38,7	436,5	315	22	25,0	3,7
Trivium _x8fp	201	59,8	461,5	262	35	45,4	-48.8
Trivium _x16	373	50,5	459,7	424		0,0	0,0
Trivium _x16mp	318	57,3	464,2	375	11	14,7	-13,5
Trivium_x16fp	256	98,4	540,8	355	16	31,4	-94,9

*$(Mejora = \%(P_{estandar} - P_{paralelizado})/P_{estandar})$.

Para apreciar mejor los detalles del consumo total de potencia, se ha representado gráficamente en la Figura 4.7. Los casos más favorables en cuanto a la mejora del consumo de potencia se dan en las versiones de menor paralelización en los bits de salida, siendo los casos de uno y dos bit de los Trivium FPLP y MPLP los que mejores resultados ofrezcan en su conjunto.

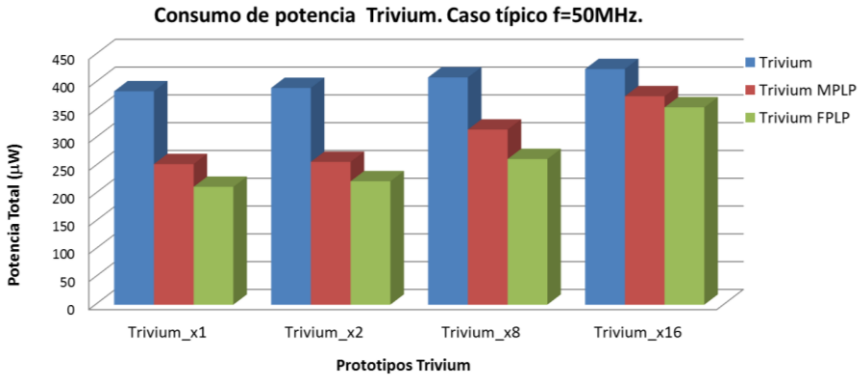


Figura 4.7 Consumo potencia total cifradores Trivium.

En la Figura 4.8 se presenta un diagrama de barras con los porcentajes de mejora en los consumos. Los porcentajes mostrados son positivos porque el consumo es menor que el del estándar. Obviamente el valor de mejora del estándar es 0. En la Figura 4.7 es inmediato apreciar el menor consumo de todas las propuestas de los cifradores Trivium MPLP y FPLP ("x1", "x2", "x8" y "x16").

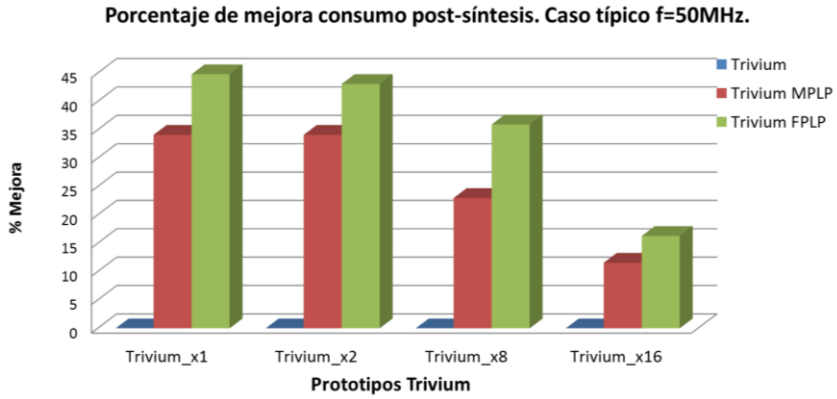


Figura 4.8 Porcentaje de mejora en el consumo post-síntesis.

Las versiones de los cifradores de bajo consumo de uno y dos bits permiten reducir de forma significativa el consumo de la versión estándar del Trivium (34-44%). En el caso de ocho bits, la reducción es algo menor aunque apreciable también (22-35%) mientras que para el caso de dieciséis bits esta mejora es poco apreciable. La

versión FPLP tiene siempre una mayor mejora de consumo respecto a la que logra la versión MPLP.

Una vez finalizada la etapa de diseño lógico se pasa a la implementación física en la cual también se va a realizar este cálculo.

4.4. Implementación física

La herramienta utilizada en este caso es *Encounter RTL-GDSII* de *Cadence* para la generación del *layout* sin los *pads* de entrada y salida y *Virtuoso* para generar el *layout* con los *pads*.

En el Apéndice B se muestran los ficheros de configuración y los *scripts* utilizados para la generación del *layout*. En los siguientes apartados se detalla el proceso seguido.

4.4.1. *Layout*: Colocación, árbol de reloj y conexionado del chip

Esta fase comienza utilizando la descripción del circuito a nivel de puertas generada tras la síntesis, los ficheros temporales de la librería y la descripción física de las celdas estándar para realizar un emplazamiento (*floorplanning*) automático de todos los bloques que forman el circuito integrado CITIES. Se añaden también las posiciones de los pines de entrada y salida en la periferia del área del circuito mediante un fichero de texto. El fichero de configuración que se muestra en el Apéndice B detalla todo este proceso.

Se muestra en la Figura 4.9 el resultado de esta fase, con el detalle del emplazamiento automático generado por la herramienta de los diferentes módulos del ASIC CITIES, con especial atención al bloque que contiene todas las versiones Trivium (*uut_bt*).

El siguiente paso es añadir el anillo principal de alimentación y tierra alrededor de las celdas del circuito CITIES y hacer el conexionado de los anillos con las alimentaciones y las tierras de las celdas estándar. Posteriormente se generan los canales de conexionado y los árboles de reloj de todos los relojes del circuito definidos por el diseñador.

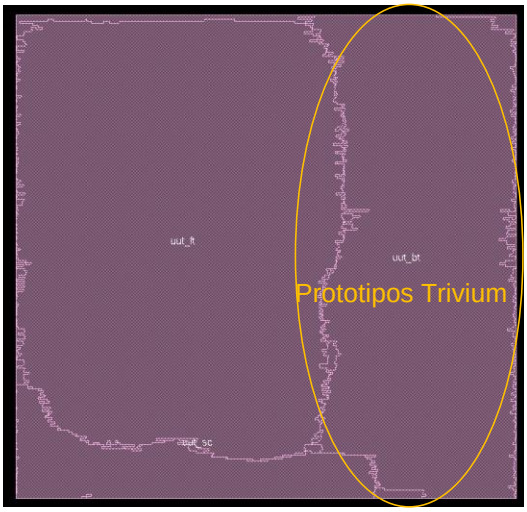


Figura 4.9 Colocación (*placement*) de los módulos del ASIC CITIES.

Un punto de vital importancia para conseguir la adecuada operación de las versiones Trivium es conseguir que las señales de reloj, y sus caminos de distribución, tengan un comportamiento adecuado a las exigencias temporales del circuito, evitando retardos indeseados. Los árboles de reloj son fundamentales tanto en las versiones MPLP como las FPLP. En el caso de la versión FPLP el reloj de los registros de desplazamiento es de frecuencia mitad al principal y el árbol de reloj generado se debe de adaptar a esta circunstancia. En el caso de la versión MPLP al tener registros de desplazamiento de frecuencia mitad y de frecuencia principal que se conectan entre sí el problema se agrava y la generación del árbol de reloj se hace más complicada ya que estos dos relojes (el principal y el dividido) no deben tener retrasos entre ellos, para que así el registro de desplazamiento total opere adecuadamente y no se tengan errores de temporización.

En las versiones Trívium de bajo consumo FPLP el reloj dividido es objeto de la generación de árboles diferentes a los del reloj principal por la herramienta, y que actúan sobre los biestables de los registros paralelizados.

Para los cifradores Trivium de bajo consumo MPLP sin embargo no es suficiente esta opción. Se ha tenido que generar dos árboles de reloj, uno para el reloj principal y otro para el dividido, pero con retraso idealmente nulo entre ellos para poder tener un correcto funcionamiento temporal. Para conseguir esto, se ha optado por tener en los bloques MPLP dos entradas de reloj, una la principal (*clk_bt*) y otra la del reloj dividido (*clkm*), generándose arboles de reloj para ambas señales.

Se ha utilizado un fichero de configuración (ver Apéndice B) para la generación de

todos los árboles de reloj. En este fichero se detallan las diferentes señales de reloj necesarias para construir los arboles de reloj así como sus dependencias. Este punto es de vital importancia para conseguir las exigencias de temporización del circuito y evitar retardos indeseados en las líneas de reloj que pueden llevar a un mal funcionamiento de los diversos cifradores Trívium

En la Tabla 4.9 se muestra un informe a nivel de celdas y profundidad de los diferentes árboles de reloj. Se tiene una rama que lleva el reloj dividido por dos (*clkm*) a todos los cifradores Trivium MPLP y 4 ramas de los relojes (*clk_fp*, *clk_x2fp*, *clkx8_fp*, *clkx16fp*) que van a los diferentes cifradores Trivium FPLP. Además hay una rama para el reloj de carga serie (*clk2_bt*) y otra para el reloj principal (*clk_bt*).

Tabla 4.9 Informe de los diferentes árboles de reloj para los cifradores Trivium.

Trivium	**Informe árbol de reloj**		
Árbol de reloj	**Total flip-flop**	**Sub-árboles**	**Descripción**
clk2_bt	184	44	CLK2_BT
clk_bt	2308	484	CLK_BT
clk_fp	290	59	CLK_BT/2
clk_x2fp	290	59	CLK_BT/2
clk_x8fp	297	75	CLK_BT/2
clk_x16fp	318	83	CLK_BT/2
clkm	628	126	CLK_BT/2

El *layout* implementado con celdas estándar de la librería tecnológica se muestra en la Figura 4.10. El tamaño total es de 689x670 μm^2, incluyendo el anillo de polarización y tierra que rodea a todas las celdas.

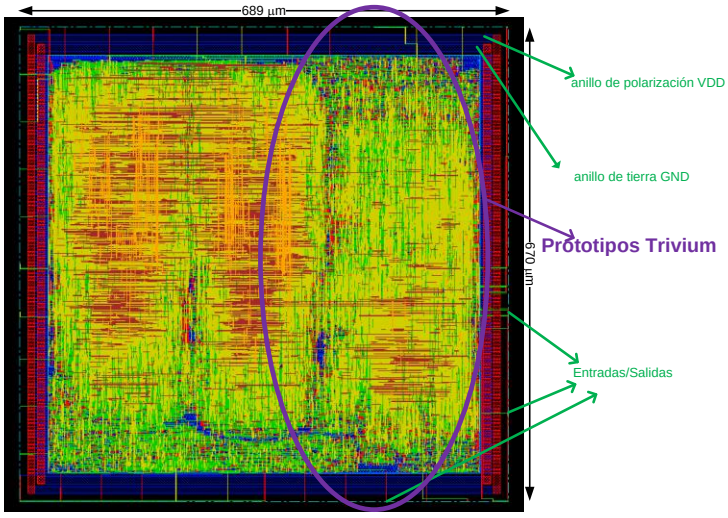


Figura 4.10 *Layout* del ASIC CITIES implementado con celdas estándar.

Tras la generación del *layout* hay que realizar un conjunto de verificaciones, tanto físicas como temporales. Para las verificaciones temporales se genera una descripción en formato *Verilog* y un fichero de retro-anotaciones con los retardos producidos por las líneas de interconexión del layout (formato sdf) que se van a utilizar para realizar simulaciones post-*layout*.

4.4.2. Verificación de diseño post-*layout*

Una vez finalizada la generación del *layout* se chequeó con la herramienta *Encounter* que no había errores de conexionado y de geometría en la implementación. También se comprobó que la implementación física concordaba con el esquemático del que provenía (LVS).

Posteriormente se volvió a simular el diseño con los mismos patrones de test que se utilizaron en las simulaciones post-síntesis pero añadiendo en esta ocasión el fichero de retrasos obtenido del *layout*, comprobando el correcto funcionamiento de cada uno de los prototipos Trivium. Se han utilizado las parejas de valores de llave y de vector de inicialización que se mostraron en la Tabla 4.6, los valores de selección de relojes de la Tabla 4.3 y de salida de los cifradores de la Tabla 4.2.

4.4.3. Colocación de los *pads* y conexionado final.

Finalmente se añadieron los *pads* de entrada, salida y alimentación al circuito implementado y se generó un esquemático en la herramienta de *Cadence*. La implementación física con el anillo de *pads* se hizo con la herramienta *Virtuoso* dentro del entorno de *Cadence*. Para esta operación se generó un *layout* con todos los *pads* necesarios, con un tamaño final de 1875 x 1875 μm^2, en una disposición de anillo y posteriormente se instanció el *layout* generado automáticamente por la herramienta de *Encounter* correspondiente al circuito CITIES.

Las interconexiones entre los *pads* de entrada y salida del circuito con las entradas y salidas del *layout* se realizaron manualmente, incluidas la alimentación y masa correspondiente al anillo de *pads* y las del anillo del núcleo o *core*.

El ASIC tiene un total de 60 *pads*, con una asignación de la siguiente forma:

- 8 *pads* para alimentación y tierra del anillo de los *pads*.
- 4 *pads* para alimentación y tierra del núcleo o *core.*
- 6 *pads* para señales de reloj.
- 1 *pad* para una señal de inicialización.
- 20 *pads* de propósito general de entrada.
- 21 *pads* de propósito general de salida.

Los pines de los *pads* del ASIC CITIES referentes a los cifradores Trivium, objeto de esta Tesis, se describen en la Tabla 4.10 y se muestran en la Figura 4.11 en el *layout* del circuito:

Tabla 4.10 *Pads* del módulo *uut_bt* en el ASIC CITIES.

Pads	Nombre	Descripción y comentarios
64,48,32,25	VSS_RING	Alimentación de tierra del anillo de *pads*.
50,34,18,8	VDD_RING	Alimentación positiva anillo *pads*. 2,5 V
42,1	VDD	Alimentación positiva *core*. 1,2 V
41,7	VSS	Alimentación de tierra del *core*.
63	RESETN	Entrada. *Pullup*.
62-61	CNTRL<1:0>	Entrada. *Pullup*.
60	LOAD	Entrada. *Pullup*.
59	CLK_BT	Entrada. *Pullup.* Reloj principal Trivium (~10-200 MHz).
58	CLK2_BT	Entrada. *Pullup.* Reloj de carga Trivium (~10-100 MHz).
57	SERIE_BT	Entrada. *Pullup.* Entrada serie de datos, para la carga de iv<79:0>, key<79:0> y habilitadores de reloj<11:0>.
56-54	SLCT_BT<2:0>	Entrada. *Pullup*.
53-51	KEY_OUT<0:2>	Bus salida multiplexado.
47-43	KEY_OUT<3:7>	Bus salida multiplexado.
40-35	KEY_OUT<8:13>	Bus salida multiplexado.
31-30	KEY_OUT<14:15>	Bus salida multiplexado
49,33,17,16	N.C	No conectados.
Otros	--	22 *pads* para otros propósitos.

En la Figura 4.11 se muestra el *layout* definitivo del ASIC CITIES generado una vez finalizada la operación de conexionado con los *pads*.

Solamente se muestran los *pads* usados por el módulo de los cifradores Trivium. Como se comentó anteriormente se tiene un anillo de pads de entrada, salida y alimentación en la periferia y en el centro se tiene instanciado el *layout* generado automáticamente por la herramienta de *Encounter* correspondiente al circuito CITIES.

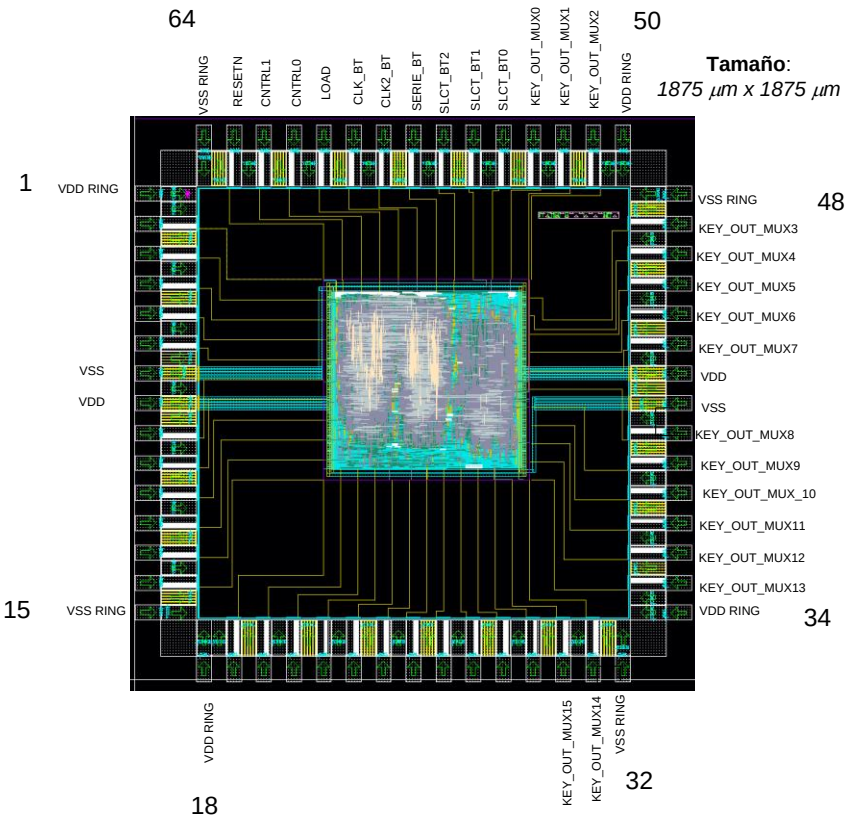


Figura 4.11 *Layout* con los *pads* del ASIC CITIES.

4.4.4. Verificación final

Por último se realizó el chequeo de las reglas de diseño del fabricante, DRC (*Design Rule Check*), chequeo de las reglas eléctricas, ERC (*Electrical Rule Check*) y comparación entre el esquemático y el *layout*, LVS (*Layout versus Schematic*) en el nivel más alto de jerarquía, que se corresponde con el circuito con los *pads* puestos. El resultado final fue un chequeo correcto de todas las reglas, sin ningún error y una comparación LVS sin errores.

4.4.5. Encapsulado

Se seleccionó un encapsulado cerámico CQF (*Ceramic Quad Flat Package*) con 64 pines, que es el más pequeño de entre los que cabría el ASIC y se propuso un conexionado (*bonding*) entre el encapsulado y el circuito integrado ASIC CITIES

como se representa en la Figura 4.12, en concordancia con lo especificado en la Tabla 4.10.

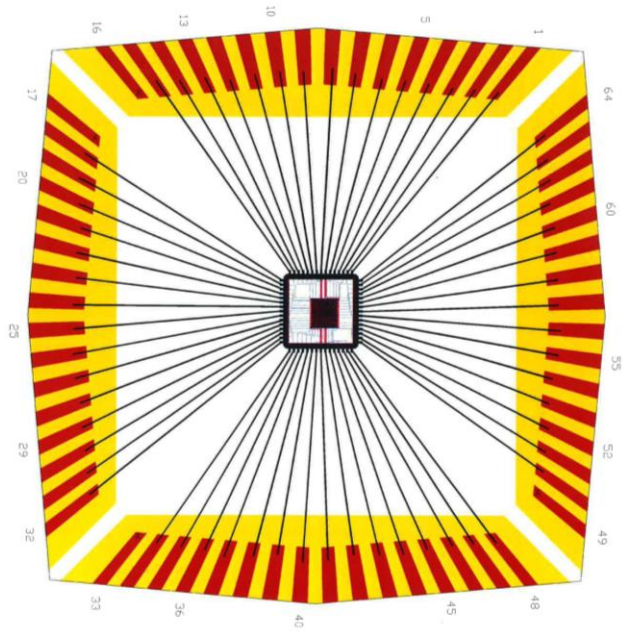


Figura 4.12 *Bonding* ASIC CITIES.

Una de las razones que motivó la elección de este encapsulado fue conseguir un compromiso óptimo entre tamaño de la cavidad y tamaño del circuito integrado para que las conexiones de los hilos (*bonding)* no interfieran negativamente sobre la temporización del circuito integrado. Cuanto más cortas sean las conexiones del *bonding* menos interferencia sobre la frecuencia máxima del *core* tendrá el encapsulado.

A continuación se muestra la hoja de características del encapsulado [81], (Figura 4.13) que proporciona el fabricante, necesaria para elegir el zócalo de fuerza cero que va a albergar al circuito integrado en las pruebas de laboratorio y que va a ser soldado en la placa de test.

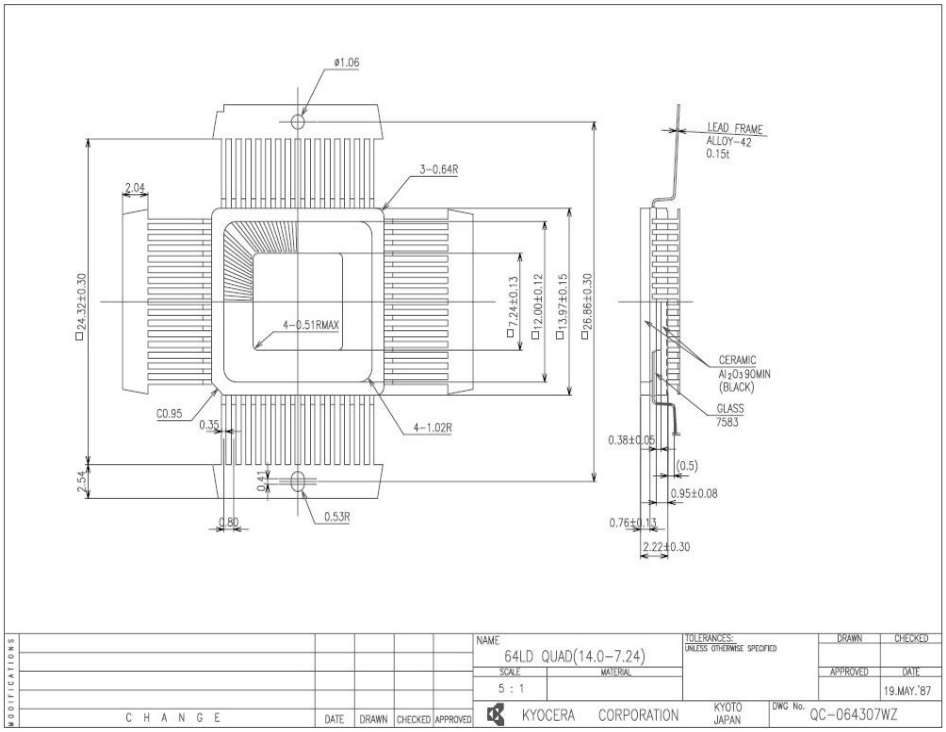


Figura 4.13 Encapsulado CQFP 64 de la página del fabricante.

4.5. Mediciones post-*layout* del consumo promedio de potencia

El consumo promedio de potencia tras finalizar el *layout* se ha obtenido utilizando una metodología muy parecida a la del cálculo del consumo post-síntesis, con la diferencia de que en este caso se ha usado la herramienta de cálculo de potencia de la herramienta de implementación física basada también en estímulos, con la inclusión también de la actividad de conmutación de los nodos del circuito. Para la generación de este fichero de actividad se ha utilizado igualmente el simulador y la *netlist* generada a partir del *layout*.

La medida de consumo de potencia es dependiente de los estímulos y es por ello importante hacer una adecuada elección de los patrones de vectores para obtener unos resultados significativos. En este caso se llevó a cabo una simulación utilizando unos patrones que activan todos los relojes internos de los bloques Trivium y se utilizó un fichero de retrasos post-*layout* tipo sdf obtenido de la herramienta de generación de *layout* (*Encounter*). Este fichero de retro-anotaciones contiene la información sobre el retardo de las interconexiones y de las celdas de la implementación física. De la simulación post-*layout* se obtuvo el fichero vcd que se

usó para el cálculo del consumo de potencia.

Se han calculado datos de potencia de las diferentes versiones Trivium y también la de los diferentes árboles de reloj que actúan sobre cada uno de ellos en el circuito para ver su efecto ya que el árbol de reloj añade un consumo adicional al total debido a los *buffers* que añade.

El cálculo de consumo de potencia se llevó a cabo para una polarización del núcleo (*core*) de 1,08 V y temperatura de 125ºC y datos temporales de peor caso.

4.5.1. Datos del consumo de los árboles de reloj

El árbol de reloj añade un consumo extra al circuito que no viene recogido en los análisis de potencia post-síntesis que se realizaron en el apartado 4.3.3. Se pretende ahora realizar este estudio para comprender los resultados que se han obtenido de los prototipo de bajo consumo Trivium.

Para calcular el consumo de los árboles de reloj generados en el interior del circuito se ha tenido solo en cuenta los referidos al módulo *uut_bt*. Para ello se ha seleccionado el reloj principal de este bloque y los relojes generados a partir de él para cada prototipo.

En el caso de los Trivium MPLP el árbol de reloj generado correspondiente al reloj principal (*clk_bt*) y al dividido (*clkm*) se tiene fuera de los cifradores. Para los casos Trivium FPLP el árbol de reloj dividido se tiene dentro del cifrador con lo que el consumo de cada cifrador se penaliza y aumenta un poco más con respecto a si se hubiera sintetizado fuera del mismo como en el caso MPLP y estándar. Esto es así porque los módulos de los cifradores Trivium MPLP tenían problemas temporales y funcionales si no se generaba un árbol de esta forma. Los cifradores MPLP van a tener dos entradas de reloj en sus módulos, el del sistema (*clk_bt*) y el reloj dividido (*clkm)*, mientras que en los Trivium FPLP solo se tienen una entrada de reloj (*clk_bt*) e internamente se divide el reloj, con lo que el árbol de reloj queda dentro del cifrador. Esta modificación de los pines de reloj en los módulos Trivium se llevó a cabo tras realizar una primera verificación temporal post-*layout* del diseño y supuso la modificación del código *VHDL* de las versiones MPLP y del bloque de mayor jerarquía que englobaba a todos los cifradores.

Teniendo en cuenta esta problemática se ha optado por hacer un análisis del consumo aportado por cada uno de los árboles de reloj generados para los cifradores Trivium.

Los datos del consumo de los diferentes árboles de reloj cuando se tienen todas las versiones Trivium funcionando se muestran en la Tabla 4.11. También se incluyen en ella el número de celdas que conforman cada uno de los árboles de reloj.

Tabla 4.11 Consumo de potencia post-*layout* árboles de reloj

VDD=1,08 V-T=125°C-50 MHz-caso peor temporización					
Trivium		**Consumo de Potencia (μW)**			
Árbol de reloj	**Celdas**	***Internal***	***Switching***	***Leakage***	**Total**
clk_bt	1215	649,8	512,3	7,06	1169
clk2_bt	43	207,6	114,4	2,195	324.2
clkm	125	164	105,4	3,766	273,1
clk_fp	74	72,67	51,75	1,632	126,1
clk_x2fp	92	74	54,35	1,668	130
clk_x8fp	244	155,5	132	3,462	291
clk_x16fp	350	172,5	178,3,	3,402	354,2

El consumo de los árboles de reloj de los módulos FPLP (*clk_fp, clk_x2fp, clk_x8fp, clk_x16fp*) se hace mayor conforme se aumenta el número de bits a la salida de los cifradores Trivium, debido al mayor número de buffer que necesita incluir para generar el árbol correspondiente. Esto es así debido a que, como ya se comentó en el Capítulo 3, es necesario controlar gran cantidad de multiplexores además de los *flip-flops* de los registros.

Los datos de consumo del árbol de reloj *clkm* se corresponde con el consumo de los relojes de frecuencia mitad de los cifradores Trivium MPLP, incluyendo los de uno, dos, ocho y dieciséis bits.

4.5.2. Datos del consumo en los cifradores Trivium propuestos

Para el adecuado análisis del consumo de potencia de los cifradores se debe tener en cuenta la contribución de la parte específica del consumo que aportan los árboles de reloj. Los arboles de reloj, aunque son parte de la implementación del ASIC -lo que justifica que sean tenidos en cuenta en el análisis-, en el sentido más apropiado no forman parte del objeto de estudio –que son los prototipos Trivium-, por lo que incluirlos enturbiaría el análisis.

Por otra parte, como ya se ha comentado, las implementaciones físicas de los Trivium estándar, MPLP y FPLP presentan una desigualdad importante respecto a los árboles de reloj. En las propuestas estándar y MPLP los arboles de reloj están separados de la parte propia del circuito cifrador y no pueden ser estudiados sus aportaciones individuales a cada una de las versiones, mientras que en las propuestas Trivium FPLP están considerados internamente en cada una de las versiones y pueden ser medidos. Este hecho tiene como consecuencia una importante disimetría a la hora de contabilizar el consumo entre las diferentes versiones de Trivium.

En esta Memoria de Tesis se ha optado por incluir dos comparaciones en el análisis

del consumo de potencia: una con los datos del consumo de las implementaciones incluyendo los arboles de reloj en los cifradores FPLP y otra detrayendo el consumo de los árboles de reloj en estos cifradores FPLP. Se ha rechazado la otra opción, la de añadir en los cifradores estándar y MPLP el porcentaje correspondiente al árbol de reloj de cada uno de ellos, por ser tremendamente difícil, por no decir imposible, debido a la nomenclatura de los nodos del circuito que hace inabordable discernir sobre que módulo Trivium actúa cada nodo.

La Tabla 4.12 representa los datos de potencia post-*layout* de las versiones Trivium incluyendo la del árbol de reloj para el FPLP, así como el porcentaje de la mejora del consumo respecto a la versión estándar.

Para hacer más fácil el análisis de los datos, estos se han representado también gráficamente (Figura 4.14).

Tabla 4.12 Consumo de potencia post-*layout*.

VDD=1,08 V-T=125°C-50 MHz-caso peor temporización					
Propuestas Trivium	**Consumo de potencia (μW)**				**% de Mejora**
	Interno	**Conmutación**	**Fugas**	**Total**	
Trivium_x1	289,3	38,41	2,89	330,6	
Trivium_x1mp	192,5	31,53	2,98	227,0	31,3
Trivium_x1fp	220,5	73,75	4,62	298,9	9,6
Trivium_x2	294,2	42,37	2,92	339,5	
Trivium_x2mp	194,1	34,58	3,0	231,7	31,8
Trivium_x2fp	228,2	83,6	4,69	316,5	6,8
Trivium_x8	314,2	78,28	3,13	395,6	
Trivium_x8mp	248,2	78,77	3,17	330,1	16,6
Trivium_x8fp	345,4	195,2	6,44	547,0	-38,3
Trivium_x16	323,8	91,34	3,32	418,5	
Trivium_x16mp	304,4	115,9	3,35	423,7	-1,2
Trivium_x16fp	433,3	297,1	6,78	737,2	-76,2

*$(Mejora = \%(P_{estandar} - P_{paralelizado})/P_{estandar})$.

Como cabía esperar, el consumo crece con el aumento del número de bits a la salida. Por otra parte, las versiones propuestas de bajo consumo efectivamente lo reducen hasta la versión de ocho bits mientras que no lo consiguen en la versión de dieciséis bits. Nuestra propuesta MPLP tiene unas importantes mejoras de casi una tercera parte para las versiones con menos bits a la salida ("x1" y "x2") y una mejora significativa (16.6%) en la versión con ocho bits ("x8"). La versión FPLP, sin embargo, presenta unas aportaciones menos claras puesto que mientras mejora al estándar de un bit a la salida en aproximadamente un 10% y al de dos bits en algo más del 6%, ya en la versión de ocho bits consume significativamente más que el estándar (más de un tercio) disparándose ese mayor consumo en la versión de dieciséis bits.

Se observa un incremento en el consumo de potencia de las versiones Trivium FPLP con el aumento del número de bits a la salida debido sobre todo al árbol de reloj, como ya se había comentado anteriormente. A continuación se presenta un diagrama de barras con estos datos donde se aprecia este incremento y como se hace más apreciable con el aumento del número de bits a la salida.

Es la aportación de los árboles de reloj la que penaliza los resultados de nuestras propuestas de bajo consumo en el caso FPLP.

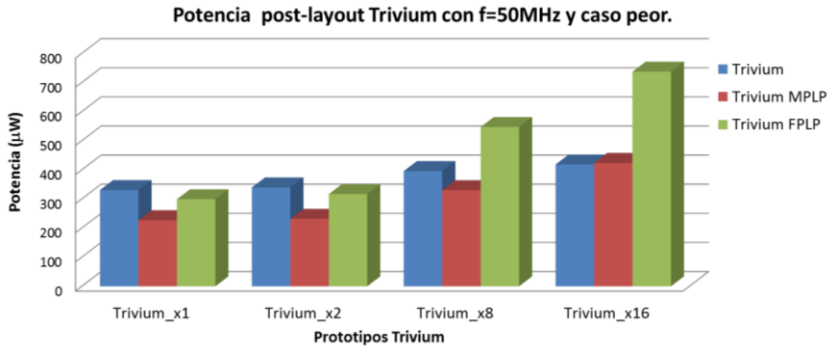


Figura 4.14 Gráfica de comparación del consumo de potencia post-*layout.*

Se calculan los datos del consumo de los árboles de reloj generados dentro de cada versión de los cifradores Trivium FPLP multi-bit, utilizando los informes de potencia que proporciona la herramienta y restándolos del dato total dado en la Tabla 4.12. La Tabla 4.13 presenta, de forma similar a la anterior, los datos de consumo restando a los Trivium FPLP la parte del árbol de reloj.

Tabla 4.13 Consumo de potencia post-*layout* (sin árbol de reloj).

VDD=1,08 V-T=125°C-50MHz-caso peor temporización		
Propuestas Trivium	**Consumo de Potencia**	
	µW	**% deMejora**
Trivium_x1	330,6	
Trivium_x1mp	227,0	31,3
Trivium_x1fp	179,8	45,6
Trivium_x2	339,5	
Trivium_x2mp	231,7	31,8
Trivium_x2fp	197,4	41,9
Trivium_x8	395,6	
Trivium_x8mp	330,1	16,6
Trivium_x8fp	334,8	15,4
Trivium_x16	418,5	
Trivium_x16mp	423,7	-1,2
Trivium_x16fp	512,18	-22,4

*$(Mejora = \%(P_{estandar} - P_{paralelizado})/P_{estandar})$.

Al igual que antes, para facilitar el análisis de estos datos también se han representado gráficamente en la Figura 4.15.

En el caso estándar y en el MPLP los datos de consumo son los ya calculados previamente ya que los arboles de reloj quedan fuera de estos cifradores, por lo que las observaciones nuevas se limitan a las versiones FPLP.

Se observa que ahora la versión FPLP, con la resta del consumo de los árboles de reloj, mejora el consumo de potencia del estándar en todas las versiones Trivium hasta la versión de dieciséis bits donde ya no existe mejora. Es importante destacar que, en el caso de un bit a la salida, la versión FPLP reduce a casi la mitad el consumo del Trivium estándar.

En cuanto a su comparación con la otra versión de bajo consumo, de hecho posee valores mucho mejores que la versión MPLP para las versiones de uno y dos bits, siendo muy similares para la versión de ocho bits.

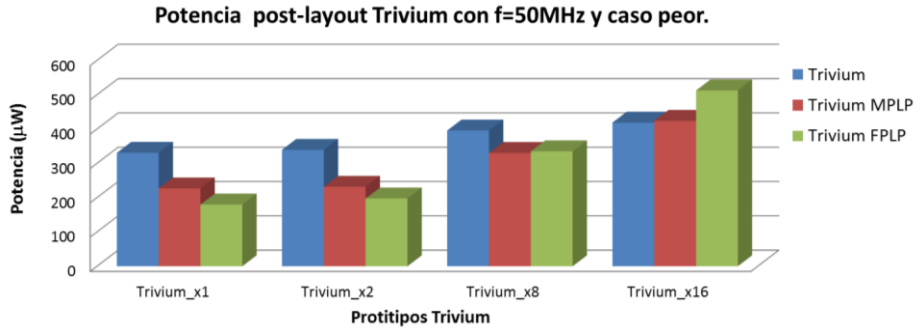


Figura 4.15 Datos del consumo de potencia en los cifradores Trivium.

Por último, en la Figura 4.16 se presenta un diagrama de barras con los porcentajes de mejora en los consumos. Los porcentajes son positivos cuando el consumo es menor que el del estándar y negativo en caso contrario. Obviamente el valor de mejora del estándar es 0. En dicha figura es inmediato apreciar el menor consumo de las versiones MPLP y FPLP excepto en el caso de dieciséis bits, en el que consumen más.

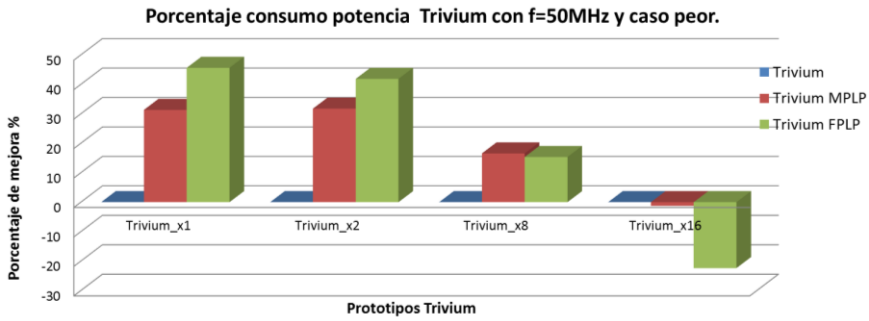


Figura 4.16 Porcentaje de mejora en el consumo post-*layout*.

Los principales resultados del análisis de datos de consumo de potencia post-*layout* tras eliminar los efectos de los árboles de reloj son:

- Las dos versiones implementadas, MPLP y FPLP, en los casos de las propuestas Trivium de uno y dos bits permiten reducir enormemente el consumo de la versión estándar del Trivium. En el caso de la propuesta Trivium de ocho bits, aunque significativa, la reducción es bastante menor. En el caso de dieciséis bits los diseños de bajo consumo no lo reducen.
- La versión FPLP mejora entre 15 y 10 puntos el porcentaje de mejora de consumo a la que logra MPLP cuando el número de bits a la salida es bajo. Para ocho bits a la salida ambas versiones logran reducciones similares.
- Comparando los datos de post-síntesis y post-*layout*, teniendo en cuenta las diferentes tensiones que se tienen en uno y otro caso para la medida (1,2 V en el caso de síntesis y 1,08 V en el caso de *layout*), se deduce que los resultados de mejora se han mantenido en los casos de uno y dos bits, pero a partir del caso de ocho bits se empieza a tener una disminución progresiva de 6-12 puntos en la versión MPLP y 20-38 puntos en FPLP.

4.6. Conclusiones

En este capítulo se ha descrito el diseño de un prototipo de circuito integrado en una tecnología de 90 nm que contiene las propuestas de los cifradores Trivium de bajo consumo junto con las versiones estándar. Estas propuestas son realizadas con salidas de uno, dos, ocho y dieciséis bits, para obtener resultados de recursos, área y consumo de potencia a partir de datos de post-síntesis y post-*layout*.

De los resultados obtenidos se deduce que los recursos consumidos crecen significativamente con el aumento de bits de salida. Las propuestas Trivium de bajo consumo MPLP y FPLP, mantienen una buena relación en el número de celdas e interconexiones frente a la versión estándar en las propuestas de un bit y dos bits

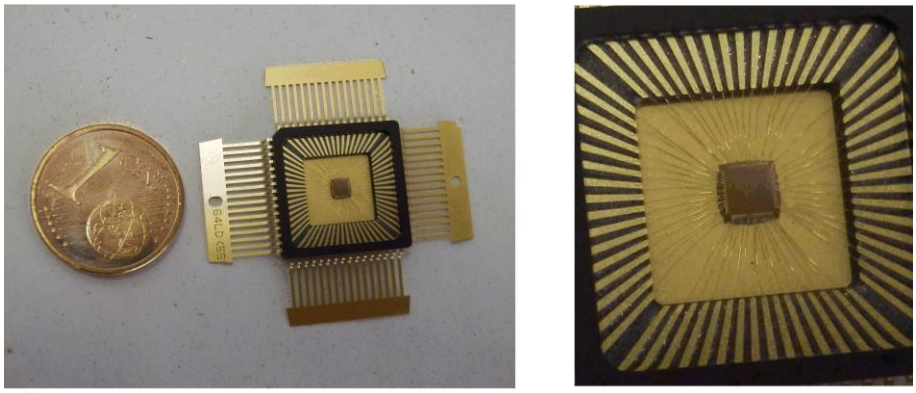
de salida, dando un área muy parecida cuando se comparan entre ellas, pero a partir de ocho y dieciséis bits la diferencia se hace mucho mayor, sobre todo en la versión FPLP de dieciséis bits donde hay una clara diferencia de recursos y área. Los cifradores Trivium FPLP tienen más celdas lógicas y más área combinacionales que las otras propuestas de cifradores Trivium.

Los datos obtenidos en cuanto al consumo de potencia tras la síntesis y la implementación física son bastante acordes entre sí, teniendo en cuenta las diferentes tensiones que se tienen en uno y otro caso para la medida (1,2 V y 1,08 V) y sus porcentajes coinciden prácticamente hasta la versión de ocho y dieciséis bits, donde las diferencias se hacen más apreciables. Los casos más favorables en cuanto a la mejora del consumo de potencia se dan en las versiones de menos bits de salida, siendo los casos de uno y dos bit de los Trivium FPLP y MPLP los que mejores resultados ofrezcan en su conjunto.

Las dos versiones implementadas, MPLP y FPLP, en los casos de uno y dos bits reducen hasta en un 50% el consumo de la versión estándar de Trivium mientras que en el caso ocho bits, la mejoría es mínima y en el caso dieciséis bits el diseño FPLP de bajo consumo ya no lo reduce, quedando igual en el caso MPLP.

5. TEST Y CARACTERIZACIÓN DE LOS PROTOTIPOS TRIVIUM

Una vez diseñado, fabricado y recibido los prototipos del circuito integrado, de los que dos fotografías se muestran en la Figura 5.1, el siguiente paso consiste en verificar si el circuito cumple con las especificaciones para las que fue diseñado, en caracterizar su funcionamiento y en localizar posibles defectos de fabricación en el circuito.



a) b)

Figura 5.1 Fotografía del ASIC CITIES, a) con encapsulado CQFP 64 y b) detalle.

La utilización de equipos automáticos de test también llamados por sus siglas ATE (*Automated Test Equipment*) genera flexibilidad y versatilidad en las diferentes tareas que se tienen que llevar a cabo, aumentando la reproducibilidad de las pruebas de caracterización del circuito, acortando considerablemente el tiempo y coste del test. Además proporcionan una solución muy eficiente y compacta a la hora de la realización de los diferentes test, eliminando buena parte de los problemas derivados del uso de material adicional de laboratorio (cables, sondas, etc.) requerido en una solución basada en equipos convencionales (fuentes de alimentación, osciloscopios, analizadores lógicos y generadores de señal, entre otros).

Las versiones del cifrador Trivium incluidas en el chip CITIES han sido testadas y caracterizadas en el laboratorio utilizando el sistema de test automático de señal mixta *Agilent 93000 SOC C200e*, del que se muestra una fotografía en la Figura 5.2. Este equipo proporciona, sobre una única plataforma, facilidades para la realización de test tanto para prototipado como para fabricación de circuitos de señal mixta ya encapsulados. Cubre una amplia gama de aplicaciones y ofrece las capacidades de prueba necesarias para el test de una amplia gama de circuitos digitales, analógicos y mixtos.

Figura 5.2 Equipo de test automático *Agilent 93000.*

Estos equipos son muy útiles cuando se tienen que testar gran cantidad de muestras con una variedad de medidas y diferentes configuraciones. En nuestro caso había que testar todas las muestras encapsuladas recibidas y realizar bastantes medidas en el circuito integrado. El sistema nos va a ayudar en la repetición y reproducción de las medidas además de permitirnos una mayor automatización. La utilización del equipo también nos ha permitido, realizar cómodamente las verificaciones funcionales a todas las muestras, automatizar las medidas a diferentes frecuencias de reloj y de tensión de polarización, utilizar diferentes configuraciones para la medida de la corriente de consumo y generar informes para todas estas medidas.

Para la realización del test y caracterización del circuito integrado ha sido necesario la creación de un plan de test, en el que se plasmaron las pruebas y medidas a realizar sobre el circuito, y el diseño de una placa de prueba denominada DIB (*device interface board*) que forma la interfaz eléctrica entre el dispositivo bajo prueba (DUT: *device under test*) y el equipo de test (ATE).

A continuación, en el primer apartado se introduce el equipo de test *Agilent 93000* y sus características más importantes. El segundo apartado presenta el plan de test de

los prototipos Trivium donde se contemplan las diferentes pruebas realizadas y el diseño de la placa de prueba. En el tercer apartado se muestran los resultados de la verificación funcional de los prototipos y por último en el cuarto apartado se presentan las pruebas de caracterización de los prototipos llevadas a cabo y los resultados obtenidos.

5.1. Equipo de test *Agilent 93000*

El sistema de test automático de señal mixta *Agilent 93000 SOC C200e*, es llamado actualmente *V93000 SOC,* siendo la empresa que ahora lo comercializa *Advantest Corporation*. Haciendo un poco de cronología, el equipo comenzó siendo fabricado por *Agilent Technologies Inc* pero en el año 2006 fue rebautizado como *Verigy 93000* al ser separada y renombrada la división de equipos de test de semiconductores de la matriz *Agilent* por una compañía llamada *Verigy Ltd*. Posteriormente en 2011 la empresa *Verigy Ltd* fue adquirida por *Advantest Corporation*.

Algunas de las características más relevantes del equipo se refieren a su capacidad para adaptarse a diferentes escenarios de uso (caracterización, producción, control de procesos, pruebas de envejecimiento o *burn-in-test*, entre otras) y a su arquitectura de procesador de test por pin (3ª generación de los equipos de test automáticos de *Agilent*), aportando menor ruido y mayor precisión en las medidas o el uso de refrigeración por agua.

El equipo de test está formado por los componentes que se muestran en la Figura 5.3. Entre los más destacados están:

- Cabeza de test: Es la interfaz para realizar el test al dispositivo bajo prueba. Contiene los módulos digitales y analógicos de test además de la placa donde va montado el dispositivo bajo prueba. A través de unos terminales de entrada/salida también llamados *pogo pins,* se tiene acceso a los canales de entrada, salida y a las alimentaciones.
- Manipulador: Permite posicionar la cabeza de test. Puede moverla hacia arriba, abajo y girarla.
- Armario de soporte de los equipos internos: En él se encuentran instaladas las alimentaciones generales a la cabeza de test y las conexiones para la refrigeración por agua. Además tiene un espacio para la estación de trabajo y otros equipos de test adicionales.
- Unidades de refrigeración: Proporcionan el sistema de enfriamiento del equipo de test, que en este caso se realiza por circulación de agua en las placas internas del equipo. En nuestra instalación se encuentran en una habitación insonorizada contigua.

- Estación de trabajo HP: Contiene el software de control del equipo de test (*SmarTest*). El software funciona bajo el sistema operativo HP-UX aunque actualmente ya existen versiones para el sistema operativo *Windows*.

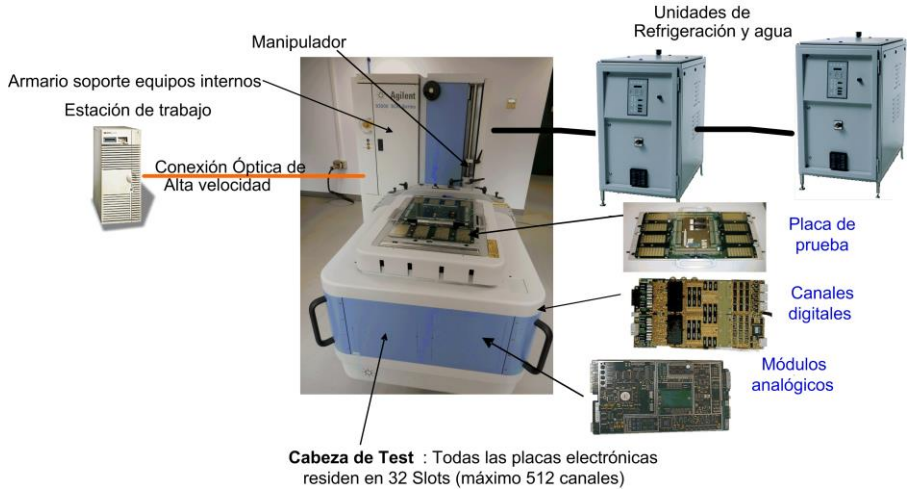


Figura 5.3 Componentes *Agilent 93000 SOC 200e*.

El equipo de test *Agilent 93000 SOC C200e* instalado en nuestro laboratorio puede tener hasta 512 pines entrada/salida de test, que pueden ser digitales, analógicos o conectados a las fuentes de alimentación (DPS). Los canales digitales, pueden ser configurados como entrada, salida o bidireccional. Todos los pines provienen de los módulos digitales y analógicos instalados en su cabeza de test. La cabeza de test dispone de 4 celdas para insertar placas, cada una de las cuales puede contener un máximo de 8 módulos (analógicos o digitales) con 16 pines cada uno. En total dispone de 4 celdas de placas con 8 módulos (32 *slots*) con 16 pines por módulo instalado, lo que da el número máximo de 512 pines.

Si se configura el equipo con módulos analógicos y digitales lo habitual es dividir en dos zonas la cabeza de test, una de ellas para módulos digitales y otra para módulos analógicos.

El equipo de test instalado en nuestro laboratorio está configurado solamente con 128 pines. Tiene 3 módulos analógicos y 5 digitales. Esto supone 80 canales digitales y 48 canales analógicos.

5.1.1. Características digitales

Los canales digitales del equipo de test que tenemos en nuestro laboratorio pueden trabajar a una frecuencia máxima de reloj de 200 MHz. Existe la posibilidad de aumentar hasta 400 MHz por medio de la compra de una licencia software que no

se ha comprado en este caso.

Los canales digitales son completamente configurables individualmente y se utilizan para:

- Aplicar y capturar las señales digitales sobre el dispositivo a testar (vectores de test).
- Sincronizar los módulos digitales y analógicos.
- Realizar test de continuidad y test funcionales.

Para la definición de los vectores de test, cada pin o canal digital del equipo tiene asociado una memoria para vectores de datos con capacidad de 7 MB (*megabyte*) y una memoria por pin para instrucciones y configuración de 2 MB.

La secuencia de los vectores de datos enviadas al *DUT* durante la ejecución del test está determinada por la secuencia de índices que son a su vez punteros a una tabla de formas de ondas. Se pueden tener hasta 32 formas de onda diferentes para cada pin o grupo de pines.

Para definir los vectores de datos se utiliza el término "forma de onda" que representa una secuencia de acciones en un pin digital. Una acción define una puesta de valores lógicos (*drive)* o de toma de valores (*receive)* que ocurre en un punto de tiempo determinado denominado flanco, durante un período del test. Una secuencia de flancos junto con las acciones asociadas determina la forma de onda. Se dispone de hasta 14 flancos, 8 para la puesta de valores y 6 para la toma de valores lógicos.

Un ejemplo se muestra en la Figura 5.4, donde se definen tres formas de onda diferentes, con los índices 0, 1 y 2, que se van aplicando según se desee en los períodos de test. Cada uno de ellos lleva asociado la puesta de unos valores lógicos en un momento determinado del período de test. En el ejemplo mostrado, en el primer y segundo período se usa el índice 0, en el tercero y quinto el índice 1 y en el cuarto el índice 2.

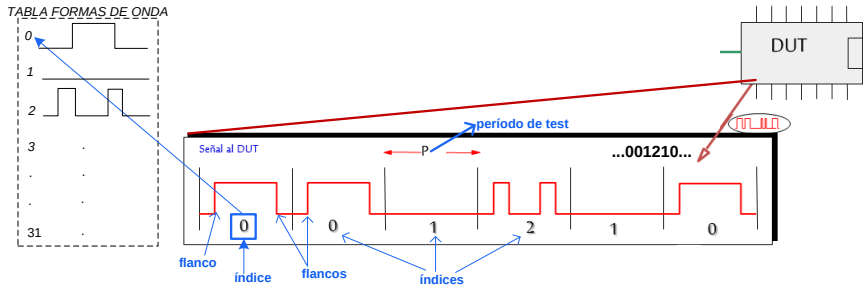


Figura 5.4 Forma de onda en un pin digital.

También es posible generar automáticamente vectores de test con un cierto formato y periodicidad o trasladar patrones test desde herramientas CAD de simulación al entorno del equipo de test mediante programas de ayuda que proporciona el propio equipo de test.

La temporización de las señales también se puede definir para cada pin individualmente o por grupos. Se definen para las señales los tiempos donde se producen las puestas o tomas de valores durante un período de test.

Para la realización del test, como se muestra en la Figura 5.5, el equipo de test coloca una secuencia de vectores de datos en las entradas del circuito y mide los resultados que se producen en las salidas. Estos test se denominan de adquisición de datos y en ellos los datos de las salidas del circuito son muestreados en cada ciclo de reloj y sus valores almacenados en memorias para una posterior visualización. El equipo de test puede almacenar en sus memorias los valores esperados en las salidas, con lo que es posible realizar un test de paso/fallo. En cada ciclo de reloj se produce la comparación entre los valores en las salidas y los valores esperados. El resultado del test es únicamente una "P" de pasado (*passed*) si ha sido correcto o una "F" de fallado (*failed*) si no ha sido correcto. Este test es la base para la realización de las pruebas de verificación funcional y de caracterización.

Los resultados comparados se almacenan en una memoria global de errores y en una memoria detallada por pin. El resultado final es un test de paso/fallo general o más detallado para visualizar las salidas del diseño. En este último caso se usan las herramientas de caracterización y depuración que proporciona el equipo de test.

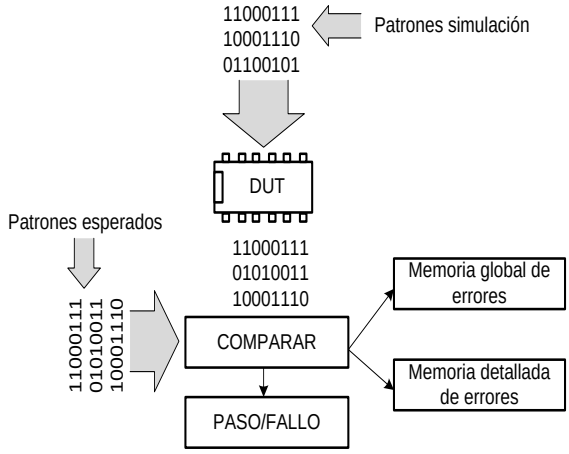


Figura 5.5 Comparación de los vectores en un test funcional.

5.1.2. Características analógicas

Los módulos analógicos se utilizan para la realización de test de señal mixta, como la captura de señales analógicas, la generación de formas de onda o el análisis de intervalos de tiempo para medir frecuencias, retrasos y *jitter*.

Hay un catálogo de módulos analógicos con diferentes funcionalidades y prestaciones, tales como:

- Digitalizadores de formas de onda (*waveform digitizers*, WDG).
- Muestradores (*samplers*).
- Generadores de forma arbitraria (*arbitrary waveform generators*, AWG).
- Analizadores de intervalos de tiempo (*time interval analyzer*, TIA).

En el caso del equipo de test que tenemos en nuestro laboratorio los módulos analógicos instalados y disponibles son tres:

- Generador de señales analógicas de alta velocidad (AWG-WGD): señal sinusoidal de frecuencia máxima 125 MHz, 500 Msps (*sps: samples per second*), 12 bits de resolución, 2.5 Vpp máximo a 50 ohm y 8 salidas individuales o 4 diferenciales.
- Generador de señales analógica de alta resolución (AWG-WGE): señal sinusoidal de frecuencia máxima 7.5 MHz, 30 Msps, 16 bits de resolución, 2.5 Vpp a 50 ohm y 8 salidas individuales o 4 diferenciales.
- Digitalizador de formas de onda de alta velocidad (WDA): Muestreo máximo a 41 Msps, ancho de banda de 100 MHz, 12 bits de resolución, rango mínimo 250 mV y 8 entradas individuales o 4 diferenciales.

Además contiene 8 placas con fuentes de polarización (*DPS11-14*, *DPS21-DPS24*) situadas en la cabeza de test, con un rango de salida ±8 V y hasta 8 A por canal.

Cada módulo analógico tiene al menos un pin de entrada de disparo o *trigger*, una entrada de reloj, un módulo secuenciador con memoria y una memoria de formas de onda. La memoria del secuenciador contiene la secuencia de formas de onda que serán generadas o capturadas.

El bloque generador de la temporización es el encargado de generar el reloj de muestreo. Este reloj se genera a partir de la entrada de reloj al módulo analógico y se activa con una señal de disparo. La memoria de formas de onda, mantiene los datos de las formas de onda generadas o capturadas (hasta 256). Los módulos analógicos tienen un banco de multiplexores para encaminar las señales a/desde varios pines de salida o entrada de los módulos.

En la Tabla 5.1 se detallan las especificaciones y características más relevantes del módulo digitalizador de señal analógica WDA de 41 Msps, que es la única placa analógica que va a ser utilizada en el test del ASIC CITIES.

Tabla 5.1 Especificaciones digitalizador WDA.

Frecuencia muestreo	1 Msps a 41 Msps.
Resolución	12 bit.
Ancho de banda (BW)	100 MHz a ± 0,5 V de rango.
Pines por módulo	8 individuales o 4 diferenciales.
Memoria formas de onda	512K muestras.
Rango de entrada	± 2 V, ± 1 V, ± 0,5 V, ± 0,25 V.
Rango de offset DC	± 4 V.
Rango de modo común	± 2 V a 50 ohm, 37,5 ohm, 10 Kohm unipolar y 100 ohm, 10 Kohm diferencial (sin *offset*). ± 2 Va 50 ohm, 37.5 ohm unipolar y 100 ohm diferencial (*offset*). ± 6 V a 10 Kohm.
Impedancia de entrada	10 Kohm, 50 ohm, 37.5 ohm unipolar. 10 Kohm, 100 ohm diferencial.
Filtros	6.1 MHz, 13 MHz, 26 MHz.

5.1.3. Utilización del equipo de test *Agilent 93000*

El control del equipo de test se realiza desde un programa que se ejecuta en una estación de trabajo, desde el cual se configura el equipo, se manipulan los datos, se realizan y verifican los test, etc. Todas estas tareas son realizadas desde un entorno software de configuración modular llamado *SmarTest*, propio del equipo *Agilent* y con la ayuda de programas escritos en C, C++ o con funciones ya predefinidas del equipo. De esta forma se pueden ejecutar test individuales o test más complejos compuestos de varias secuencias de test y ficheros asociados de configuración.

Antes de ejecutar cualquier test se tiene que llevar a cabo una serie de configuraciones básicas utilizando la consola principal del software *SmarTest*. Se trata de las configuraciones de pines, de los niveles de tensión de las entradas o salidas y de las polarizaciones, de la temporización y de los patrones de prueba:

- Configuración de pines: Se asocia a cada pin físico del equipo un nombre lógico. Puede ser un pin de entrada, salida, bidireccional, analógico o de alimentación (DPS). Pueden hacerse grupos de pines y así aplicarles valores en el resto de las configuraciones.
- Niveles de tensión de polarización y niveles lógicos: Se definen los valores de la tensión para las alimentaciones, los niveles lógicos de '0' y '1' y el nivel lógico de comparación para las señales de salida. Estos valores pueden aplicarse individualmente a cada pin o a todos los pines de un grupo.
- Temporización: Se definen las posibles formas que pueden tener las señales de entrada en cada uno de los períodos de test. Además se define,

para cada ciclo, el instante de muestreo de las señales de salida.

- Patrones de vectores de test: Para cada uno de los pines de entrada se define la forma de señal en cada ciclo. Esta forma lleva implícita el valor lógico. También puede definirse el valor esperado en cada ciclo de las salidas del circuito.

Si el test a realizar es de señal mixta se tiene que configurar también cada uno de los módulos analógicos que se vayan a utilizar.

Para la ejecución de los test se utiliza desde el gestor de ficheros el editor de flujo de test, que proporciona el software del equipo, el cual permite mediante una interfaz gráfica definir la secuencia de test individuales o tareas de test que se van a ejecutar. Estos test pueden ser conectados de varias formas: en modo secuencial, con alguna dependencia, mientras una condición sea válida, etc.

Un ejemplo puede verse en la Figura 5.6, donde aparecen las principales ventanas del software *SmarTest:* la consola principal, el gestor de los ficheros de test (*data manager*), el editor de flujo de test (*testflow*) y la consola de usuario. El editor de flujo de test permite gráficamente ejecutar varios test que son conectados secuencialmente o en bucle, dando mensajes de si el test es correcto (triángulo verde) o reportando errores si alguno falla (octógono de color rojo).

Entre los tipos de test que se pueden ejecutar destacan:

- Funciones de test predefinidas en el equipo y que cubren la mayoría de los test encaminados a evaluar el comportamiento en DC o AC del dispositivo bajo prueba.
- Métodos de test: Funciones configuradas por el usuario a partir de un programa en C o C++ que usan funciones predefinidas y librerías propias del usuario.
- Procedimientos de usuario: Un test definido en base a un programa de test escrito en C.

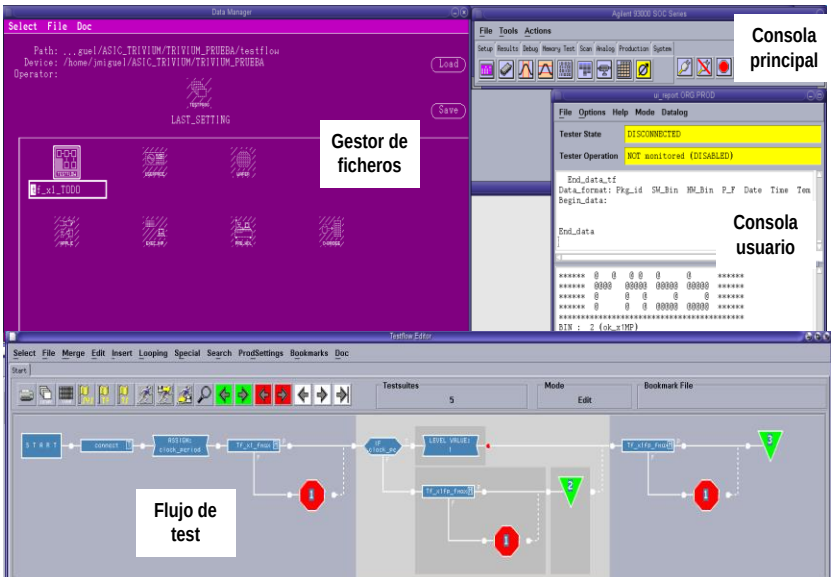


Figura 5.6 Ventanas principales del software *SmarTest*.

5.1.3.1. Funciones de test predefinidas

El software tiene ya predefinidas algunas funciones de test que evalúan y caracterizan el comportamiento en DC o AC del dispositivo bajo prueba.

Para evaluar el comportamiento en DC del dispositivo, el software dispone de funciones de test de continuidad, de medida de la tensión en cualquier pin, de medida de la corriente en un pin de entrada, salida o en alta impedancia forzando una tensión, y de medida de las corrientes de las fuentes de alimentación o de *Iddq*, entre otras.

Para el comportamiento en AC y caracterización [82] del circuito bajo prueba, se dispone de test funcionales (que comparan las salidas del circuito con los valores esperados), test que miden la frecuencia máxima de operación, que miden tiempos de *setup* y de *hold* de la señales de reloj, de medida de los niveles umbrales de las señales de entrada y salida, de los tiempos de propagación de las salidas y los límites de operación al variar uno de los parámetros de configuración.

También existen definidos test que muestran de forma gráfica los límites de funcionamiento de un circuito cuando se varían dos parámetros. Son los llamados *Shmoo plots*.

En el caso de test de señal mixta el equipo dispone entre otras, de ciertas funciones relativas a test para distorsión, linealidad y ganancia, para convertidores analógicos-digitales y digitales-analógicos.

También hay definidos otros test más específicos que se refieren a funciones de diagnóstico, y de almacenamiento de datos.

5.1.3.2. Importar patrones y temporización de una herramienta CAD

La introducción de patrones de test en el equipo de test puede ser una tarea muy tediosa, más aun cuando hay que introducir no solo los patrones de entrada sino también las salidas esperadas. Por ello es posible convertir los vectores y datos de la temporización de herramientas de CAD comerciales (*Cadence*, *Mentor*, *Synopsys* y *otras*) a ficheros de configuración de temporización y de vectores para el equipo *Agilent 93000*.

Se utiliza para ello una herramienta software del equipo de test denominada *interfaz ASCII*, la cual proporciona unos programas de ayuda para poder trasladar los patrones de test en formato ASCII y las definiciones de temporización al formato requerido por el equipo de test. Para la interfaz de temporización se utiliza el programa, ait (*ASCII interface timing*) y para la definición de los patrones de test el programa, aiv *(ASCII interface vectors)*.

Las tareas a llevar a cabo para convertir los vectores y datos de la temporización de la herramienta de CAD utilizada a ficheros de configuración de temporización y de vectores para el equipo *Agilent 93000* son:

a) Almacenar en un fichero ASCII las entradas y salidas del circuito bajo prueba y modificar este fichero para añadirle una cabecera que contenga los nombres de las señales de los pines de entrada y salida conforme a la tabla generada.
b) Ejecutar el programa de generación de la temporización (ait) junto con el fichero de patrones en formato tabla, el cual genera el fichero de temporización y algunos ficheros adicionales que son utilizados a posteriori.
c) Ejecutar el programa de traslación de patrones (aiv) junto con los generados anteriormente, generándose el fichero con formato propio del equipo de patrones de test. Es necesario usar el fichero de configuración de pines que se ha creado cuando se configura el equipo de test para estas operaciones.

5.1.4. Herramienta de resultados

El software del equipo *SmarTest* posee varias utilidades destinadas a la visualización de resultados, necesarias para la caracterización y depuración de los circuitos bajo prueba. En el caso de visualizar los errores en un test se necesita la posibilidad de identificar el pin que falla y el ciclo de test donde ha ocurrido, para

así determinar si el problema viene de la configuración o del dispositivo. Para ello se tienen las siguientes herramientas de depuración:

- Mapa de errores *(Error map)*: Visualiza los errores de forma gráfica para identificar el pin que falla y el ciclo de test donde ha ocurrido (determina si el problema viene de la configuración o del dispositivo).
- Diagrama de formas de onda *(Timing Diagram* y *Softscope)*: Produce una forma de onda gráfica con los datos recibidos. Tiene una opción de alta resolución, consistente en realizar múltiples test con los mismos patrones pero muestreando en momentos distintos de forma que se aproxime lo más posible al comportamiento real del sistema.
- Lista de estados *(State List)*: Produce un listado de los valores lógicos capturados en formato tabla.
- *Shmoo plot:* Realiza una representación visual del comportamiento del dispositivo mientras se varían dos parámetros específicos.

5.2. Testado de los prototipos Trivium

El equipo de test de señal mixta *Agilent 93000* es utilizado para generar las señales digitales necesarias para estimular el circuito integrado (señales de entrada, control y reloj de las versiones Trivium), proporcionar las alimentaciones y tensiones de referencia y además, adquirir y procesar la salida de las diferentes versiones de las arquitecturas propuestas.

Para la realización del test de los prototipos es necesario, como se comentó en la introducción del capítulo, el desarrollo y ejecución de un plan de test donde se contemplen las diferentes medidas que se quieren realizar sobre el circuito a testar y las pruebas a las que se va a someter. Acorde con este plan de test hay que diseñar una placa de prueba (DIB) de adaptación al equipo de testado *Agilent 93000* donde se coloque el dispositivo bajo prueba.

Después se procede a configurar el equipo de test. Este proceso ha sido comentado en el apartado anterior y consiste en preparar las configuraciones básicas de pines, niveles de tensión de las entradas y salidas, de las polarizaciones, de la temporización y de los patrones de prueba.

Una vez realizadas las configuraciones se definen los test mediante funciones específicas de test del equipo y métodos de test, con el editor de flujo de test. Finalmente se analizan los resultados con las herramientas proporcionadas por el software del equipo.

Las principales fases del test digital que se ha llevado a cabo sobre el circuito CITIES se pueden resumir:

1) Desarrollo del plan de test para los prototipos Trivium contenidos en el chip.
2) Diseño de una placa a medida para el ASIC CITIES.
3) Configuración del equipo de test *Agilent 93000:*
    a. Pines, niveles de tensión y temporización.
    b. Configuración y generación de los patrones de test.
4) Configuración de los diferentes flujos y tareas de test.
5) Análisis de los resultados.

5.2.1. Propuesta del plan de test. Verificación de los prototipos.

El plan de test propuesto para el ASIC CITIES consta de dos grupos de prueba:

- Pruebas de verificación funcional.
- Pruebas de caracterización.

Las pruebas de verificación funcional tienen un doble objetivo: verificar experimentalmente si el circuito tiene la funcionalidad deseada y en segundo lugar, separar de las muestras aquellas que tengan defectos de fabricación. De esta forma solo las muestras sin errores serán caracterizadas.

Todas las versiones del cifrador Trivium incluidas en el circuito integrado CITIES tendrán que ser testadas individualmente para comprobar su funcionalidad. Para la realización del test funcional se escogerá una frecuencia de operación no muy alta y se utilizaran valores de tensión mínimos, máximos y típicos de polarización y de entrada y salida de datos, suministrados por el fabricante para la tecnología utilizada, en nuestro caso TSMC 90 nm. También se utilizarán diferentes parejas de valores de la clave y vector de inicialización, en concreto las mostradas en el Capítulo 4, Tabla 4.6 y que son recomendadas por la organización ECRYPT [61].

Para las pruebas de caracterización se planteó medir la frecuencia máxima de operación y la corriente de consumo promedio en operación de cada una de los cifradores Trivium con los tres valores de tensión de polarización. Para la medida de corriente se utilizará la propia circuitería interna que posee la fuente de polarización. En este caso es necesario usar una señal de disparo para activar la circuitería de medida en la fuente y además se utilizarán varias frecuencias de operación. Por otro lado, también se quiere medir la curva de consumo instantáneo haciendo uso en este caso del módulo digitalizador que tiene el equipo de test. La frecuencia de operación en este caso tendrá que ser baja para poder capturar un número adecuado de muestras en la memoria del módulo analógico ya que su frecuencia de muestreo es de 41 Msps. La medida de la corriente de consumo promedio y de consumo instantáneo hay que hacerla sobre los pines de

alimentación del anillo y en los del núcleo (*core)* cuando el circuito está en modo operación.

Para llevar a cabo este plan de test hay que considerar por tanto la realización de una placa de prototipado a medida que permita la realización de todas las pruebas especificadas incluyendo los elementos necesarios para ello. Además, con el fin de agilizar las medidas y evitar un consumo de tiempo alto para llevarlas a cabo se automatizará en la mayoría de los casos las pruebas a realizar con el equipo de test *Agilent 9300*.

5.2.2. Diseño de la placa de prototipado para el test.

La interfaz entre el equipo de prueba y el circuito a testar se realiza mediante una placa de prototipado dedicada específicamente a ello. Esta placa se fija con tornillos sobre un soporte metálico (*Stiffener*), específico del equipo de prueba que contiene las marcas necesarias para su posicionamiento correcto en la cabeza de test. Para los test a realizar se optó por diseñar una placa específica que se conectara a otra placa ya diseñada y fabricada denominada placa base. Esta placa contiene unos conectores especiales tipo *ERMET* para permitir acoplar sobre ella una segunda placa de tamaño más reducido y menor coste, que se diseña a medida (placa *DIB*) para el test de los circuitos. Esta solución es muy práctica y económica, porque contiene todo lo necesario para el test del dispositivo específico, además de tener un menor tamaño, ser más flexible y ofrecer múltiples posibilidades al usuario final del sistema.

En la Figura 5.7 se muestra cómo la placa base está atornillada sobre el soporte *Stiffener* y sobre ella se ha acoplado una segunda placa más pequeña.

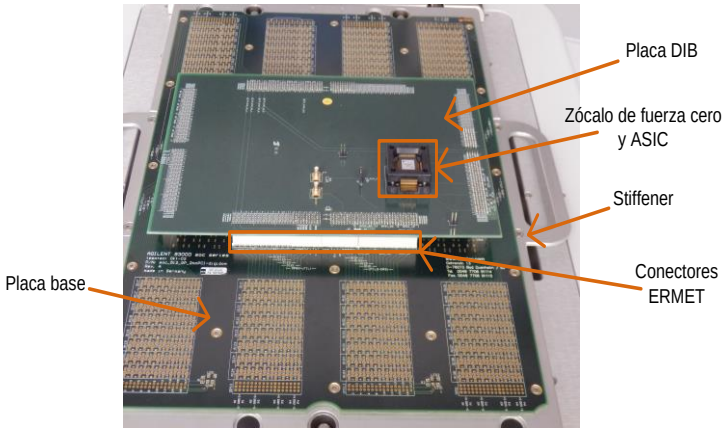


Figura 5.7 Montaje final de la placa de prototipado ASIC CITIES.

En nuestro caso, esta segunda placa a medida se ha diseñado para alojar el ASIC CITIES y contiene todos los elementos necesarios para llevar a cabo el plan de test propuesto. El chip va alojado en un zócalo de fuerza cero para poder intercambiar cómodamente todas las muestras.

Cada canal de DPS que proporciona las alimentaciones al chip tiene dos convertidores digital analógico (DAC). Uno de ellos para configurar las tensiones y el otro la corriente límite positiva y negativa. También dispone de una entrada de control (*pin VB+*) que permite cambiar entre estos dos DAC. Cada canal de las fuentes de alimentación del equipo *Agilent 93000* puede medir la tensión y corriente que proporcionan al dispositivo bajo prueba (DUT). Este módulo tiene una entrada de disparo (*pin TR+*) que permite sincronizar la medida con los patrones de test durante un test funcional.

Para sincronizar las medidas de corriente promedio con los patrones de test se conectó un canal de uno de los módulos digitales como señal de disparo (*trigger*) al correspondiente pin *TR+* de la placa de prototipado. El pin *TRVB-* (pin de referencia de las señales *TR+* y *VB+*) de la placa de prototipado para la polarización digital se conectó a la masa digital de la misma placa. Estas conexiones se muestran en la Figura 5.8.

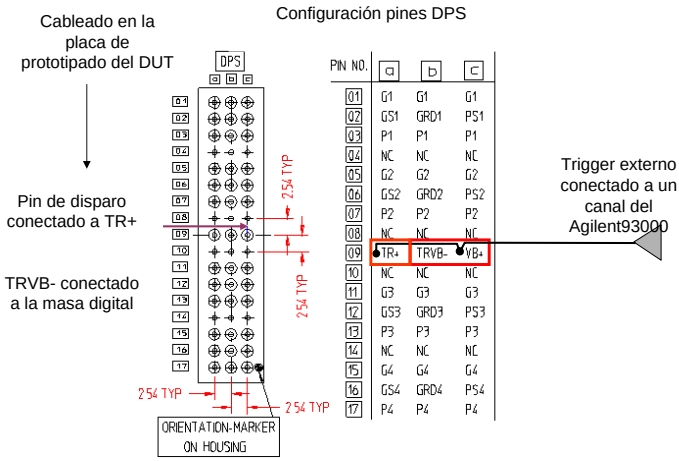


Figura 5.8 Configuración de las DPS en la placa de prototipado.

Para permitir la medida del consumo instantáneo de los cifradores Trivium con el módulo analógico se añadió la opción de colocar una resistencia en serie en cada camino de las alimentaciones del circuito con el objetivo de que midiendo la caída de tensión en esas resistencias hallar la corriente. En caso de no necesitarlo se puede poner un puente o resistencia de valor cero.

Un esquema del conexionado realizado en la placa de prueba para las dos fuentes de polarización (anillo y *core*), se muestra en la Figura 5.9, donde se incluyen las resistencias serie de cada fuente (con valor de 100 ohmios) y algunos condensadores de desacoplo.

La placa DIB tiene también otros elementos opcionales como por ejemplo unos conectores SMB para la entrada de un reloj externo y para una señal de disparo, así como un puente o *jumper* de configuración para la selección del reloj externo o el generado por el equipo. Se tienen también varios puntos de prueba para visualizar señales con un osciloscopio externo en caso de anomalías durante el test.

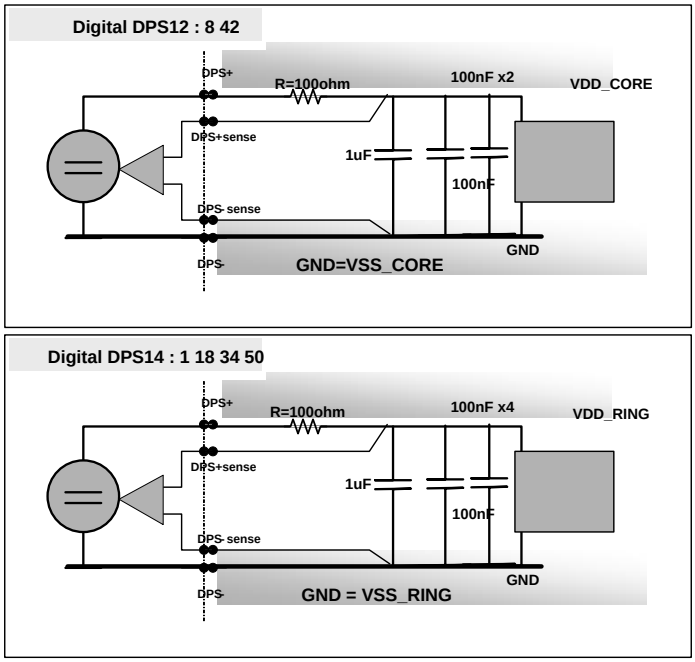


Figura 5.9 Configuración del conexionado a las placas de polarización DPS.

El esquemático de la placa DIB puede verse en la Figura 5.10, donde se realiza el conexionado de los canales del equipo de test, a través de los conectores *ERMET* con los pines del ASIC y los diferentes componentes auxiliares de la placa.

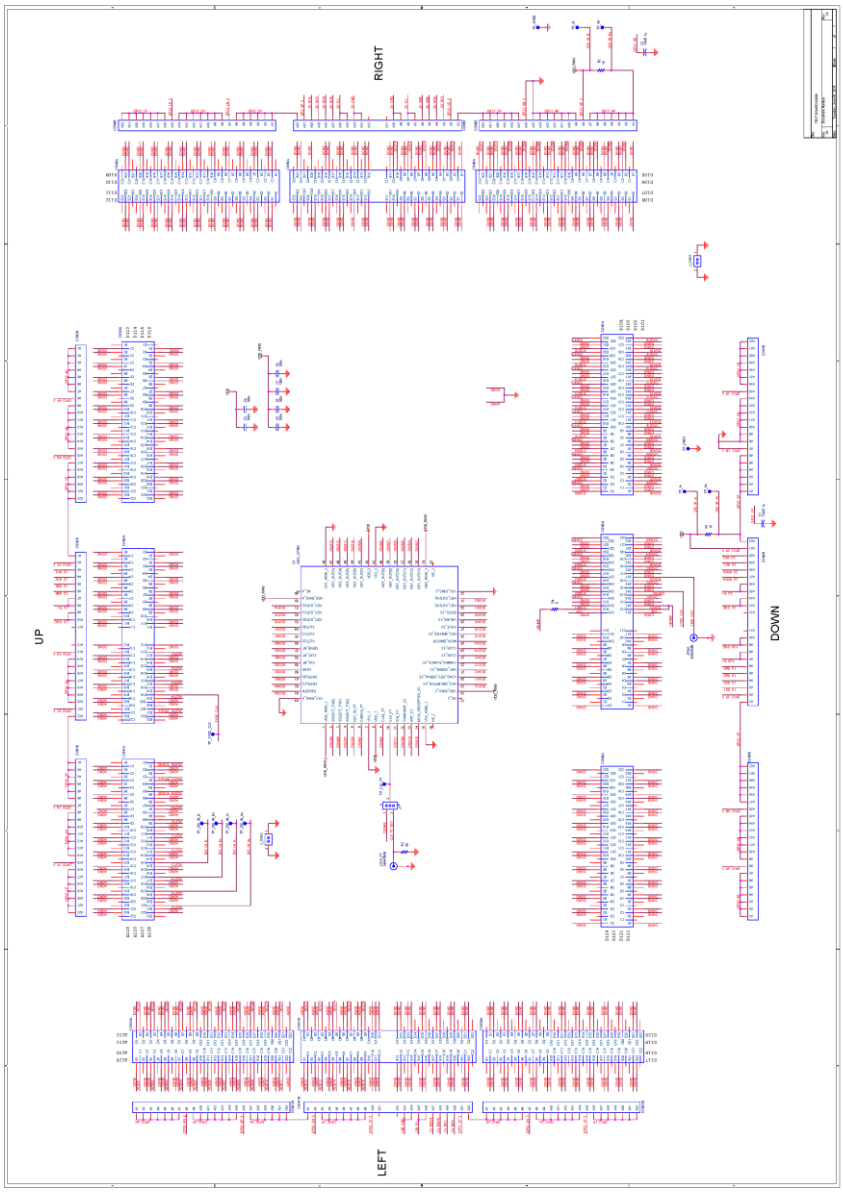


Figura 5.10 Esquemático de la placa (DIB) para el testado del chip CITIES.

5.2.3. Configuración del equipo de test

Una vez desarrollado el plan de test y diseñada la placa a medida para nuestro circuito integrado se procedió a la etapa de configuración del equipo de test *Agilent 93000*.

5.2.3.1. Configuración de los pines

Los parámetros básicos configurados para los pines en nuestro circuito integrado son el número del pin en el zócalo, nombre, modo y tipo del pin, valores de componentes externos a pines como por ejemplo la resistencia serie de algún pin o valores de los condensadores de desacoplo para las DPS y canal del equipo al cual se conecta el pin. La asociación de pines del ASIC y los canales del equipo de test son fijados por la placa DIB diseñada.

La configuración de los pines se hace con los datos que se muestran en la Tabla 5.2, que asocia a cada pin del ASIC CITIES su tipo (si es de entrada o salida), su posición en el encapsulado y el canal del equipo de test al que está conectado.

Por ejemplo, en el caso de la señal de reloj *CLK_BT*, entrada que se conecta en el pin 59 del ASIC y que está conectada al canal 10305 del equipo de test. El canal 10305 es el canal 5 del módulo 3 del equipo.

Por otra parte, los pines que aparecen en la parte superior de la tabla están dedicados a la configuración de los módulos analógicos y a la generación de señales de sincronismo, por lo que no tienen correspondencia con ningún pin del encapsulado del ASIC. En el caso de señales analógicas como las conectadas a los bornes de la resistencia serie (*WDA+*, *WDA-*), se conectan a los canales analógicos 16 y 15 del módulo analógico *WDA227*, que se corresponde con el digitalizador.

Para los test funcionales solamente son necesarios los pines de entrada y salida del ASIC y la definición de las fuentes de alimentación (*DPS+*). Para realizar la medida de la corriente con los módulos de las fuentes de polarización se necesitarán además el canal digital de disparo (*TRIG_BUMP*) y para usar el modulo digitalizador *WDA* se añadirán los pines de las entradas del amplificador (*WDA, WDB*), de la señal de disparo (*WDA_TRIGGER*) junto con el canal digital que actúa sobre ella (*SYNC*).

Tabla 5.2 Configuración de los canales del equipo de test *Agilent 93000*.

Nombre del Pin	Tipo	Pin del ASIC	Canal del equipo de test
WDA+	Entrada		WDA227-16
WDA-	Entrada		WDA227-15
WDA_TRIGGER	Entrada		WDA227-04
WDB+	Entrada		WDA227-14
WDB-	Entrada		WDA227-13
TRIG_BUMP	Entrada		10101
SYNC	Entrada		10102
RESETN	Entrada	63	10304
CNTRL<1:0>	Entrada	62-61	10404-10403
LOAD	Entrada	60	10303
CLK_BT	Entrada	59	10305
CLK2_BT	Entrada	58	10402
SERIE_BT	Entrada	57	10302
SLCT<2:0>	Entrada	56-54	10405-10401-10301
KEY_OUT<0:2>	Salida	53-51	10516-10515-10514
KEY_OUT<3:7>	Salida	47-43	10513-10512-10511-10510-10509
KEY_OUT<8:13>	Salida	40-35	10508-10507-10506-10505-10504-10503
KEY_OUT<14:15>	Salida	31-30	10502-10501
VDD_CORE	DPS+	8-42	DPS12
VDD_RING	DPS+	1-18-34-50	DPS14

5.2.3.2. Configuración de los niveles de tensión

Los niveles de tensión de los canales del equipo fueron configurados con los valores ofrecidos por el fabricante y que se muestran en la Tabla 5.3. Se dan valores mínimos (min.), nominales (nom.) y máximos (max.) de los parámetros utilizados.

Tabla 5.3 Valores procedentes de la tecnología TSMC 90 nm.

Nombre	Parámetro	Min.	Nom.	Max.	Unidades
VDD_CORE	Alimentación celdas (*core*)	1,08	1,2	1,32	V
VDD_RING	Alimentación *pads* I/O	2,25	2,5	2,75	V
Vin	Tensión de entrada			2,75	V
VIL	Tensión de entrada a nivel bajo	-0,3		0,7	V
VIH	Tensión de entrada a nivel alto	1,7		2,75	V
VOL	Tensión de salida a nivel bajo			0,7	V
VOH	Tensión de salida a nivel alto	1,7			V

Para configurar los niveles de tensión de los pines de entrada, salida y polarizaciones se utilizó un fichero de configuración en formato texto junto con la definición de los principales parámetros definidos en la tabla anterior como

variables, para que puedan ser cambiados cómodamente y permitan también la automatización de los test. Los valores de estas variables se introducen en la ventana de especificaciones junto con sus valores mínimos y máximos.

Un ejemplo de la configuración para la definición de los principales parámetros en el formato del equipo de test se muestra a continuación. Se comienza con la declaración de las principales variables de la Tabla 5.3 dentro del apartado denominado especificaciones (*SPECS).* Se ha definido la alimentación del circuito con los mismos nombres que en la tabla, la tensión de entrada a nivel bajo (*VIL*) como *entradas_low*, la tensión de entrada a nivel alto (*VIH*) como *entradas_high*, la tensión de salida a nivel bajo (*VOL*) como *salidas_low* y la tensión de salida a nivel alto (*VOH*) como *salidas_high.*

```
SPECS    #starts the declaration of the specs

VDD_CORE        [V]
VDD_RING        [V]
entradas_low [V]
entradas_high [V]
salidas_low [V]
salidas_high [V]
```

Estas variables se asocian a los parámetros que se utilizan en la definición de los niveles de tensión dentro del software. Para definir el valor de las polarizaciones se utiliza el apartado *EQUATIONS,* anteponiendo la palabra reservada *DPSPINS* a los nombres de los pines y la palabra reservada *vout* a su valor. Para definir el valor de las tensiones de entrada y salida se utiliza el apartado *LEVELSET* con los nombres de los pines igual que antes, con la palabra reservada *DPSPINS* y el valor de ellos con la palabra reservada *vol, voh* en caso de salida y *vih, vil* en caso de entrada.

Una parte del texto de un fichero de configuración se muestra a continuación.

```
EQUATIONS

DPSPINS VDD_CORE
vout   = VDD_CORE
…….
DPSPINS VDD_RING
vout   = VDD_RING
…….
LEVELSET 1 "Test Funcional sin terminación "
PINS salidas
vol= salidas_low
voh= salidas_high
PINS entradas
vil= entradas_low
vih= entradas_high
```

5.2.3.3. Configuración de la temporización

Para la configuración de la temporización de las señales de entrada y salida ha sido necesario la definición de diferentes tablas de formas de onda (una por cada pin o grupo de pines). Se han utilizado también ficheros de texto y variables como en el caso de la configuración de niveles de tensión, que son cómodamente modificables mediante especificaciones, y utilizadas en las funciones de test y caracterización. Los valores son también introducidos en la ventana de especificaciones junto con sus valores mínimos y máximos. En el ejemplo siguiente solo es necesario definir el valor del período de test (*T*) con el valor actual y los valores mínimo y máximo que pueda alcanzar.

Un ejemplo de definición de señales se muestra en la Figura 5.11, donde se define la forma de onda de la señal de reloj (*CLK_BT*) y de las señales de entrada y salida. El reloj se define mediante las acciones de los flancos d_1, d_2 y las entradas mediante d_3. Las salidas son muestreadas con el flanco r1. Posteriormente estos valores son relacionados con el período del ciclo de test (*T*), d_2=0.5*T, d_3=0.9*T, y r_1=0.7*T. La secuencia de flancos junto con las acciones asociadas determina la forma de onda de cada pin. El período de reloj (*Tclk*) se relaciona con el período del ciclo de test (*T*) según cada configuración, como se muestra en la Figura 5.12.

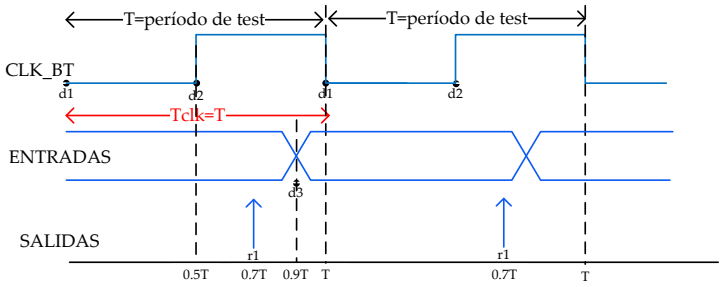


Figura 5.11 Configuración de la temporización y formas de onda.

La definición equivalente para las señales en formato texto se muestra a continuación, donde se definen los patrones 0 y 1 para el reloj y las entradas.

```
PINS CLK_BT
0 "d1:0 d2:0" 0
1 "d1:0 d2:1" 1
PINS ENTRADAS
0 "d3:0" 0
1 "d3:1" 1
PINS SALIDAS
0 "r1:L" 0
1 "r1:H" 1
2 "r1:X"
```

En el caso de las entradas los patrones coinciden con su valor lógico pero en caso del reloj el patrón 1 define un pulso completo. En el caso de las salidas se definen los patrones 0, 1 y 2 que se corresponden con el valor lógico ´0´ o L, ´1´ o H e indeterminado (X).

La relación entre la posición de los flancos y el período se hace en un fichero de texto bajo la declaración de ecuaciones, *EQNSET*.

Un ejemplo de cómo se define en formato texto esta relación se muestra a continuación, donde se ha especificado el parámetro del período del ciclo de test como *T* en nanosegundos bajo la declaración de las especificaciones (*SPECS*). Se definen los flancos de las señales de entradas y el muestreo de las salidas en función de este parámetro bajo la declaración *TIMINGSET*.

```
EQNSET 1 "equation set 1"
SPECS
T [ns]
TIMINGSET 1 "functional"
period=T
PINS  CLK
d1=0
d2=0.5*T
PINS  ENTRADAS
d3=0.9*T
PINS  SALIDAS
r1=0.7*T
```

Para conseguir frecuencias de reloj que lleguen a los 200 MHz hay que cambiar la definición de la temporización del reloj principal, para que en un período de test se produzcan cuatro ciclos de reloj. Como el valor mínimo para el período de test es de 20 ns, para conseguir una señal de reloj que alcance la frecuencia máxima permitida por el equipo (*Tclk*=200 MHz), la definición de los relojes tiene que hacer uso de todos los flancos de señal disponibles (ocho en total, *d0-d7*). En la Figura 5.12 se muestra la definición del reloj usada para bajas frecuencias y la usada para frecuencias altas.

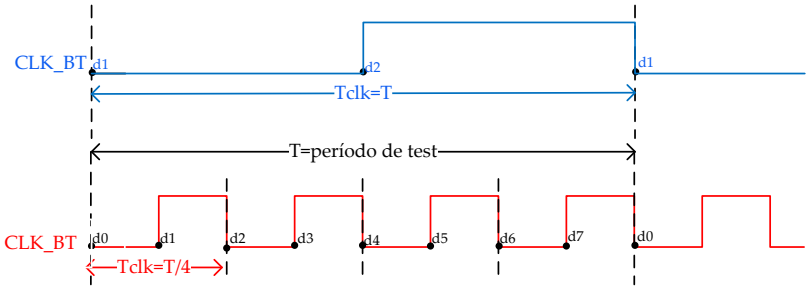


Figura 5.12 Temporización de la señal de reloj, para frecuencias bajas y altas.

5.2.3.4. Configuración de los vectores de test

Para la generación de los vectores o patones de test se tomaron como referencia las simulaciones realizadas en la etapa de diseño y verificación con la herramienta *Modelsim*. Los resultados de simulación (patrones de entrada y de salida en cada ciclo de reloj) se guardaron en un fichero de texto. Este fichero es procesado con la herramienta del equipo *Agilent 93000* denominada *interfaz ASCII*, como se comentó en el apartado 5.1.3.2, generándose los ficheros de patrones de vectores compatibles con el formato requerido para la herramienta *SmarTest*.

Para cada una de las versiones del cifrador Trivium se ha generado un fichero de patrones que lo hacen funcionar con tres parejas diferentes de valores de clave y vector de inicialización (unos 9200 ciclos de reloj). En total doce ficheros de patrones divididos en 4 ficheros de patrones para las versiones Trivium estándar con salida de un solo bit y multi-bit (dos, ocho y dieciséis bits) y 8 ficheros de patrones para las versiones Trivium de bajo consumo en sus versiones FPLP y MPLP, con salidas de un solo bit y multi-bit (dos, ocho y dieciséis bits).

Estos ficheros son usados para verificación y caracterización. Para las medidas del consumo de corriente se han utilizado otros patrones diferentes que serán comentados posteriormente.

5.3. Verificación funcional de los prototipos

Una vez configurado el equipo y generados los patrones de test, se procedió a la verificación funcional de los circuitos. Esta prueba engloba la verificación de todas las versiones del cifrador Trivium integradas en el ASIC.

En total se tienen doce versiones divididas en:

- 4 versiones Trivium estándar correspondientes a salidas de un solo bit y multi-bit (dos, ocho y dieciséis bits).
- 8 versiones Trivium de bajo consumo en sus versiones FPLP y MPLP con salidas de un solo bit y multi-bit (dos, ocho y dieciséis bits).

Todos los test funcionales fueron realizados a una frecuencia de operación de 10 MHz, con tensiones de alimentación del anillo (*ring*) y del núcleo mínima, nominal y máxima.

Se ha automatizado el proceso de realización de estos test funcionales con el equipo de test ya que había que testar diez muestras del chip CITIES. Para la automatización de las pruebas se utilizó la programación del equipo *Agilent 93000* con flujos de test (*testflow*) que ya fue mostrada anteriormente. Con este procedimiento fue posible ejecutar varios test secuencialmente con diferentes patrones de prueba y diferentes configuraciones obteniendo una validación de paso/fallo de las muestras.

Los flujos de test llevados a cabo son mostrados en la Tabla 5.4. Se agruparon en un solo flujo de test los tres test funcionales correspondientes a las versiones estándar y de bajo consumo Trivium (MPLP y FPLP), según el número de bits a su salida. Posteriormente para cada test funcional se hicieron versiones de cada flujo con las variaciones de la tensión de polarización. Así por ejemplo, para la prueba de las versiones Trivium con salida de un bit (test 1.1) se generaron tres flujos de test, correspondientes a los casos de tensión de polarización típica, mínima y máxima (*01_tf_x1_TODO_TYP*, *02_tf_x1_TODO_MIN*, *03_tf_x1_TODO_MAX*). En total doce ficheros de flujo de test y doce ficheros de patrones de vectores para la realización del test funcional.

Tabla 5.4 Ficheros del flujo de test para el test funcional.

Test	Versiones Trivium	Flujo de test
1.1	Trivium_x1	01_tf_x1_TODO_TYP
	Trivium_x1mp	02_tf_x1_TODO_MIN
	Trivium_x1fp	03_tf_x1_TODO_MAX
1.2	Trivium_x2	04_tf_x2_TODO_TYP
	Trivium_x2mp	05_tf_x2_TODO_MIN
	Trivium_x2fp	06_tf_x2_TODO_MAX
1.3	Trivium_x8	07_tf_x8_TODO_TYP
	Trivium_x8mp	08_tf_x8_TODO_MIN
	Trivium_x8fp	09_tf_x8_TODO_MAX
1.4	Trivium_x16	10_tf_x16_TODO_TYP
	Trivium_x16mp	11_tf_x16_TODO_MIN
	Trivium_x16fp	12_tf_x16_TODO_MAX

En la Figura 5.13 se muestra la ventana de edición de flujos de test, en la que se puede apreciar cómo se van ejecutando secuencialmente los test funcionales de las versiones Trivium para el caso de salida de un solo bit.

En primer lugar el test funcional de la versión Trivium estándar (*T_x1*), después la de bajo consumo MPLP (*T_x1MPLP*) y por último la versión FPLP (*T_x1FPLP*). La definición del flujo de test lleva implícito un fichero de configuración de pines, niveles de tensión, temporización y ficheros de patrones. A su vez cada tarea de test tiene una configuración propia para seleccionar los niveles de tensión, temporización y vectores específicos.

Para ayudar a la ejecución y depuración del test se tiene la figura del triángulo, que indica una ejecución con éxito (color verde) y del octógono (color rojo), que indica un fallo.

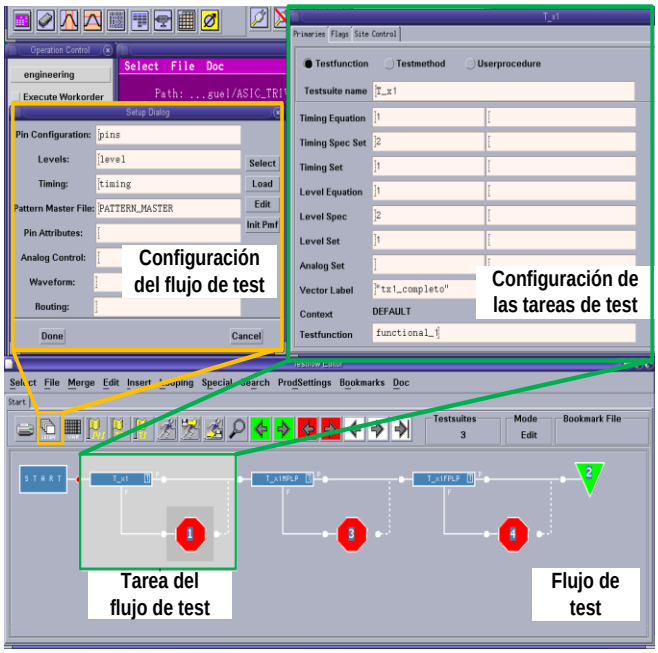


Figura 5.13 Editor de las tareas del flujo del test 1.1.

En la Figura 5.14 se muestran los resultados gráficos de las formas de onda con las salidas de las versiones Trivium de un bit, obtenidas a partir de un test funcional, con la herramienta de visualización temporal, *Timing Diagram* del equipo de test. Por ejemplo se observa cómo se obtienen las mismas formas de onda para las tres versiones de los Trivium. El bit de mayor peso del bus de salida (*key_out<15>*) contiene la salida del cifrador Trivium estándar de un solo bit. El bit *<14>* se corresponde con la salida de la versión FPLP y el bit *<13>* con la salida de la versión MPLP.

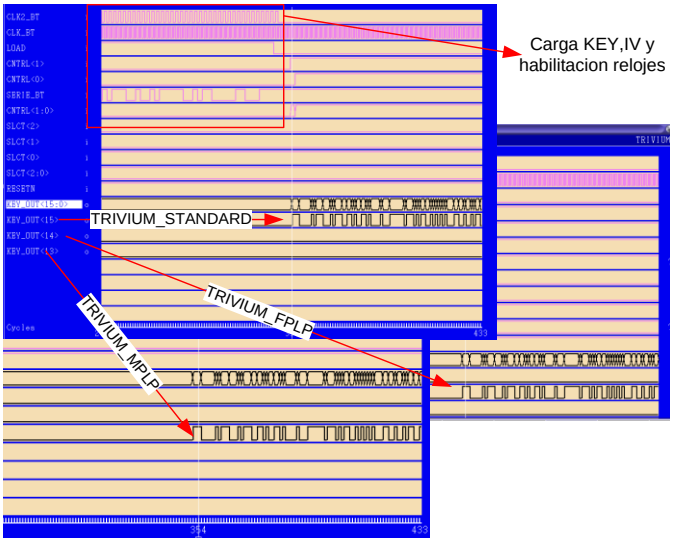


Figura 5.14 Formas de ondas para los Trivium de un bit.

Al realizar el test funcional se observó un comportamiento anómalo en los Trivium MPLP de un bit de salida. El hecho consistía en que para que funcionase correctamente, se debía de actuar en la selección de los relojes, activando tanto su reloj como el reloj de la versión estándar de 2 bits. Este comportamiento se repetía en todas las muestras. Este hecho, que en un principio pasó desapercibido en las simulaciones funcionales post-*layout*, llevó a hacer un estudio más detallado de la versión "x2" a nivel de simulación post-síntesis y post-*layout*. Con este análisis se descubrió un pequeño error en el diseño que no había sido detectado con las simulaciones anteriores. El problema se originó por un cambio en la etapa final del diseño impuesto por la introducción de los árboles de reloj. Como ya se comentó en el Capítulo 4, a los módulos de las versiones de bajo consumo MPLP multi-bit se les añadió una nueva entrada de reloj que se correspondía con el reloj dividido por dos (reloj *clkm*). De esta forma se consiguió que se cumpliesen las fuertes restricciones temporales que el funcionamiento de las versiones MPLP de los Trivium imponen al reloj dividido frente al reloj principal, como se comentó en el apartado 4.5.1 del Capítulo 4. Sin embargo hubo un error en el código *VHDL* modificado y el reloj dividido por dos del módulo MPLP de un bit quedó ligado al bit que activaba el reloj dividido que actuaba sobre el módulo de la versión del Trivium estándar de dos bits. Así pues, si se quiere activar el reloj del Trivium MPLP de un bit ("x1mp") hay que activar también el reloj de la versión de dos bits estándar. Este hecho fue comprobado también con las simulaciones funcionales.

Una vez modificados los patrones de entrada para tener esto en cuenta, se verificó que el comportamiento es idéntico al obtenido en las simulaciones funcionales, lo que comprobó la correcta funcionalidad de todas las versiones implementadas en

el chip. Una vez realizado el test se obtuvieron los resultados del test funcional paso/fallo mostrados en la Tabla 5.5. De las diez muestras encapsuladas todas pasaron correctamente los test funcionales en las tres condiciones de polarización.

Tabla 5.5 Resultados del test 1.1 en las muestras 1-10.

Versiones Trivium	Tensión	Muestra									
		1	2	3	4	5	6	7	8	9	10
Trivium_x1	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x1mp	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x1fp	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x2	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x2mp	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x2fp	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x8	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x8fp	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x16	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x16mp	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P
Trivium_x16fp	Max.	P	P	P	P	P	P	P	P	P	P
	Nom.	P	P	P	P	P	P	P	P	P	P
	Min.	P	P	P	P	P	P	P	P	P	P

P de paso. **F** de fallo

5.4. Pruebas de caracterización de los prototipos.

Las pruebas de caracterización consistieron en medir la frecuencia máxima de operación y el consumo de potencia de cada una de las versiones de las versiones propuestas del cifrador Trivium. Para la realización de las pruebas de medida de frecuencia máxima hay que comentar que el dato de frecuencia medido para cada cifrador está afectado por la lógica adicional que engloba el circuito de carga de la clave y vector de inicialización, los multiplexores para encaminar la salida específica de la versión Trivium elegida y la lógica de la generación de los diferentes relojes a cada Trivium.

Los patrones de test utilizados están basados en las pruebas funcionales, con patrones esperados generados por el propio equipo de test y varias tensiones de polarización, en concreto, las relativas a los casos de tensión típica, mínima y máxima que nos da el fabricante. Hay que hacer hincapié que el test calcula el dato de frecuencia del sistema implementado con el cifrador Trivium seleccionado, junto con los bloques anteriormente comentados. Para el cálculo de la medida de corriente se utilizaron también patrones de test basados en las pruebas funcionales, realizándose medidas de corriente instantánea y de promedio de consumo en todas las versiones Trivium, midiendo las corrientes de las alimentaciones. De estos valores se dedujo el consumo de potencia de todas las versiones del cifrador Trivium.

5.4.1. Frecuencia máxima de operación

Para la medida de la frecuencia máxima se utilizaron varios métodos. En un primer momento se usó una función de test incluida en el software del equipo de test y que permite, definiendo los valores de frecuencia y periodo como parámetros, realizar automáticamente la medida de frecuencia máxima, obteniendo los resultados en la propia ventana de resultados. Esta opción, aunque válida, resulta poco práctica cuando se tienen que testar muchas muestras pues no se pueden almacenar los resultados en un fichero y su automatización se hace difícil. Por ello se optó por generar una función de test específica para la medida de máxima frecuencia, con la ayuda de la programación en C++ y de las funciones específicas API (*application programming interface*) que proporciona el equipo. La parte más importante del programa principal de la función generada para este propósito, llamada *Frequency_max.cpp*, se presenta a continuación.

```
//HRESULT CCaptureTest::Frequency_max(STRING& input_string, double
_results[4])
HRESULT CCaptureTest::Frequency_max(double _results[4])
{
......
SPEC_SEARCH search_fmax("T",TM::TIM);
search_fmax.method(Binary).unit("ns").resolution(0.1);
search_fmax.execute();
DOUBLE Pval, Fval;
switch(search_fmax.getResultSpec()) {
   case TransitionPassFail :
   case TransitionFailPass :
     Pval = search_fmax.getPassVal();
     Fval = search_fmax.getFailVal();
   break;
   case AllPass :
   case AllFail :
   cout << "No transition found within the search interval" << endl;
   break;
}
cerr << "Periodo_fallo=  " << Fval << endl;
cerr << "Periodo_valido=  " << Pval << endl;
TEST(search_fmax);
cerr << endl << "fin test...." << endl;
return S_OK;
}
```

Con esta opción se consigue facilitar repetitividad. Además los resultados son mostrados en la ventana de información del equipo y también son guardados en un fichero de texto. Esta función de test va realizando test funcionales variando el parámetro del valor del ciclo de test (T) desde un valor máximo (100 ns) a un valor mínimo (20 ns), con saltos lineales de 0,1 ns, utilizando la función *SPEC_SEARCH*. Cuando el test funcional falla, se anota el valor del ciclo de test (*Periodo_fallo*) y del ciclo de test anterior a éste donde no hay fallo (*Periodo_valido*) con las variables *Fval* y *Pval*.

Los patrones de test utilizados se basan en los usados para el test funcional en cuanto a la carga de los vectores de la clave e inicialización pero en este caso se utilizan como patrones esperados los capturados por el propio equipo de test para una frecuencia baja. Sin embargo la configuración temporal es diferente a la de los test funcionales. Para poder llegar a frecuencias de más de 50 MHz se tiene un período de ciclo de test que genera cuatro pulsos de reloj como ya se comentó en el apartado 5.2.3.3.

Los datos de la frecuencia máxima se obtienen mediante el valor del ciclo de test donde no ha habido fallo (*Pval*) dado como $T_{\text{clk}} = T/4$, siendo la frecuencia máxima, $f_{\text{max}} = 1/T_{\text{clk}}$.

Los flujos de test, mostrados en la Tabla 5.6, se han basado en la prueba de verificación funcional del plan de test. Se han agrupado en un solo flujo de test los

tres test de frecuencia correspondiente a las versiones estándar y de bajo consumo MPLP y FPLP, según el número de bits de su salida.

Tabla 5.6 Ficheros de flujos de test para la caracterización de frecuencia.

Test	Versiones Trivium	Flujo de test
2.1	Trivium_x1	13_x1_fmax
	Trivium_x1mp	17_x1_fmax_MIN
	Trivium_x1fp	18_x1_fmax_MAX
2.2	Trivium_x2	14_x2_fmax
	Trivium_x2mp	19_x1_fmax_MIN
	Trivium_x2fp	20_x1_fmax_MAX
2.3	Trivium_x8	15_x8_fmax
	Trivium_x8mp	21_x1_fmax_MIN
	Trivium_x8fp	22_x1_fmax_MAX
2.4	Trivium_x16	16_x16_fmax
	Trivium_x16mp	23_x1_fmax_MIN
	Trivium_x16fp	24_x1_fmax_MAX

Cada uno de los test de frecuencia se ejecutó con configuraciones de tensión mínima, típica y máxima. Así pues, para la prueba de la versión Trivium con salida de un bit se generaron tres ficheros de flujo de test, correspondientes a los casos de tensión de polarización típica, mínima y máxima (*13_x1_fmax*, *17_x1_fmax_MIN*, *18_x1_fmax_MAX*). En total doce ficheros de flujo de test para la realización del test de frecuencia máxima.

Finalmente, una vez realizado el test se calculó la media y la desviación típica de las medidas para hallar la dispersión de los datos. No se apreciaron dispersiones grandes en las medidas, las cuales presentaron una desviación típica menor a 1.

Los datos de las frecuencias medias y su desviación se muestran en la Tabla 5.7, para el caso de polarización típica, en las diez muestras testadas.

La versión de Trivium estándar tiene siempre mejores datos de frecuencia máxima que las versiones de bajo consumo, penalizadas sobre todo por los árboles de reloj.

Tabla 5.7 Media de las medidas de frecuencia y su desviación típica.

T=25°C. Datos de frecuencia en MHz. Caso Típico		
Trivium	**MHz**	**Desviación típica**
Trivium_x1	128,6	0,6
Trivium_x1mp	104,5	0,4
Trivium_x1fp	114,3	0,6
Trivium_x2	122,3	1,0
Trivium_x2mp	118,2	0,6
Trivium_x2fp	116,9	0,7
Trivium_x8	115,2	0,7
Trivium_x8mp	100,8	0,9
Trivium_x8fp	86,1	0,5
Trivium_x16	98,5	0,8
Trivium_x16mp	97,5	0,8
Trivium_x16fp	75,2	0,9

Para facilitar el análisis, en la Figura 5.15 se representan gráficamente los datos de frecuencia máxima. Para cada una de las versiones de los cifradores Trivium se muestran, con las barras de error, los máximos y los mínimos de cada medida.

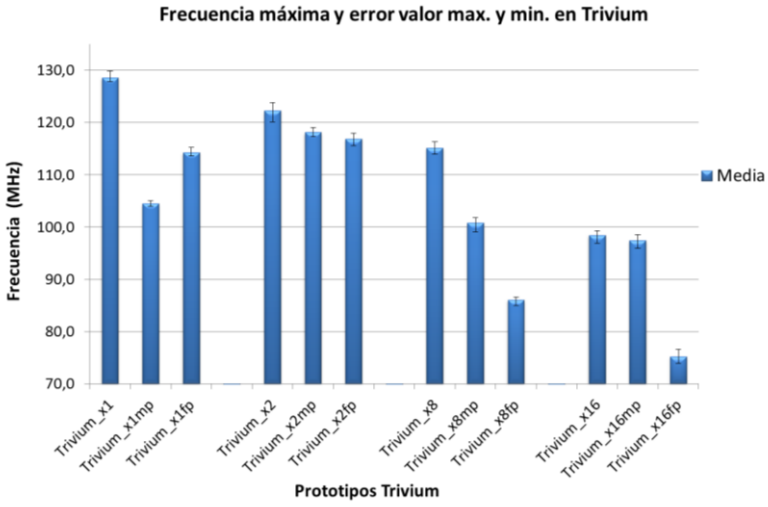


Figura 5.15 Resultados de los test de frecuencia máxima y desviación.

La versión de bajo consumo FPLP tiene una frecuencia máxima más baja que la versión MPLP excepto para el caso de salida de 1 bit, donde su frecuencia es mayor y para el caso de salida de 2 bits que son parecidas. Para los casos de salidas de 8 y 16 bits la versión MPLP tiene mejores datos de frecuencia. La penalización de los árboles de reloj y el aumento de la lógica combinacional en la versión FPLP se hace más acusada y los datos de frecuencia son peores.

Como nota discordante, las medidas de las versiones de 2 bits de bajo consumo alcanzan frecuencias más altas que las versiones de 1 bit. Para explicar este hecho se visualizó el *layout* de las diferentes versiones y se observó como en las versiones de un bit de bajo consumo sus celdas están más dispersas por el área del circuito integrado que las de la versión de 2 bits (mas compactada en área) teniendo como consecuencia unos tiempos de propagación mayores que repercuten en su frecuencia máxima.

En la Figura 5.16 se muestra la disposición en el *layout* de las versiones Trivium de bajo consumo de uno y dos bits.

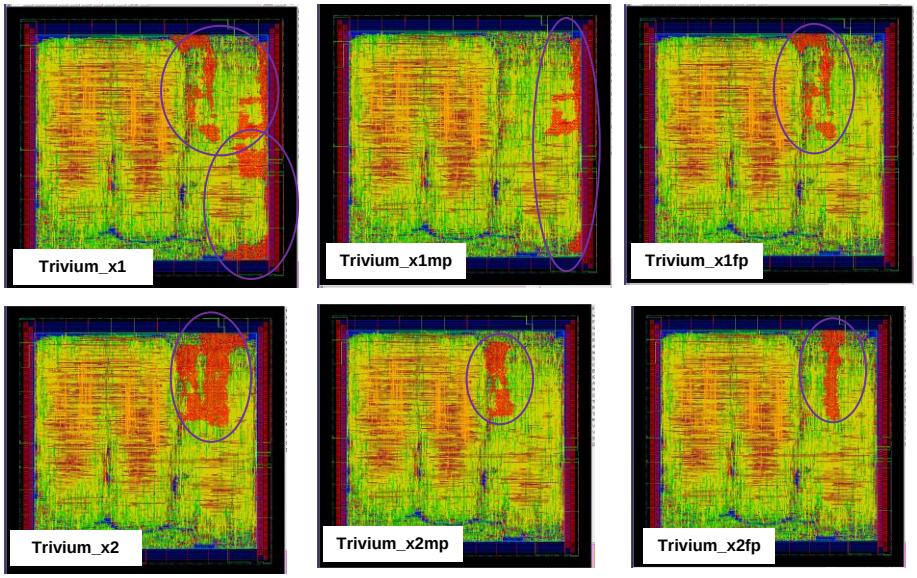


Figura 5.16 Disposición de las versiones Trivium "x1" y "x2" en el layout.

También se representan los datos para todas las versiones Trivium de la muestra 2 y el porcentaje de disminución de la frecuencia máxima con respecto a la versión estándar que es la que mejores prestaciones en frecuencia presenta. La Tabla 5.8 presenta estos datos (en MHz) según la versión de arquitectura Trivium, dependiendo del valor de tensión de polarización del anillo y del núcleo. Las demás medidas relativas a las otras muestras se muestran en el apéndice C de resultados.

La versión de un bit con arquitecturas para bajo consumo presenta frecuencias máximas del orden de un 12-21% menor respecto a la estándar mientras que para la versión con salida de dos bits, las tres versiones dan frecuencias máximas parecidas, solo un 4-5% menor en las arquitecturas de baja potencia. Esta diferencia es más acusada en las otras versiones, si bien es cierto que para el caso de 16 bits la versión MPLP da frecuencias del orden de la estándar. Esto nos lleva a pensar que

en estos casos los problemas derivados de los árboles de reloj afectan a todas las versiones del mismo modo y por ello sus resultados son bastante parecidos. Sin embargo las versiones de bajo consumo FPLP de ocho y dieciséis bits tienen mayores problemas derivados de los caminos de reloj en el *layout*, además de toda la lógica combinacional adicional añadida por su propia arquitectura que penaliza bastante el dato de frecuencia máxima. Se obtienen medidas de frecuencia máxima del orden de un 23-23% más baja que la de la versión estándar.

Tabla 5.8 Medidas de la frecuencia máxima en la muestra 2.

T=25°C Datos de frecuencia máxima (MHz). Muestra 2.						
Prototipos	**Condiciones polarización**			**Porcentaje disminución**		
Trivium	**MIN**	**TYP**	**MAX**	**%MIN**	**% TYP**	**% MAX**
Trivium_x1	117	129	142			
Trivium_x1mp	94	105	112	19	19	21
Trivium_x1fp	101	114	124	14	12	12
Trivium_x2	110	123	133			
Trivium_x2mp	106	118	129	4	4	4
Trivium_x2fp	104	117	127	5	5	4
Trivium_x8	102	114	124			
Trivium_x8mp	92	101	110	10	12	11
Trivium_x8fp	78	87	94	23	24	24
Trivium_x16	89	99	107			
Trivium_x16mp	88	98	106	1	1	1
Trivium_x16fp	66	75	83	26	24	23

La máxima frecuencia para las versiones de un solo bit se obtiene en la versión Trivium estándar de un bit de salida con 142 MHz (caso máxima polarización), y la menor con la versión de bajo consumo FPLP para salidas de 16 bits con 66 MHz (caso mínima polarización).

5.4.2. Variación de la frecuencia con la tensión de alimentación.

Se ha hecho un análisis más general de la variación de la frecuencia máxima con las tensiones de polarización, utilizando la herramienta del equipo de test denominada *shmoo plot,* la cual realiza una representación gráfica del comportamiento del dispositivo mientras se varían dos parámetros.

El *shmoo plot* lleva a cabo un test funcional para cada par de valores de dos parámetros que van cambiando dentro del intervalo específico de búsqueda. Los parámetros que se modifican pueden ser niveles de tensión, valores de temporización, etc.

Para la realización de la medida se definió un parámetro global (V_{DD}) de

polarización y se referenciaron todos los valores de tensión definidos en la configuración (incluidas las polarizaciones del núcleo y anillo, y los niveles lógicos de entrada y salida) en función de este parámetro. Este parámetro V_{DD} se varió entre 1,0 V y 2,75 V con resoluciones de 50 mV. En el caso de la frecuencia se varió el período del test (T) desde 100 ns a 20 ns con resoluciones de 2 ns, que equivale a frecuencias desde 40 MHz a 200 MHz.

En este caso el reloj está definido según se mostró en la Figura 5.12 para las medidas de frecuencia máxima. De la medida se obtiene una frecuencia máxima vinculada con el valor de tensión V_{DD} y el periodo de ciclo de test (T). Como ya se ha indicado el valor del período de reloj (T_{clk}) es función del período del ciclo de test (T), donde la frecuencia máxima de reloj viene dada por $T_{clk} = T/4$, siendo $f_{max} = 1/T_{clk}$.

Las definiciones para los niveles de tensión del circuito quedan como se muestra en la Tabla 5.9.

Tabla 5.9 Vinculación de las diferentes tensiones con V_{DD} para los *Shmoo Plot*.

Nombre	Parámetro	Shmoo	Mínimo-Máximo	Unidades
VDD_CORE	Alimentación núcleo	V_{DD}*0,48	0,48-1,32	V
VDD_RING	Alimentación anillo	V_{DD}	1,0-2,75	V
Vin	Tensión de entrada	V_{DD}		V
VIL	Tensión de entrada a nivel bajo	0		V
VIH	Tensión de entrada a nivel alto	V_{DD}		V
VOL	Tensión de salida a nivel bajo	V_{DD}*0,48		V
VOH	Tensión de salida a nivel alto	V_{DD}*0,485		V

La secuencia de test con los ficheros de los flujos de test seguida se basó en la prueba de caracterización de frecuencia máxima del plan de test y se muestra en la Tabla 5.10.

Como en los test anteriores se agruparon en un solo flujo de test los tres test de *shmoo plot* correspondientes a las versiones estándar y de bajo consumo MPLP y FPLP, según el número de bits a su salida. Así, por ejemplo, en el caso de salida de un bit se tiene el flujo de test *25_x1_shmoo*. En total, cuatro ficheros de flujo de test para la realización del test 3.

Tabla 5.10 Ficheros de test para la variación V_{DD}-frecuencia.

Test	Versiones Trivium	Flujo de test
3.1	Trivium_x1 Trivium_x1mp Trivium_x1fp	25_x1_shmoo
3.2	Trivium_x2 Trivium_x2mp Trivium_x2fp	26_x1_shmoo
3.3	Trivium_x8 Trivium_x8mp Trivium_x8fp	27_x1_shmoo
3.4	Trivium_x16 Trivium_x16mp Trivium_x16fp	28_x1_shmoo

Los *shmoo plot* correspondientes a las versiones Trivium de un bit de la muestra 2 se representan a continuación en la Figura 5.17. Los demás resultados de test se muestran en el Apéndice C de resultados. Los casos favorables del test se muestran en color verde y los de fallo en color rojo.

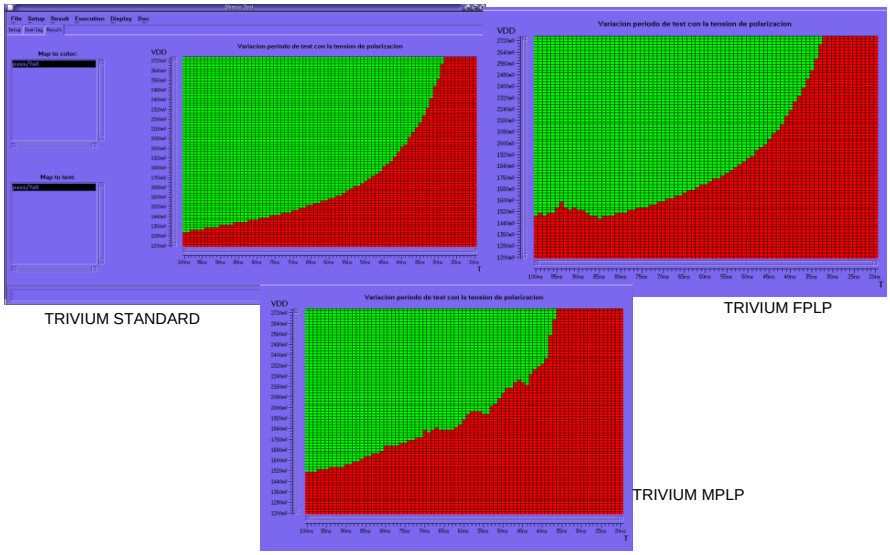


Figura 5.17 *Shmoo plot* para Trivium x1 en la muestra 2.

La versión estándar comienza a ser operativa con valores de V_{DD} menores que las versiones de bajo consumo MPLP y FPLP. También los valores de frecuencia para la misma V_{DD} son más altos. Los casos de test sin fallos son mayores en la versión estándar (color verde) comparándolos con las versiones de bajo consumo. En el caso de comparar las versiones de bajo consumo entre sí, los casos de test sin fallos

en la versión FPLP son mayores en la versión MPLP (color verde).

Finalmente se obtuvieron de las gráficas, los valores de la tensión de polarización mínima del anillo y del *core*, a los que el circuito comienza a operar correctamente, y la frecuencia de operación en este caso. Los resultados se muestran en la Tabla 5.11 para la muestra 2. Los valores de frecuencia máxima ya han sido calculados en el test anterior con los valores de tensión máxima de alimentación.

El prototipo ASIC comienza a ser operativo funcionalmente con tensiones más bajas que las mínimas indicadas por el suministrador de la tecnología aunque como se ve en las gráficas las prestaciones en frecuencia varían con la tensión. A partir de una polarización de 1,32 V de polarización del anillo y frecuencia de 40 MHz los prototipos Trivium proporcionan salidas correctas. Recordemos que según la tabla 5.2 la polarización mínima del anillo era 2,25 V.

Tabla 5.11 Datos de test para la variación V_{DD}-frecuencia muestra 2.

Test	Versiones Trivium	Vmínima (V) V_{DD_CORE}-V_{DD_RING}	Frecuencia mínima (MHz)
3.1	Trivium_x1	~0,63-1,32	40
	Trivium_x1mp	~0,72-1,5	
	Trivium_x1fp	~0,71-1,48	
3.2	Trivium_x2	~0,63-1,32	40
	Trivium_x2mp	~0,66-1,38	
	Trivium_x2fp	~0,71-1,48	
3.3	Trivium_x8	~0,67-1,4	40
	Trivium_x8mp	~0,71-1,48	
	Trivium_x8fp	~0,74-1,54	
3.4	Trivium_x16	~0,69-1,44	40
	Trivium_x16mp	~0,72-1,5	
	Trivium_x16fp	~0,81-1,68	

5.4.3. Medida del consumo instantáneo de corriente

Con objeto de poder medir el consumo de cada uno de los cifradores Trivium,en el circuito integrado se ha previsto un mecanismo para deshabilitar individualmente los relojes de cada uno de los cifradores Trivium implementados, mediante la carga en serie de una palabra de control de 12 bits, como ya se describió en el Capítulo 4, apartado 4.1.

La medida del consumo en las diferentes propuestas de arquitectura Trivium tiene el problema del efecto que aportan los diferentes árboles de reloj del circuito. Como se recordará, a partir del reloj principal de entrada se han generado diversos árboles de reloj para los cifradores, que se detallaron en el Capítulo anterior.

Recordando lo ya comentado en él, el árbol del reloj principal está dimensionado para que llegue a todos las versiones en buenas condiciones y no para una versión Trivium determinada. Además, para el caso de las versiones Trivium de bajo consumo MPLP y FPLP se tienen que añadir otros árboles de reloj correspondientes a los diferentes relojes divididos que necesitan dichas propuestas. En el caso de las versiones de bajo consumo MPLP estos árboles de reloj se encuentran fuera de los cifradores, como es el caso también del reloj principal, mientras que los relativos a las versiones FPLP se encuentran dentro de cada cifrador y, por tanto, penalizan más a éste. Los relojes de cada cifrador pueden ser apagados mediante señales de selección configurables por el usuario pero, aunque se apague la entrada de reloj del cifrador, se tiene un consumo extra debido a la lógica generada para cada árbol reloj.

También hay que sumar siempre el consumo de los multiplexores que se han tenido que introducir para conectar todas las salidas de todas las versiones de los cifradores Trivium con las 16 salidas del circuito integrado.Las medidas de consumo instantáneo se realizaron utilizando el módulo digitalizador WDA disponible en el equipo de test. Las características de este módulo analógico se detallaron en la Tabla 5.1.

En la placa DIB se han colocado dos resistencias en serie en el camino de las alimentaciones del circuito para medir la caída de tensión en ellas, así como el conexionado necesario para llevar a cabo la medida en el equipo de test. La medida se lleva a cabo en la fuente de consumo del *core* que es la que da la alimentación a las celdas internas. La Figura 5.18 muestra un esquema del conexionado entre el módulo digitalizador y la lógica implementada en la placa DIB para la medida.

Se necesita también una señal de disparo para activar el digitalizador conectado a un canal digital (10102). La medida de la corriente instantánea ha consistido en el muestreo de 200 puntos por cada ciclo de reloj. Se ha trabajado a la frecuencia máxima del digitalizador (41 MHz), con lo que la frecuencia de operación de los cifradores es de 200 KHz. Los datos de tensión en la resistencia fueron capturados por el digitalizador y almacenados en un fichero ASCII que posteriormente es procesado para representar el consumo instantáneo de corriente.

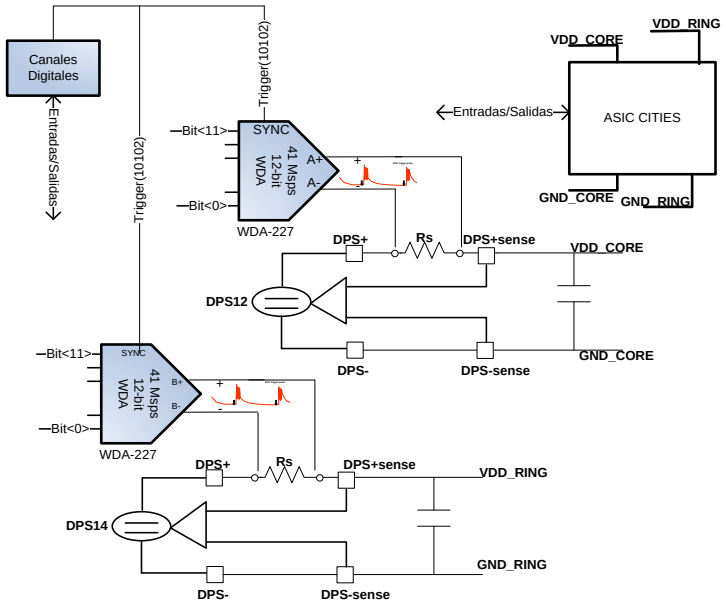


Figura 5.18 Conexionado hardware del módulo analógico WDA y las DPS.

Un ejemplo de la tensión obtenida en bornes de la resistencia en serie de 10 ohm para el caso de la versión estándar del Trivium con salida 1 bit se representa en la Figura 5.19, representándose 1000 puntos de medida, siendo la gráfica muy parecida para las otras versiones MPLP y FPLP.

De estas medidas es difícil comparar el consumo de potencia de las tres versiones pues el ruido y el efecto de los árboles de reloj impiden tener diferencias apreciables entre unas versiones y otras. Solamente es claramente apreciable el pico de consumo debido a los flancos del reloj principal.

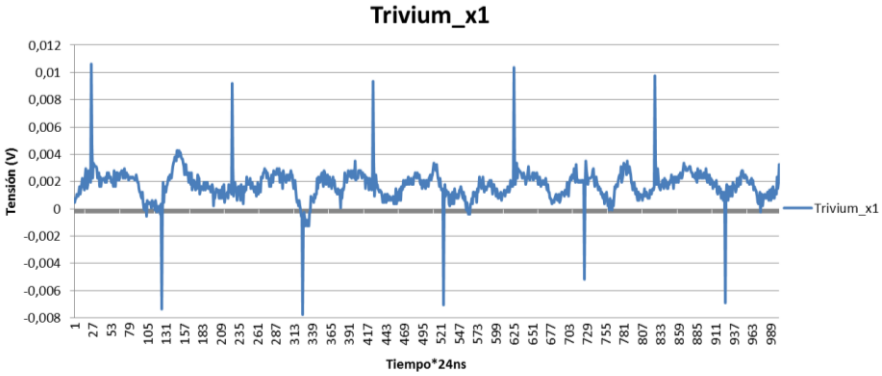


Figura 5.19 Medida de la tensión en bornes de la resistencia serie, Trivium 1 bit.

Con estos valores de frecuencia de operación la corriente de los cifradores es muy pequeña y los valores de caída en la resistencia serie son del orden de mV con lo que la medida con el digitalizador no es lo suficientemente precisa para darla por válida, además se enmascara con el ruido intrínseco del sistema. Si se aumenta la frecuencia de operación de los cifradores se reducirá el número de muestras en cada período con lo que se perderá precisión en la medida instantánea.

Para intentar solucionar este problema se optó por realizar una medida diferencial. Se trata de un procedimiento que conlleva dos medidas de tensión. En primer lugar se realiza una medida de la caída de tensión en la resistencia con todas las versiones Trivium funcionando y en segundo lugar otra medida con todas las versiones funcionando menos la versión Trivium elegida que se haya elegido para medir. La resta de ambas dará como resultado el consumo de la versión Trivium seleccionada. De esta forma se eliminan todos los caminos comunes de los árboles de reloj.

Las curvas de tensión capturada en bornes de la resistencia serie para las muestras a una frecuencia de operación de 200 KHz muestran también valores con una pobre validez para realizar una comparación entre las versiones ya que las medidas siguen experimentando efectos similares a los del método anterior.

Un ejemplo de ello se muestra en la Figura 5.20 con la versión FPLP de un bit. Por un lado se representan 1000 puntos de medida de la tensión en bornes de la resistencia funcionando todos los Trivium ("Trivium x1 todo") y por otro la medida de todos los Trivium funcionando menos el Trivium FPLP ("Trivium x1 todo menos FPLP").

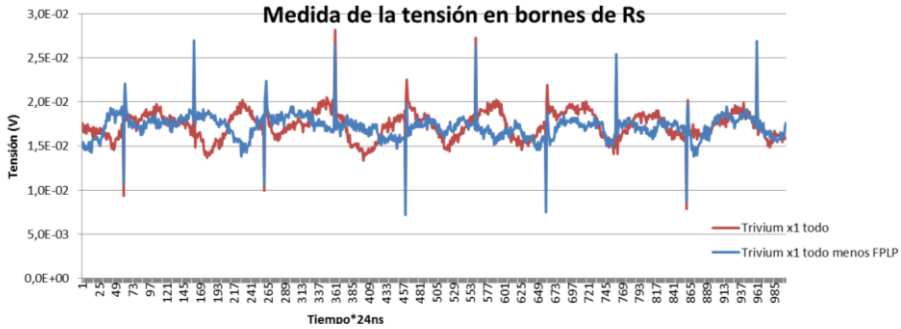


Figura 5.20 Medida de la tensión en bornes de la resistencia, Trivium FPLP 1 bit.

Se observan los picos de tensión (y por tanto de corriente) en cada flanco de reloj del Trivium (cada 2500 ns), siendo de amplitud algo menor los que se producen con la versión de "Trivium x1 todo menos FPLP". La resta de la medida de las tensiones mostrada en la Figura 5.21 no permite obtener las curvas o traza de corriente de consumo debido a que el valor de la medida es comparable con el ruido.

Figura 5.21 Medida de la resta de tensiones para Trivium FPLP 1 bit.

Finalmente se ha calculado el consumo promedio de corriente de cada una de las versiones Trivium de un solo bit de salida, dividiendo el valor de la tensión por el valor de la resistencia serie.

Los resultados de la Tabla 5.12 muestran los valores de corriente de los cifradores Trivium con salida de 1 bit junto con el porcentaje de mejora de corriente de las versiones de bajo consumo. Se obtienen valores de mejora en las versiones de bajo consumo, de entre un 35-40%.

Tabla 5.12 Consumo de corriente promedio Trivium x1.

VDD=1,2 V-T=25°C-f=200 KHz-		
	Consumo de Corriente promedio	
Trivium	**µA**	**% de Mejora**
Trivium_x1	1,64	
Trivium_x1mp	0,98	40
Trivium_x1fp	1,06	35

Pese a que los valores promedio obtenidos a partir de las medidas instantáneas de corriente ofrecen unos valores en el rango de lo esperado, como los valores instantáneos tienen mucho ruido, este método no puede ser dado como válido. Se va a utilizar un segundo método para medir el consumo, esta vez de forma promedio. Para ello se tomarán los datos de la intensidad suministrada por las fuentes de alimentación.

5.4.4. Medida del consumo promedio de corriente.

Para la medida del consumo promedio en operación de las versiones Trivium se ha utilizado la posibilidad que ofrecen las fuentes de polarización (DPS) del equipo de test *Agilent 930000*, que permiten que cada canal de las fuentes de alimentación pueda medir la corriente que proporcionan al dispositivo bajo prueba. Se detalla en la Tabla 5.13 la resolución y la precisión en la medida de corriente del módulo DPS,

en función del rango de intensidades medidas.

Para la medida de la corriente promedio de cada cifrador Trivium también se ha utilizado un método diferencial. En primer lugar se llevó a cabo una medida de la corriente con todas las versiones Trivium funcionando. Después se realizó una medida de la corriente con todas las versiones Trivium funcionando menos la versión elegida de la que se quiera calcular la corriente. Por último se restaron las dos corrientes. El resultado es la corriente consumida por la versión Trivium elegida.

Es importante para el test adaptar la frecuencia de operación de los cifradores Trivium para que el consumo del circuito esté en el rango de medida de corriente adecuado y así obtener la mayor precisión posible. En nuestro caso se ha elegido el rango de –10 a +10 mA con resoluciones de 1 μA y precisión de ± 10 μA ±0,1% para las medidas de corriente ya que las medidas sin reloj demandan ya más de 100 μA en las fuentes.

Tabla 5.13 Especificaciones de corriente en las placas de DPS.

Modo	Rango	Resolución	Precisión	Comentarios
Corriente	–8 to +8 A	1 mA	± 20 mA ±0.1%	Rango 4.
	–0,3 to +0,3 A	30 μA	± 450 μA±0.15%	Rango 3.
	–10 to +10 mA	1 μA	± 10 μA ±0.1 %	Cload < 100 μF. Rango 2.
	–100 to +100 μA	10 nA	± 100 nA ±0.1%	Cload < 1 μF. Rango 1.

Para conseguir facilidades de programación, repetitividad y facilidad para almacenar los resultados en un fichero, la medida se incluyó en una función de test junto con el test funcional. Para ello se generó una función de medida de corriente controlada por los comandos *firmware* que aparecen a continuación.

Se ha utilizado el comando *DPSE* (*Define Power Supply External trigger measurement*) para configurar las medidas junto con unos límites de paso/fallo de la medida. Después se ha ejecutado un test funcional con el comando *PSEE* (*Power Supply External trigger measurement Exec*), el cual mide la corriente cuando la señal de disparo externa se activa. Esto se repite un número programable de veces. Los resultados de cada medida son almacenados en los registros de la memoria y mediante el comando *PSER* (*Power Supply External trigger measurement Read results*) se han obtenido los resultados del test previamente ejecutado.

```
// CorrienteTest.cpp
HRESULT CCorrienteTest::DPS_ExtTrig_Idd_Medida(STRING& DPS_name, INT
numPts, INT avg_count, DOUBLE lowLim, DOUBLE highLim, FLOAT t_wait,
double _results[4])
{
    // add your codes
   string fwtask,fwanswer;
   WAIT_TIME(t_wait);// additional wait time from previous TestSuit -->
relays etc --> e.g. 2ms
    // add your codes
    fwtask = "DPSE 1,(";
    fwtask += DPS_name;
   fwtask += "),#400251,04,-0.0001000,0.0010000";
    fwtask += "\n";
    FW_TASK(fwtask,fwanswer);
    fwtask = "PSEE 1,,";
    fwtask += avg_count;//Number of averages e.g.5
    fwtask += ",(@)\n";
    FW_TASK(fwtask,fwanswer);
    fwtask = "PSER? 1,PVAL\n";
    FW_TASK(fwtask,fwanswer);
    cerr << "1." << fwanswer << endl;
    int Idd_pass=0;
    if (fwanswer[7]=='P') Idd_pass=1;
    TEST(DPS_name,"Idd_max",Idd_pass);
    return S_OK;
}
```

La señal de disparo externa que activa la medida de corriente en las placas de polarización debe tener una anchura de pulso entre 300 ns y 100 μs mientras que el intervalo entre señales de disparo debe ser al menos de 40 μs. También su amplitud debe ser mayor que 1 V. Esta señal es añadida y configurada manualmente en los patrones de prueba.

La secuencia para la medida de la corriente conlleva tres pasos principales:

- Conexión del equipo de test: La medida de la corriente será incorrecta si se mide demasiado rápido después de conectar el equipo (menos de 5 ms). El tiempo de transición dependerá del entorno del test (capacidad de carga, dispositivo bajo prueba, etc).
- Ejecución de un test funcional: No se realizará ninguna medida mientras se realiza un test funcional.
- Medida de la corriente de operación: Se lleva a cabo la medida de la corriente que será ya estable y constante.

Los test tienen que hacer uso, entre otras, de la señal de entrada *SLCT_BT<2:0>* que selecciona los multiplexores de salida para acondicionarlos a la selección de la versión Trivium elegida y también se necesita la configuración adecuada del registro serie del ASIC para seleccionar adecuadamente el reloj de las diferentes

propuestas Trivium (Tabla 4.2 y Tabla 4.3 del Capítulo 4).

La selección de los multiplexores con la señal *SLCT_BT<2:0>* penaliza el consumo de potencia pues los multiplexores consumen más potencia si sus entradas y salidas están conmutando que si por el contrario permanecen estáticas. Lo ideal hubiese sido tener disponible todas las salidas de las diferentes versiones (81 salidas) pero el número de *pads* entonces sería prohibitivamente alto.

Los flujos de test llevados a cabo van calculando las corrientes promedio de operación de las versiones Trivium a las frecuencias de operación de 20 MHz y 50 MHz, mientras se realiza un test funcional, con la polarización de las fuentes en modo nominal (típico).

En primer lugar, se calcula la medida de la corriente de operación de todas las versiones Trivium funcionando y la de todas las versiones funcionando menos la elegida. Estas medidas se guardan en un fichero y restando ambos valores se halla la corriente de operación de la versión Trivium elegida. La potencia se calcula multiplicando la intensidad por el valor de la fuente de polarización del *core* (1,2 V). También se realizan medidas de la corriente de operación sin el reloj principal funcionando, y activando solo el reloj que afecta a la versión Trivium a testar.

Los flujos de test llevados a cabo son mostrados en la Tabla 5.14. Consistieron en agrupar en un solo flujo los tres test de medida de corriente consumida correspondientes a las versiones estándar y de bajo consumo MPLP y FPLP, según el número de bits a su salida. Posteriormente se hicieron para cada test versiones de cada flujo con las diferentes frecuencias de operación.

Tabla 5.14 Flujos de test para la medida de corriente promedio.

Test	Versiones Trivium	Flujo de test
4.1	Trivium_x1 Trivium_x1mp Trivium_x1fp	30_tx1_extDPS_20MHz 32_tx1_extDPS_50MHz
4.2	Trivium_x2 Trivium_x2mp Trivium_x2fp	33_tx2_extDPS_20MHz 35_tx2_extDPS_50MHz
4.3	Trivium_x8 Trivium_x8mp Trivium_x8fp	36_tx8_extDPS_20MHz 38_tx8_extDPS_50MHz
4.4	Trivium_x16 Trivium_x16mp Trivium_x16fp	39_tx16_extDPS_20MHz 41_tx16_extDPS_50MHz

Así por ejemplo, para la prueba de las versiones Trivium con salida de un bit (test 4.1) se generaron dos flujos de test, correspondientes a los casos de las frecuencias de operación, 20 MHz, y 50 MHz (*30_tx1_extDPS_20MHz, 32_tx1_extDPS_50MHz)*.

En total ocho ficheros de flujo de test y ocho ficheros de patrones de vectores para la realización del test de medida de corriente.

Para el caso de la frecuencia de operación de 50 MHz, se calculó la potencia media y la desviación típica a partir de los datos de corriente promedio medidos, para hallar la dispersión de los datos medidos.

En la Tabla 5.15 se muestran los datos del consumo medio y su desviación para el caso de polarización típica y los datos obtenidos en las diez muestras testadas. No se aprecian dispersiones grandes en las medidas, con una desviación típica menor que 20.

Las medidas de corriente cuando no se tiene ningún reloj activado dan valores cercanos a la centena de microamperios siendo por tanto su valor de consumo alrededor de 120-150 μW.

Tabla 5.15 Media de la medida de potencia y su desviación típica.

Datos de consumo en μW. Caso Típico. 50 MHz.		
Trivium	**Media**	**Desviación**
Trivium_x1	572,0	13,3
Trivium_x1mp	271,4	16,9
Trivium_x1fp	274,2	13,6
Trivium_x2	775,4	9,9
Trivium_x2mp	314,1	7,4
Trivium_x2fp	287,0	12,0
Trivium_x8	537.9	11,8
Trivium_x8mp	492,0	7,7
Trivium_x8fp	506,5	9,0
Trivium_x16	566,4	10,3
Trivium_x16mp	566,8	12,4
Trivium_x16fp	684,2	18,7

Para facilitar el análisis de los datos se representa también gráficamente la medida en la Figura 5.22, donde se muestran un diagrama de barras de error con los valores máximos y mínimos de cada versión Trivium.

Analizando los resultados, se observa cómo la versión estándar de dos bits tiene un consumo visiblemente mayor al esperado. Esto es debido al hecho, ya comentado anteriormente en la verificación funcional de que, por error, para activar el reloj del módulo de un bit MPLP ("*x1mp*") hay que activar también el reloj de la versión de dos bits estándar ("*x2*"). Esto lleva a que la medida de la corriente en la versión "*x2*" no sea correcta pues cuando se está ejecutando el flujo de test que activa todas las versiones menos la del Trivium "*x2*", se anula también el reloj dividido que va a la versión "*x1mp*". El dato de corriente que se obtiene será menor que el esperado. Con lo cual cuando se restan las dos corrientes el valor resultante será mayor que el

esperado al sumarse la componente de corriente del reloj "*x1mp*".

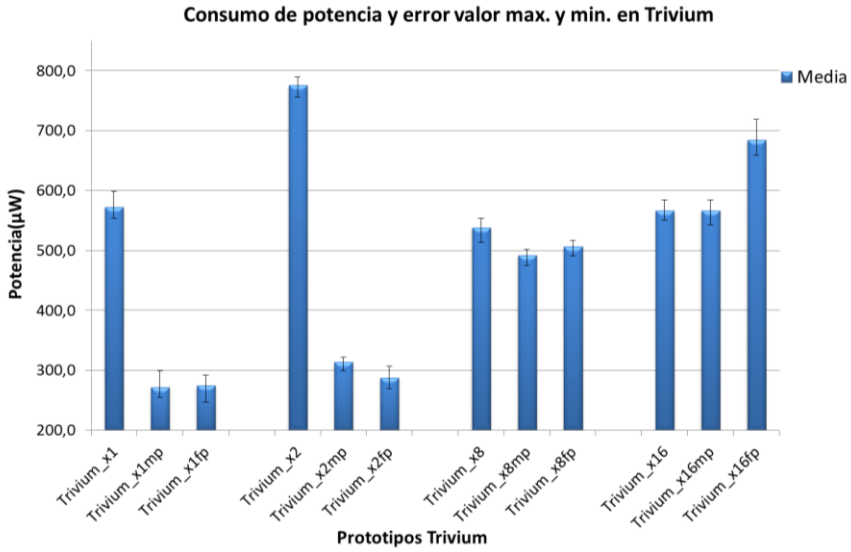


Figura 5.22 Gráfica de resultados del consumo a 50 MHz y su desviación.

También la medida de corriente de la versión "*x1mp*" es algo menor que el esperado pues cuando se está ejecutando el flujo de test que activa todas las versiones menos la del Trivium "*x1mp*", hay parte del módulo que sigue funcionando, en concreto, la que se ve afectada por el reloj dividido que no se ha apagado al depender del reloj de la versión "*x2*". El dato de corriente que se obtiene será mayor que el esperado. Con lo cual cuando se restan las dos corrientes el valor resultante será menor que el esperado al no sumarse la componente de consumo del reloj dividido.

La opción elegida para subsanar en lo posible este hecho y poder realizar un análisis más realista es hacer que el valor de consumo para el caso "*x2*" sea tomado como en el caso "*x1*". En la Tabla 5.16 se presentan los nuevos datos y porcentajes de mejora (valores positivos) o de empeoramiento del consumo (valores negativos) de las versiones de bajo consumo frente a las versiones estándar. En el caso de "*x1mp*" no se hace nada y se deja el valor medido, ya que es difícil computar el dato no medido.

Se aprecia que las versiones de bajo consumo MPLP y FPLP mejoran el consumo de potencia respecto de la versión estándar hasta la versión de 8 bits donde ya no se aprecia esa mejora. Es importante destacar que, en el caso "*x1*" y "*x2*", se reduce a casi la mitad el consumo respecto del Trivium estándar.

En cuanto a la comparación entre la versión de bajo consumo MPLP y FPLP, se observó que tenían valores similares en el caso "*x1*", "*x2*" y "*x8*", aunque la versión

FPLP tiene consumos ligeramente menores que la versión MPLP hasta el caso "*x8*". Para el caso "*x16*", la versión FPLP es claramente peor que la versión MPLP.

Tabla 5.16 Nuevo consumo de potencia y porcentaje de mejora.

Datos de consumo en µW. Caso Típico. f=50 MHz.		
Trivium	**Media**	**% Mejora**
Trivium_x1	572,0	
Trivium_x1mp[1]	271,4	53
Trivium_x1fp	274,2	52
Trivium_x2[2]	572,0	
Trivium_x2mp	314,1	45
Trivium_x2fp	287,0	50
Trivium_x8	537,9	
Trivium_x8mp	492,0	9
Trivium_x8fp	506,5	6
Trivium_x16	566,4	
Trivium_x16mp	566,8	0
Trivium_x16fp	684,2	-21

[1*] Valor menor que el esperado. [2] Valores de la versión "x1"

Por último, en la Figura 5.23 se presenta un diagrama de barras con los porcentajes de mejora en los consumos. Los porcentajes son positivos cuando el consumo es menor que el del estándar y negativo en caso contrario. Obviamente el valor de mejora del estándar es 0. Son realmente destacables las grandes reducciones de las versiones "x1" y "x2".

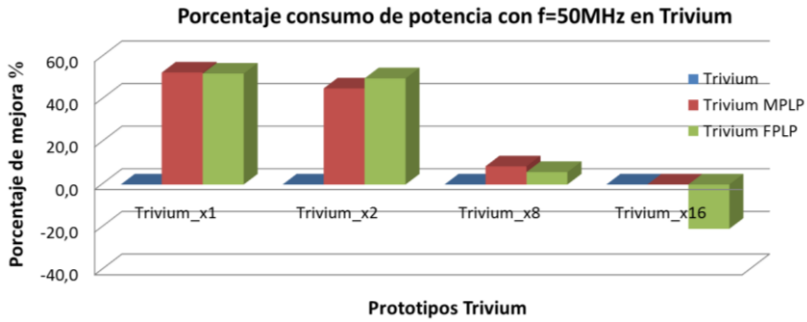


Figura 5.23 Porcentaje de mejora en el consumo post-*layout*.

A continuación se van a mostrar los datos de consumo medido a las dos frecuencias de operación elegidas sobre una de las muestras, en concreto sobre la muestra 2. En la Tabla 5.17 se muestran los datos de consumo medido y los porcentajes de mejora, aunque a diferencia de la Tabla 5.15 ahora se muestran datos del consumo medido a distintas frecuencias. Se han añadido los datos de consumo modificados y su nuevo porcentaje entre paréntesis. En el apéndice de resultados se muestran los resultados de todas las muestras.

Tabla 5.17 Consumo de potencia versiones Trivium medidas con *Agilent 93000.*

Caso típico. Muestra 2.				
Prototipos	**Consumo de potencia (µW)**		**% Mejora Consumo de potencia**	
Trivium	**20 MHz**	**50MHz**	**20 MHz**	**50 MHz**
Trivium_x1	221	558		
Trivium_x1mp[1]	105[1]	255[1]	52,5	54,4
Trivium_x1fp	109	273	50,8	51,0
Trivium_x2[2]	341[2] (221[3])	771[2] (558[3])		
Trivium_x2mp	135	318	60,4 (39)	58,7 (43,0)
Trivium_x2fp	120	300	64.8 (45,8)	61.2 (46,3)
Trivium_x8	213	543		
Trivium_x8mp	183	502	14,0	7,6
Trivium_x8fp	202	517	5,3	4,8
Trivium_x16	221	573		
Trivium_x16mp	225	562	-1,7	2,0
Trivium_x16fp	273	685	-23,7	-19,6

[1*] Valores menores que los esperados. [2] Valores mayores que los esperados.

[3] Valores de la versión "x1"

Como se comentó anteriormente para no tener datos tan dispares se toma el valor de consumo de la versión "*x2*" como la de "*x1*". Los datos del consumo "*x1mp*" son también algo menores que los que debería por los motivos ya comentados anteriormente.

El análisis de los resultados obtenidos en ambas frecuencias es muy similar al realizado en la Tabla 5.16, donde se aprecia que la versión Trivium estándar tiene siempre peores datos de consumo que las versiones de bajo consumo excepto para el caso "*x16*" donde su consumo es más parecido a la de la versión MPLP y mucho mejor que para la versión FPLP, debido sobre todo a la penalización de los árboles de reloj y la lógica combinacional adicional añadida conforme aumenta el número de bits a su salida.

Los resultados del análisis de datos de consumo permiten concluir que las dos versiones implementadas, MPLP y FPLP, en los casos de 1 y 2 bits consiguen reducir enormemente el consumo de la versión estándar del Trivium mientras que en el caso de 8 bits aunque significativa, la reducción es bastante menor. En el caso de 16 bits la versión MPLP tiene un consumo comparable al estándar, mientras que la versión FPLP tiene un consumo mayor.

Se ha caracterizado también el consumo de potencia al cambiar las condiciones de alimentación de las fuentes de alimentación del anillo y *core*, a valores mínimos y máximos. Solamente se han tomado las medidas para una frecuencia de 50 MHz. Se han obtenido los resultados mostrados en la Tabla 5.18 junto con el porcentaje de mejora de los mismos respecto a la versión estándar. Los datos del consumo "*x2*" y "*x1mp*" no son los correctos por los motivos ya comentados anteriormente. Los

porcentajes de la versión "*x2*" son tomados con los datos de "*x1*".

Las conclusiones a la vista de los resultados son las mismas que para el caso típico, el porcentaje de mejora en el consumo se mantiene también en las alimentaciones extremas.

Las versiones de bajo consumo MPLP y FPLP tiene siempre mejores datos de consumo que las versiones Trivium estándar excepto para el caso "*x16*" donde el consumo de la versión estándar es más parecido a la de la versión MPLP y mucho mejor que para la versión FPLP.

Tabla 5.18 Consumo de potencia a 50 MHz con diferentes casos V_{DD}.

f=50 MHz. Muestra 2. Caso V_{DD} mínimo-típico-máximo						
	Consumo de potencia (µW)			**% Mejora Consumo de potencia**		
Trivium	**MIN**	**TYP**	**MAX**	**MIN**	**TYP**	**MAX**
Trivium_x1	492	558	721			
Trivium_x1mp[1]	243[1]	255[1]	321[1]	51	54	55
Trivium_x1fp	249	273	334	49	51	54
Trivium_x2[2]	650	771	980			
Trivium_x2mp	273	318	391	45[3]	43[3]	46[3]
Trivium_x2fp	256	300	371	48[3]	46[3]	49[3]
Trivium_x8	479	543	667			
Trivium_x8mp	445	502	622	7	8	7
Trivium_x8fp	445	517	618	7	6	7
Trivium_x16	472	573	680			
Trivium_x16mp	462	562	684	2	2	-1
Trivium_x16fp	581	685	844	-23	-20	-24

[1] Valores menores que los esperados. [2] Valores mayores que los esperados.

[3] Valores corregidos, se toman los datos de *Trivium_x1* para el porcentaje.

Para verificar si este mecanismo de medida produce valores razonables y parecidos a los obtenidos con la medida instantánea de corriente, se volvió a realizar la medida de consumo de corriente con las fuentes de polarización para el cifrador Trivium de un bit de salida, pero ahora para una frecuencia de reloj de 200 KHz. Los resultados obtenidos se muestran en la Tabla 5.19.

Tabla 5.19 Consumo de corriente promedio Trivium x1.

Trivium	**Caso típico-f=200 KHz. Método DPS**		**Caso típico-f=200 KHz. Método WDA**	
	Consumo de Corriente promedio		**Consumo de Corriente instantánea.**	
	µA	**% de Mejora**	**µA**	**% de Mejora**
Trivium_x1	2.02		1.64	
Trivium_x1mp	1.13	44	0.98	40
Trivium_x1fp	1.37	32	1.06	35

Se puede observar que los valores obtenidos son algo mayores que los calculados mediante el digitalizador (método WDA) pero los porcentajes de mejora son muy parecidos, mostrando mejoras del 32-44% frente a la versión estándar

5.4.5. Comparación de medidas experimentales y post-*layout*.

Finalmente se ha realizado una comparación de los resultados obtenidos en laboratorio sobre el prototipo ASIC con las medidas obtenidas en las simulaciones post-síntesis y post-*layout*.

El principal interés es comparar las medidas realizadas en el laboratorio con el equipo de test *Agilent 93000* con las medidas obtenidas de las herramientas de CAD. La Tabla 5.20 muestra los resultados de consumo para los tres casos y las mejoras encontradas, frente a la versión Trivium estándar. Este análisis muestra que todas las medidas presentan valores similares en los porcentajes de mejora de las versiones de bajo consumo, lo que avala las propuestas de bajo consumo Trivium MPLP y FPLP desarrolladas en esta Tesis. Sin embargo, los valores de cada Trivium en sus medidas post-síntesis, post-*layout* y experimental difieren entre sí, lo cual es debido a las diferencias en cuanto al modo, método y condiciones de la medida.

Por un lado, en el cálculo de consumo de potencia post-síntesis y post-*layout* se utilizan las herramientas CAD. En el caso de post-síntesis se manejan los parámetros de consumo interno de las celdas modelados estadísticamente, dados en los ficheros de la librería de celdas del fabricante, junto con la actividad de conmutación estimada a partir de simulaciones. Se utiliza la tensión de polarización típica y el consumo de potencia típico para las celdas de librería.

Por otro lado, en el cálculo de consumo de potencia post-*layout* se manejan los parámetros de consumo interno de las celdas, junto con la actividad de conmutación estimada a partir de simulaciones, además de añadir los datos de las capacidades de las interconexiones y de los árboles de reloj de la implementación física. Esta estimación es más precisa que la de post-síntesis. Se utiliza la tensión de polarización mínima y los datos de potencia de las celdas de la librería de peor caso o máximos.

Para la medida en el laboratorio de los prototipos Trivium del ASIC, la medida de corriente se realiza en el circuito integrado que se halla en una placa de prototipado y se utiliza como instrumento de medida un equipo de test. La medida de corriente se ha basado en la medición de la corriente en las fuentes de alimentación a temperatura ambiente bajo condiciones de polarización mínima, máxima y típica.

Tabla 5.20 Comparación de los consumos de potencia.

Frecuencia 50 MHz - Muestra 2.								
Versiones Trivium								
	Consumo de potencia (μW)				% Mejora del Consumo			
Trivium	Post-Sintesis[1] 1,2V	Post-Layout[2] 1,08V	ASIC 1,2V	ASIC 1,08V	Post-Sintesis[1] 1,2V	Post-Layout[2] 1,08V	ASIC 1,2V	ASIC 1,08V
x1	384	331	558	492				
x1mp[4]	253	227	255[4]	243[4]	34	31	54	51
x1fp	212	180	273	249	44	46	51	49
x2[3]	390	340	771[3]	650[3]				
x2mp	257	232	318	273	34	32	43	45
x2fp	222	187	300	256	43	45	46	48
x8	409	396	543	479				
x8mp	315	330	502	445	22	17	8	7
x8fp	262	335	517	445	35	15	5	7
x16	424	419	573	472				
x16mp	375	424	562	462	11	-1	2	2
x16fp	355	512	685	581	16	-22	-20	-23

[1] Modelo típico de los parámetros de potencia.

[2] Modelo máximo o de peor caso de los parámetros de potencia.

[3] Los datos del consumo del ASIC "x2" no son del todo válidos y se toman para los cálculos la medida de "x1".

[4] Los datos del consumo "x1mp" del ASIC no son totalmente válidos. Valores menores que los esperados.

También, como ya se comentó en el Capítulo anterior, la penalización en el consumo debido a los árboles de reloj no es considerada en la medida post-síntesis, mientras que en el caso post-*layout* sí se ha tenido en cuenta. En el caso del ASIC habría también que sumar la penalización de los multiplexores que habilitan la salida de cada versión Trivium y de toda la circuitería adicional incluida en el chip además de los *pads* de entrada y salida.

Los datos obtenidos tras la síntesis y el layout son bastante acordes entre sí, teniendo en cuenta las diferentes tensiones que se tienen en uno y otro caso para la medida (1,2 V y 1,08 V) y sus porcentajes coinciden prácticamente hasta la versión de 8 y 16 bits, donde las diferencias se hacen más apreciables. Su causa está motivada, como ya se ha estado comentando, por la penalización de los árboles de reloj.

Cuando se comparan los datos de síntesis y *layout* con los datos del ASIC se observa una mayor diferencia, debido sobre todo al método de medida experimental y al cómputo de la potencia de los árboles de relojes, de la cadena de multiplexores y de los efectos de la lógica adicional integrada en el circuito integrado.

Los porcentajes de mejora en el caso del ASIC reflejan mayores aumentos hasta la versión de 8 bits , aunque son coincidentes en que las dos versiones implementadas, MPLP y FPLP, en los casos de "*x1*" y "*x2*" bits reducen bastante (~50%) el consumo de la versión estándar de Trivium mientras que en el caso "*x8*", la mejoría es mínima y en el caso "*x16*" el diseño FPLP de bajo consumo ya no lo reduce, quedando igual en el caso MPLP.

5.5. Conclusiones

En este capítulo se presentan los resultados experimentales del test de los prototipos Trivium implementados en el circuito integrado CITIES, que se ha llevado a cabo con el equipo de test de señal mixta *Agilent 93000*. Se recibieron 10 muestras encapsuladas y a todas ellas se les ejecutó el plan de test que se había propuesto.

En las pruebas de verificación todas las muestras encapsuladas pasaron correctamente los test funcionales para las condiciones de polarización mínima, típica y máxima.

En las pruebas de caracterización, los resultados del test de máxima frecuencia indicaron que la máxima frecuencia de las muestras se obtenía siempre para las versiones Trivium estándar frente a las versiones de bajo consumo de potencia. La versión de un solo bit es la que mayor frecuencia consigue, siendo menor en las versiones de bajo consumo MPLP y FPLP.

En cuanto a las pruebas de consumo de potencia, los resultados obtenidos son coincidentes en verificar que las versiones de bajo consumo Trivium tienen siempre menor consumo que la estándar excepto para la versión de dieciséis bits.

Los casos más favorables en cuanto a la mejora del consumo de potencia se dan en las versiones de menos bits de salida, siendo los casos de uno y dos bit de los Trivium FPLP y MPLP los que mejores resultados ofrezcan en su conjunto. Las dos versiones implementadas, MPLP y FPLP, en los casos de uno y dos bits reducen bastante (45-53%) el consumo de la versión estándar de Trivium mientras que en el caso de ocho bits, la mejoría es mínima (6-9%) y en el caso de dieciséis bits el diseño FPLP de bajo consumo ya no lo reduce, quedando igual en el caso MPLP.

Este resultado confirma que las propuestas de diseño Trivium presentadas en esta Tesis mejoran el consumo de la versión estándar en los casos de uno, dos y ocho bits. Todas las medidas realizadas, tanto en simulación como experimentalmente, avalan los datos de mejora del consumo, mejora que es extraordinariamente importante en el caso de un bit, rondando el 50%.

Por otra parte, se dice habitualmente que las medidas experimentales en el

laboratorio conllevan a veces un tiempo mayor que el propio diseño del circuito. En nuestro caso debemos agradecer el haber podido disponer del equipo de test de señal mixta *Agilent 93000* que ha permitido que este dicho no se cumpliera y que ha facilitado enormemente las tareas de test.

Sin embargo realizar estas medidas ha presentado problemas significativos en dos aspectos principales. Uno, buscar un método adecuado para la medida de corriente sobre el chip que informe del consumo de uno de sus circuitos integrantes nada más. El segundo problema, insoslayable en el laboratorio, procede del consumo añadido por los arboles de reloj y por la circuitería adicional necesaria para hacer pruebas sobre tantos prototipos de Trivium. No obstante a pesar de estos inconvenientes, hemos conseguido medir consumos experimentalmente bastante razonables y en todo coherentes con las medidas por simulación.

6. CONCLUSIONES

En esta tesis doctoral se han realizado aportaciones en el campo de las implementaciones hardware de sistemas de cifrado y descifrado de datos para sistemas con pocos recursos (*lightweight cryptography*) especialmente indicado para aplicaciones en sistemas portables e inalámbricos y, en general, en aquellos donde las restricciones en el consumo sean importantes. En concreto, se han propuesto dos nuevas arquitecturas hardware para los cifradores de flujo Trivium, denominadas MPLP y FPLP, en las que se incorporan técnicas de reducción de consumo a nivel lógico. Los cifradores propuestos demuestran ser muy eficientes en el consumo de potencia sin pagar por ello costes apreciables en área o velocidad.

Se ha realizado una amplia revisión del estado del arte sobre algoritmos criptográficos para sistemas con bajos recursos (cifradores de clave privada) y, dentro de éstos, los que presentaban mejores características para su implementación hardware. Como conclusión del análisis bibliográfico se eligió al cifrador de flujo Trivium. De otro lado, el análisis de las técnicas de reducción de consumo junto a la consideración de la estructura del cifrador Trivium condujo a seleccionar la técnica de paralelización como aquella que proporciona mejores resultados en el diseño lógico en este cifrador.

Se ha diseñado un amplio conjunto de cifradores de flujo Trivium que pueden ser implementados en tecnologías tanto FPGA como en ASIC. En particular, se han diseñado cifradores Trivium con arquitecturas estándar y con las dos nuevas propuestas de bajo consumo, MPLP y FPLP, para las versiones de uno, dos, ocho y dieciséis bits a la salida. Los diseños se han realizado utilizando diferentes tecnologías de fabricación de circuitos integrados y de dispositivos programables, FPGA. Concretamente, en el caso ASIC se han empleado tecnologías submicrónicas y nanométricas vigentes (en concreto, 350 nm, 180 nm, 130 nm, y 90 nm), mientras que en FPGAs se han utilizado familias de dispositivos de *Xilinx Spartan-3E* y *Virtex-5*.

Los diseños realizados han sido simulados para obtener los resultados de cada propuesta en recursos, área, velocidad y consumo. También, con el fin de verificar experimentalmente el conjunto de Triviums, se ha implementado y fabricado un

ASIC en una tecnología de 90 nm. El circuito integrado contiene versiones de los cifradores Trivium con y sin arquitecturas de bajo consumo, todos ellos para salidas de uno, dos, ocho y dieciséis bits. Disponer de este circuito integrado nos ha permitido obtener resultados experimentales del consumo de potencia.

Resulta fundamental destacar la importancia de la metodología de diseño a la hora de la realización del circuito integrado y su verificación. Se ha seguido un detallado proceso de diseño, que hace uso de herramientas y entornos de diseño que resultan complejos pero sin la cual es difícil realizar con éxito el circuito integrado. En nuestro caso, se ha partido de la descripción de las arquitecturas en VHDL y se han utilizado herramientas de síntesis digital (*Design Visio*n de *Synopsys*) y esquemáticos (*Cadence Design Framework II)*. Para la verificación de las arquitecturas propuestas y del circuito integrado se han empleado simuladores eléctricos y lógicos (*Spectre, SpectreVerilog, NCSim*, de *Cadence* y *Modelsim* de *Mentor)*. Para la implementación física se ha usado la herramienta *Encounter Digital Implementation System* y la herramienta *Virtuoso,* dentro del entorno de *Cadence*.

Todas las muestras encapsuladas recibidas (diez en total) han sido testadas usando el equipo de test de señal mixta *Agilent 93000* y verificadas con éxito realizándose un exhaustivo plan de test, con más de 50 pruebas, donde todas las muestras pasaron correctamente los test funcionales[2]. La parte más importante del esfuerzo del test se ha centrado en medir experimentalmente el consumo de potencia de las diferentes implementaciones de Trivium, para lo cual se han propuesto varios mecanismos de medida con el *Agilent 93000* que, en buena parte, han podido ser automatizados.

Los principales resultados obtenidos sobre los cifradores Trivium, tanto en simulación como experimentalmente pueden resumirse en:

- Para los casos de uno y de dos bits de salida, las dos nuevas arquitecturas propuestas en esta Tesis (MPLP y FPLP) reducen el consumo de potencia de forma muy importante, alrededor del 50% del consumo de la correspondiente versión estándar, sin que el coste (medido tanto en recursos para FPGA, como en área para las implementaciones microelectrónicas de un bit) ni la velocidad de operación sufra un cambio significativo. Debe hacerse notar que en estos casos la importante mejora de consumo de las propuestas MPLP y FPLP se obtiene en todas las

[2] Quisiera destacar la utilización del equipo de test de señal mixta *Agilent 93000* para la caracterización y testado de los circuitos integrados en el laboratorio. La utilización de este equipo, complejo y de ámbito más industrial que de investigación, ha requerido un esfuerzo a veces mayor de lo esperado y no exento de dificultades y problemas derivados de su propia complejidad. Por el contrario, este equipo de test nos ha ayudado enormemente en cuanto que ha posibilitado la automatización y reproducibilidad de las pruebas de caracterización, acortando considerablemente el tiempo de testado. Además ha permitido aumentar la flexibilidad y versatilidad de las diferentes tareas que se han tenido que llevar a cabo.

formas de medida empleadas, a saber, en la simulación post-síntesis, en la simulación post-*layout* y en las medidas experimentales sobre los chips fabricados.

- Para los casos de ocho y de dieciséis bits de salida, las mejoras en consumo de las propuestas MPLP y FPLP son mucho menores e incluso llegan a tener un consumo semejante, al mismo tiempo que el coste de área y de recursos van empeorando. El sobrecoste es consecuencia del hardware adicional que hay que incorporar en las versiones de salida multibits, siendo mayor en el caso FPLP que en el MPLP.

Como conclusión final del trabajo realizado se puede destacar que se han realizado dos nuevas propuestas de diseño Trivium, con implementaciones en FPGA (para el caso de un bit) y en tecnologías submicrónicas/nanométricas de las que las de 90 nm han sido incorporadas en un ASIC, el cual ha sido testado experimentalmente, propuestas que, en los casos de uno y de dos bits a la salida, mejoran sustancialmente el consumo de potencia de la versión estándar, llegando a reducir el consumo a la mitad. Esto permitirá desarrollar aplicaciones de seguridad en entornos de muy pocos recursos y baja energía disponible.

Algunos de los resultados de la investigación expuestos en esta memoria de Tesis han generado varios artículos, publicados o en vías de publicación, los cuales enumeramos a continuación:

- Mora-Gutiérrez, J.M., Jiménez-Fernández, C.J., Valencia-Barrero, M. Low power implementation of Trivium stream cipher (2013) *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7606 LNCS,pp.113-120. DOI: 10.1007/978-3-642-36157-9_12.

- J. M. Mora, C. J. Jiménez, E. Potestad, M. Valencia, "Low power implementation of Trivium stream cipher", 17th Workshop on Cryptographic Hardware and Embedded Systems (CHES'15), 13-16 Sept. 2015. (Poster con proceso de selección directa, a presentar en Saint Malo-Francia).

- Mora-Gutiérrez, J. M., C. J. Jiménez-Fernández, and M. Valencia-Barrero. "Trivium hardware implementations for power reduction". International Journal of Circuit Theory and Applications (2016). DOI: 10.1002/cta.2281.

- Mora-Gutiérrez, J. M., C. J. Jiménez-Fernández, and M. Valencia-Barrero. "Multi-radix Trivium Hardware Implementations for low-power IoT hardware". IEEE Transactions on VLSI Systems (2016). *Enviado y pendiente de aceptación*.

REFERENCIAS

[1] Centro Criptológico Nacional, «Guía/Norma de seguridad de las TIC (CCN-STIC-401). Criptología de empleo en el esquem nacional de seguridad.».

[2] «European Comission. Internet of Things in a Roadmap for the future,2008. p. 1-32».

[3] A. K. L. Isha, «Analysis of Lightweight Cryptographic Solutions for Internet of Things,» *Indian Journal of Science and Technology*, vol. 9, nº 28, 2016. DOI: 10.17485/ijst/2016/v9i28/98382.

[4] C. Manifavas, G. Hatzivasilis, K. Fysarakis and Y. Papaefstathiou, "Lightweight Cryptography for Embedded Systems - A Comparative Analysis," *Security and Communication Networks*, vol. 9, no. 10, p. 1226–1246, 2015. DOI 10.1007/978-3-642-04101-3.

[5] B. Preneel, C. Paar y J. Pelzl, Understanding cryptography: a textbook for students and practitioners, 2009. DOI: 10.1007/978-3-642-04101-3.

[6] Pierce, D. G. y P. G. Brusius., «Electromigration: A review.,» *Microelectronics Reliability*, vol. 37, nº 7, pp. 1053-1072, 1997. DOI: 10.1016/S0026-2714(96)00268-5.

[7] Yi-Kan Cheng, Ching-Han Tsai, Chin_Chi Teng y Sung-, Electrothermal Analysis of VLSI Systems, Springer US, 2002.DOI:10.1007/b117332.

[8] H. Bhatnagar, Advanced ASIC Chip Synthesis: Using Synopsys Design CompilerTM Physical CompilerTM and PrimeTime, Springer Science & Business Media., 2007.DOI 10.1007/b117024.

[9] «Design Vision User Guide, and G. Version. "2012. Synopsys."».

[10] Olloz, S., Villar, E., Torroja, Y. y Teres, L., Vhdl lenguaje estándar de diseño electrónico, McGraw Hill, 1998.

[11] F. Pardo Carpio, Vhdl. Lenguaje Para Síntesis Y Modelado De Circuitos, Editorial Ra-ma, 2003.

[12] M. Graphics, «ModelSim,» 2016. [En línea]. Available: https://www.mentor.com/products/fpga/model/.

[13] «Cadence NC-Verilog Simulator Tutorial. Product Version 5.1,» 2003.

[14] «Cadence,» [En línea]. Available: https://www.cadence.com/content/cadence-www/global/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-circuit-simulator.html. [Último acceso: 2016].

[15] Cadence, « Cadence Design Systems. Soc Encounter RTL-to-GDSIICadence,» 2016. [En línea]. Available: https://www.cadence.com/.

[16] «Virtuoso, X. L. Layout Editor User Guide, Product Version 5.0. Cadence Design Systems Incorporated.,» 2003.

[17] «Menthor Graphics. Calibre verification user's manual.,» 2008.

[18] A. Fuster Sabater, Criptografía, protección de datos y aplicaciones: una guía para estudiantes y profesionales., Ra-Ma, 2012.

[19] L. Hernández Encinas, La Criptografía, CSIC, 2016.

[20] K. A. McKay, K. A., L. Bassham, L., M. S. Turan y Mouha, «Report on Lightweight Cryptography. NIST DRAFT NISTIR, 8114,» 2016.

[21] M. Katagi and M. Shiho , "Lightweight cryptography for the internet of things," *Sony Corporation,* pp. 7-10, 2008.

[22] . T. Eisenbarth, . S. Kumar, C. Paar, A. Poschmann y L. Uhsadel, «A survey of lightweight-cryptography implementations. IEEE Design & Test of Computers,» *IEEE Design & Test of Computers,* vol. 24, nº 6, pp. 522-533, 2007.

[23] Leander, G., Paar, C., Poschmann, A. y Schramm, «New Lightweight DES Variants.,» *Proc. 14th International Workshop on Fast Software Encryption (FSE 2007),* pp. 196-210, 2007. DOI: 10.1007/978-3-540-74619-5_13.

[24] T. Shirai, K. Shibutani, . T. Akishita, S. Moriai y T. Iwata, «The 128-bit blockcipher CLEFIA.,» *International Workshop on Fast Software Encryption,* pp. 181-195, 2007. DOI: 10.1007/978-3-540-74619-5_12.

[25] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin y C. Vikkelsoe, «PRESENT: An ultra-lightweight block cipher,» *International Workshop on Cryptographic Hardware and Embedded Systems,* pp. 450-466, 2007. DOI: 10.1007/978-3-540-74735-2_31.

[26] Beaulieu, R., Shors, D., D., Smith y Treatman-Clark, «The SIMON and SPECK Families of Lightweight Block Ciphers,» *IACR Cryptology ePrint Archive,* 2013.

[27] «ECRYPT II,» 2009. [En línea]. Available: http://www.ecrypt.eu.org/stream.

[28] A. Klein, Stream ciphers, Springer, 2013. DOI: 10.1007/978-1-4471-5079-4.

[29] M. Robshaw y O. Billet, New stream cipher designs, Springer, 2008. DOI: 10.1007/978-3-540-68351-3.

[30] M. Hell, T. Johanson y W. Meier, «Grain: a stream cipher for constrained environments.,» *International Journal of Wireless and Mobile Computing,* vol. 2, nº 1, pp. 86-93, 2007.

[31] C. De Cannière, «Trivium: A stream cipher construction inspired by block cipher design principles,» de *International Conference on Information Security*, 2006. DOI:10.1007/11836810_13.

[32] D. Shanmugam y S. Annadurai, «Secure Implementation of Stream Cipher: Trivium,» de *International Conference for Information Technology and Communications*, 2015. DOI: 10.1007/978-3-319-27179-8_18 .

[33] R. E. Atani, S. Mirzakuchaki, S. Atani y W. Meier, «On DPA-resistive implementation of FSR-based stream ciphers using SABL logic styles,» *International Journal of Computers Communications & Control,* vol. 3, nº 4, pp. 324-335, 2008.

[34] T. Good y M. Benaissa, «Hardware performance of eStream phase-III stream cipher candidates,» de *Proc. of Workshop on the State of the Art of Stream Ciphers (SACS'08)*, 2008.

[35] T. Good y M. Benaissa, «Hardware results for selected stream cipher candidates,» de *Good, T., and M. Benaissa. "Hardware Result of Stream Ciphers 2007 (SASC 2007), Workshop*, 2007.

[36] T. Good, W. Chelton y M. Benaissa, «Review of stream cipher candidates from a low resource hardware perspective,» eSTREAM, ECRYPT Stream Cipher Project, Report 2006/016, 2006 .

[37] F. K. Gurkaynak, P. Luethi y N. Bernold, «Hardware Evaluation of eSTREAM Candidates: Achterbahn, Grain, MICKEY, MOSQUITO, SFINKS, Trivium, VEST, ZK-Crypt,» ECRYPT Stream Cipher Project, 2006.

[38] E. Tena y A. J. Acosta, «DPA vulnerability analysis on Trivium stream cipher using an optimized power model,» *Proc. of International Symposium on Circuits and Systems (ISCAS´15),* 2015. DOI:10.1109/ISCAS.2015.7169016.

[39] M. Feldhofer, «Comparison of low-power implementations of Trivium and Grain." Workshop on The State of the Art of Stream Ciphers (SASC2007) pages. 2007.,» de *Workshop on The State of the Art of Stream Ciphers (SASC2007)* .

[40] «Xilinx,» 2016. [En línea]. Available: http://www.xilinx.com/support.html#documentation.

[41] M. Rogawski, «Hardware evaluation of estream candidates: grain, lex, mickey128, salsa20 and trivium." State of the Art of Stream Ciphers Workshop (SASC 2007), eSTREAM, ECRYPT Stream Cipher Project, Report. Vol. 25. 2007.»

[42] J. Marmolejo-Tejada, V. Trujillo-Olaya y J. Velasco-Medina, «Hardware implementation of Grain-128, Mickey-128, Decim-128 and Trivium,» *ANDESCON, 2010 IEEE,* pp. 1- 6, 2010.

[43] «Altera,» 2016. [En línea]. Available: https://www.altera.com/.

[44] K. Gaj, G. Southern y R. Bachimanchi, «Comparison of hardware performance of selected Phase II eSTREAM candidates,» de *State of the Art of Stream Ciphers Workshop (SASC 2007)*, 2007.

[45] P. Bulens, K. Kalach, F. X. Standaert y . J. J. Quisquater, «FPGA implementations of eSTREAM phase-2 focus candidates with hardware profile,» de *Proceedings of SASC*, 2007.

[46] D. Hwang, M. Chaney, S. Karanam, N. Ton y K. Gaj, «Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates,» de *The State of the Art of Stream Ciphers (2008)*, 2008.

[47] H. Hassan y M. Anis, Low-power design of nanometer FPGAs: architecture and EDA, Morgan Kaufmann, 2010.

[48] P. R. Panda, B. V. N. Silpa, A. Shrivastava y K. Gummidipudi, Power-efficient system design, Springer, 2010. DOI: 978-1-4419-6387-1.

[49] G. K. Yeap, Practical low power digital VLSI design, Springer, 1998. DOI: 10.1007/978-1-4615-6065-4.

[50] R. Chadha y J. Bhasker, An ASIC Low Power Primer; Analysis, Techniques and Specification, Springer, 2013. DOI:10.1007/978-1-4614-4271-4.

[51] A. P. Chandrakasan y R. W. Broderesen, Low-Power digital CMOS design, Boston: Kluwer Academics Publishers, 1995. DOI:10.1007/978-1-4615-2325-3.

[52] M. Pedran y J. M. Rabaey, Power Aware Design Methodologies, Springer, 2002. DOI: 978-1-4020-7152-2.

[53] J. M. Rabey, A. Chandrakasan y N. Borivoje, Digital Integrated Circuits, Prentice Hall, 2003.

[54] M. Keating, D. Flynn, R. Aitken, A. Gibbons y K. Shi., Low power methodology manual: for system-on-chip design, Springer, 2007. DOI:10.1007/978-0-387-71819-4.

[55] Piguet, Christian, Low Power CMOS Circuits: : Technology, Logic Design and CAD Tools, CRC Press, 2005. DOI: 10.1201/9781420036503.

[56] S. G. Narendra y A. P. Chandrakasan, Leakage in Nanometer CMOS Technologies, Springer, 2006. DOI: 978-0-387-25737-2.

[57] D. Chinnery y K. Keutzer, Closing the power gap between ASIC & custom: tools and techniques for low power design, Springer, 2008. DOI: 10.1007/978-0-387-68953-1.

[58] W. Nebel y J. Mermet, Low power design in deep submicron electronics, Springer, 1997. DOI: 10.1007/978-1-4615-5685-5.

[59] H. Veendrick, Nanometer CMOS ICs:From basics to ASICs, Springer, 2008.

[60] . F. N. Najm, «Transition density: A new measure of activity in digital circuits,» *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*

12.2, vol. 12, nº 2, pp. 310-323, 1993.

[61] A. Ghosh, S. Devadas, K. Keutzer y J. White, «Estimation of average switching activity in combinational and sequential circuits,» *Proceedings of the 29th Design Automation Conference,* p. 253–259, 1992.

[62] A. P. Chandrakasan, S. Sheng y R. W. Brodersen , «Low-power Digital CMOS Design,» *IEEE Journal of Solid State Circuits,* pp. 473-484, 1992.

[63] L. Benini, P. Siegel y . G. DeMicheli, «Saving power by synthesizing gated clocks for sequential circuits,» *IEEE Design & Test of Computers,* pp. 32-41, 1994. DOI: 10.1109/54.329451.

[64] G. Pouiklis y G. C. Sirakoulis, «Clock gating methodologies and tools: a survey,» *International Journal of Circuit Theory and Applications,* 2015.

[65] B. Vasantha Kumar, N. Murthy Sharm, . K. L. Kishore y A. Rajakumari, «Selective Glitch Reduction Technique for Minimizing Peak Dynamic IR Drop,» *Microelectronics and Solid State Electronics,* pp. 27-32, 2013. DOI: 10.5923/s.msse.201302.04.

[66] . A. P. Chandrakasan y R. W. Brodersen, «Minimizing Power Consumption in digital CMOS Circuits,» *Proceedings of the IEEE,* vol. 83, nº 4, pp. 498-523, 1995. DOI: 10.1109/5.371964.

[67] K. Nose y T. Sakurai, «Analysis and future trend of short-circuit power,» *IEEE Transactions on Computer-Aided Design of integrated circuits and systems,* vol. 19, nº 9, pp. 1023-1030, 2000.

[68] H. J. Veendrick, «Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits,» *IEEE Journal of Solid State Circuits,* vol. 19, nº 4, pp. 468-473, 1984. DOI: 10.1109/JSSC.1984.1052168.

[69] M. Horowitz, T. Indermaur y R. Gonzalez, «Low-Power Digital Design,» *Proceedings of the IEEE Simposium on Low-Power Electronics,* pp. 8-11, 1994.

[70] F. Machado Sánchez, *Estudio de la actividad de conmutación de circuitos electrónicos digitales descritos en el nivel de transferencia de registros mediante técnicas probabilísticas. Propuesta de un método de estimación.,* Tesis (Doctoral), E.T.S.I. Industriales (UPM) , 2008.

[71] A. Rubio y et al., Diseño de Circuitos y Sistemas Integrados, Barcelona: Ediciones UPC, 2003.

[72] R. Razmdideh y M. Saneei, «Two novel low power and very high speed pulse triggered flip-flops. International Journal Of Circuit Theory And Applications,» *International Journal Of Circuit Theory And Applications,* vol. 43, pp. 1925-1934, 2015. DOI: 10.1002/cta.2048.

[73] «Liberty User Guides and Reference Manual,» Version 2013.03.

[74] M. Kocheta, N. Sujatha, K. Sivakanya, R. Srikanth, S. Shetty y P. Ananda

Mohan, «A review of some recent stream ciphers,» *International conference on Circuits, Controls and Communications (CCUBE),* pp. 1- 6, 2013.

[75] J. Gong, G. Chen, L. Li y J. Li., «A secure authentication protocol for RFID based on Trivium,» *International Conference on Computer Science and Service System (CSSS),* pp. 107- 109, 2011.

[76] B. Lin y H. de Man, «Low-power driven technology mapping under timing constraints,» *Proc. ICCAD,* pp. 421-427, 1993.

[77] P. Parra-Fernandez, Ruido de conmutación en circuitos integrados digitales CMOS., Sevilla: Tesis Doctoral, 2010.

[78] Xilinx, «XPower Estimator User Guide. Xilinx power tools tutorial (2010). 2012.».

[79] «CriptoBio (Diseño Microelectrónica para Autenticación Cripto-Biomédica, TEC2008-3674), de la Junta de Andalucía, 2009-2012.».

[80] «MOBY-DIC (Model-based Synthesis of Digital Electronic Circuits for Embedded Control, Ec-IST-VIIPM Nº-248858) de la Comisión Europea, 2009-2012.».

[81] «Europractice,» [En línea]. Available: http://www.europractice-ic.com/prototyping_packaging.php. [Último acceso: 2016].

[82] A. Acosta, A. Barriga, M. Bellido y M. Valencia, , Temporización en circuitos integrados CMOS, Editorial Marcombo, 2000.

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

NOTACIÓN

AES	Advanced Encryption Standard.
ALUT	Adaptive Look Up Table.
AIT	ASCII interface timing.
AIV	ASCII interface vectors.
AMS	Austria Micro Systems.
ASCII	American Standard Code for Information Interchange.
ASIC	Application-Specific Integrated Circuit.
ATE	Automated Test Equipment.
AWG	Arbitrary waveform generators.
BC	Best Case.
Block ciphers	Cifradores de bloque.
Bonding	Conexionado entre el encapsulado y el circuito integrado.
CAD	Computer-aided design.
CESAR	Circuitos microlectrónicos seguros frente a ataques laterales.
CITIES	Circuitos integrados para transmisión de información especialmente segura.
CLBS	Configurable logic block.
CMOS	Complementary Metal-Oxide-Semiconductor.
CNM	Centro Nacional de Microelectrónica.
Corner	Esquina de simulación.
CQF	Ceramic Quad Flat Package.
CSIC	Consejo Superior de Investigaciones Científicas.
DFWII	Design Framework II.
DIB	Device interface board.
DRC	Design Rule Checking.
DPA	Differential Power Analysis.
DPS	Device Power Supply.
DUT	Device Under Test.
ECRYPT	European Network of Excellence in Cryptology.
EDI	Encounter Digital Implementation.

ERC	Electrical Rule Checking.
Flip-flop	Biestable.
FPGA	Field-Programmable Gate Array.
FPLP	Full parallel low power.
GDSII	Geometrical Data Base Standard for Information Interchange.
HDL	Hardware Description Language.
IMSE	Instituto de Microelectrónica de Sevilla.
IoT	Internet of Things.
IP	Intellectual Property.
IV	Vector de Inicialización.
Key	Clave.
Hold time	Tiempo de mantenimiento.
HW	Hardware.
LE	Logic Element.
Leakage	Corriente de fugas.
LFSR	Linear Feedback Shift Register.
LUTs	Look-Up Tables.
LWC	Lightweight Cryptography.
LVS	Layout Versus Schematic.
MB	Megabyte.
MOS	Metal-Oxide-Semiconductor.
MPLP	Mixed Parallel Low Power.
Msps	Mega samples per second.
NIST	National Institute of Standards and Technology.
RFID	Radio Frequency IDentification.
RTL	Register Transfer Language.
SABL	Sense Amplifier Base Logic.
SAIF	Switching Activity Interchange Format.
SDF	Standard Delay Format.
SDC	Standard Delay Constraints.
SMB	Surface Mounting Device.
Setup time	Tiempo de establecimiento.
SOC	System on Chip.
Stream ciphers	Cifradores de flujo.

SW	Software.
TC	Typical Case.
TIA	Time Interval Analyzer.
TIC	Tecnologías de la Información y la Comunicación.
Trigger	Señal de disparo.
TSMC	Taiwan Semiconductor Manufacturing Company Limited.
US	Universidad de Sevilla.
VCD	Value Charge Dump.
VHDL	VHSIC Hardware Description Language.
VLSI	Very Large Scale Integration.
WC	Worst Case.
WDG	Waveform digitizers.
XPA	Xilinx Power Analyzer.

APÉNDICE A. CÓDIGOS *VHDL*

A continuación se presentan los códigos *VHDL* correspondientes a las propuestas con y sin arquitecturas de bajo consumo de los cifradores de flujo Trivium correspondientes a salidas de 1 bit, 2 bits, 8 bits y 16 bits.

A.1. Trivium

```
----------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_CORE - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium 1 bit.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
-- /*checkout*/ecrypt/trunk/submissions/trivium/
-- /unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la
-- carga de key/IV.
-- Incluye función de conversión de los vectores de test
-- para compararlos con la salida
--
----------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_core is
    Port ( sys_clk : in  STD_LOGIC; --Reloj principal
           resetn :in  STD_LOGIC;
-- Control de la carga
           cntrl : in STD_LOGIC_VECTOR(1 downto 0);
-- Clave secreta
           key  : in  STD_LOGIC_VECTOR(79 downto 0);
```

```
-- Vector Inicialización
          iv    : in  STD_LOGIC_VECTOR(79 downto 0);
-- Secuencia cifrante.
           key_out : out  STD_LOGIC);
end trivium_core;
architecture Behavioral of trivium_core is
--Registro interno
signal state : STD_LOGIC_VECTOR(287 downto 0) := (OTHERS => '0');
--Nodos realimentación.
signal k1,k2,k3,t1,t2,t3,key_stream : STD_LOGIC := '0';
signal key_flip : STD_LOGIC_VECTOR(79 downto 0);
signal iv_flip : STD_LOGIC_VECTOR(79 downto 0);
-- Conversor 80-bit Big Endian a Little Endian
-- (Revertir cada bit del byte)
function little_endian (b: std_logic_vector) return
std_logic_vector is
     variable result : std_logic_vector(79 downto 0);
     --ej 0x0123456789 -> 0x084C2A6E19
   begin
       for i in 0 to 9 loop
          result(((i*8)+7) downto (i*8)) := b(i*8) &
                                            b((i*8) + 1) &
                                            b((i*8) + 2) &
                                            b((i*8) + 3) &
                                            b((i*8) + 4) &
                                            b((i*8) + 5) &
                                            b((i*8) + 6) &
                                            b((i*8) + 7);
       end loop;
       return result;
   end;
begin
-- 3 registros de desplazamiento LFSR con realimentación
-- de los nodos(t1,t2,t3).
MAIN_TRIVIUM : PROCESS (sys_clk,cntrl,resetn)
begin
    if(resetn = '0') then
         state(287 downto 0) <= (OTHERS => '0');
    elsif(sys_clk'event AND sys_clK='1') then
        if (cntrl = "10") then   --Carga estado inical.
            state(92 downto 0) <= "0000000000000" & key_flip;
            state(176 downto 93) <= X"0" & iv_flip;
            state(287 downto 177) <= "111" &
X"00000000000000000000000000";
        elsif ((cntrl = "11") OR (cntrl = "01")) then
            --Registro de desplazamiento.
            state(92 downto 0) <= state(91 downto 0) & t3;
            state(176 downto 93) <= state(175 downto 93) & t1;
            state(287 downto 177) <= state(286 downto 177) & t2;
        end if;
    end if;
end process;
--XOR Nodos
k1 <= state(65) XOR state(92);
k2 <= state(161) XOR state(176);
```

```
k3 <= state(242) XOR state(287);
t1 <= k1 XOR ((state(90) AND state(91)) XOR state(170));
t2 <= k2 XOR ((state(174) AND state(175)) XOR state(263));
t3 <= k3 XOR ((state(285) AND state(286)) XOR state(68));
key_stream <= k1 XOR k2 XOR k3;
 --Cambia valores de entrada a "little endian" para que las
 --salidas coincidan con los valores esperados.
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
key_out <= key_stream;
end Behavioral;
```

A.2. Trivium MPLP

```
----------------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREMPLP - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium 1 bit MPLP.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test para
-- compararlos con la salida
--
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_coremplp is
    Port ( sys_clk : in  STD_LOGIC; -- Reloj principal
           resetn : in  STD_LOGIC;
--Control de la carga
           cntrl : in STD_LOGIC_VECTOR(1 downto 0);
--Clave secreta
              key  : in  STD_LOGIC_VECTOR(79 downto 0);
--Vector Inicialización
              iv   : in  STD_LOGIC_VECTOR(79 downto 0);
-- Secuencia cifrante
            key_out : out  STD_LOGIC);
end trivium_coremplp;
```

```
architecture Behavioral of trivium_coremplp is
signal key_flip : STD_LOGIC_VECTOR(79 downto 0);
signal iv_flip : STD_LOGIC_VECTOR(79 downto 0);
signal state1 : STD_LOGIC_VECTOR(92 downto 64) := (OTHERS =>
'0');
signal state2 : STD_LOGIC_VECTOR(176 downto 161) := (OTHERS =>
'0');
signal state3 : STD_LOGIC_VECTOR(287 downto 241) := (OTHERS =>
'0');
signal state : STD_LOGIC_VECTOR(287 downto 0) := (OTHERS => '0');
signal statepar1 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS =>
'0');
signal stateimpar1 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS =>
'0');
signal statepar2 : STD_LOGIC_VECTOR(33 downto 0) := (OTHERS =>
'0');
signal stateimpar2 : STD_LOGIC_VECTOR(33 downto 0) := (OTHERS =>
'0');
signal statepar3 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS =>
'0');
signal stateimpar3 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS =>
'0');
--- XOR Nodos realimentación
signal k1n,k2n,k3n,t1n,t2n,t3n,key_stream : STD_LOGIC := '0';
signal s1,s2,s3: STD_LOGIC := '0';

signal qout,qout2 : STD_LOGIC :='0';   --- Reloj/2
signal sys_clk3  : STD_LOGIC;   --- Reloj/2
-- Conversor 80-bit Big Endian a Little Endian
-- (Revertir cada bit del byte)
function little_endian (b: std_logic_vector) return
std_logic_vector is
       variable result : std_logic_vector(79 downto 0);
--ex 0x0123456789 -> 0x084C2A6E19
   begin
       for i in 0 to 9 loop
          result(((i*8)+7) downto (i*8)) := b(i*8) &
                                            b((i*8) + 1) &
                                            b((i*8) + 2) &
                                            b((i*8) + 3) &
                                            b((i*8) + 4) &
                                            b((i*8) + 5) &
                                            b((i*8) + 6) &
                                            b((i*8) + 7);
       end loop;
       return result;
   end;

begin
-- Definición procesos de reloj
    SYS_CLK3_process :process  (sys_clk,resetn)
    begin
       IF(resetn = '0') then
          qout2  <= '0';
       ELSIF(sys_clk'event AND sys_clk='1') then
```

```
          qout2 <= NOT(qout2) ;
        END IF;
     end process;
     sys_clk3 <= NOT(qout2);
state(92 downto 0) <= "0000000000000" & key_flip;
state(176 downto 93) <= X"0" & iv_flip;
state(287 downto 177) <= "111" & X"00000000000000000000000000";
SECOND_TRIVIUM_RISE : process (sys_clk3,resetn)
--3 registros de desplazamiento LFSR con realimentación de los
-- nodos
begin
       if(resetn = '0') then
            statepar1  <= (OTHERS => '0');
            statepar2  <= (OTHERS => '0');
            statepar3  <= (OTHERS => '0');
       elsif(sys_clk3'event AND sys_clk3='0') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
         if (cntrl = "10") then
          -- FIRST PAR
           for i in 0 to 31 loop -- hasta 62
               statepar1 (i) <= state (2*i);
           end loop;
           for i in 0 to 33 loop -- hasta 159
               statepar2 (i) <= state (92+1+(2*i));
           end loop;
        -- THIRD
           for i in 0 to 31 loop -- hasta 239
              statepar3 (i) <= state (176+1+(2*i));
           end loop;
        elsif ((cntrl = "11") OR (cntrl = "01")) then
          statepar1(31 downto 0) <= statepar1(30 downto 0) & t3n;
          statepar2(33 downto 0) <= statepar2(32 downto 0) & t1n;
          statepar3(31 downto 0) <= statepar3(30 downto 0) & t2n;
         end if;
      end if;
end process;
SECOND_TRIVIUM_FALL : process (sys_clk3,resetn)
--3 registros de desplazamiento LFSR con realimentación de los
-- nodos
begin
--
   if(resetn = '0') then
     stateimpar1  <= (OTHERS => '0');
     stateimpar2  <= (OTHERS => '0');
     stateimpar3  <= (OTHERS => '0');
   elsif(sys_clk3'event AND sys_clk3='1') then
      if (cntrl = "10") then
   -- FIRST
        for i in 0 to 31 loop -- hasta 63
           stateimpar1 (i) <= state (1+(2*i));
        end loop;
   -- SECOND
        for i in 0 to 33 loop -- hasta 160
           stateimpar2 (i) <= state (94+(2*i));
        end loop;
```

```
   -- THIRD PAR
        for i in 0 to 31 loop -- hasta 240
           stateimpar3 (i) <= state (178+(2*i));
        end loop;
   elsif ((cntrl = "11") OR (cntrl = "01")) then
-- registro 32 bits
      stateimpar1(31 downto 0) <= stateimpar1(30 downto 0) & t3n;
-- registro 34 bits
      stateimpar2(33 downto 0) <= stateimpar2(32 downto 0) & t1n;
-- registro 32 bits
      stateimpar3(31 downto 0) <= stateimpar3(30 downto 0) & t2n;
    end if;
  end if;
end process;
--3 registros de desplazamiento LFSR con realimentación
MAIN_TRIVIUM : process (sys_clk,cntrl,resetn)
begin           --Core must be loaded with key and
   if(resetn = '0') then
      state1  <= (OTHERS => '0');
      state2  <= (OTHERS => '0');
     state3  <= (OTHERS => '0');
   elsif(sys_clk'event AND sys_clk='1') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
      if (cntrl = "10") then     ---
        state1 <= state(92 downto 64) ;    -- registro 93 bits
        state2 <= state(176 downto 161) ;  -- registro 84 bits
        state3 <= state(287 downto 241) ; -- registro 111 bits
      elsif ((cntrl = "11") OR (cntrl = "01")) then
        state1(92 downto 64) <= state1(91 downto 64) & s1;
        state2(176 downto 161) <= state2(175 downto 161) & s2;
        state3(287 downto 241) <= state3(286 downto 241) & s3;
      end if;
   end if;
end process;
-- Cambiar valores de entrada a "little endian" para que la
-- salida verifique los vectores de test.
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
-- MODIFICACION
--Nodo salida shift reg 1
s1 <= stateimpar1(31) when sys_clk3 ='0' else statepar1(31);
--Nodo salida shift reg 2
s2 <= stateimpar2(33) when sys_clk3 ='0' else statepar2(33);
--Nodo salida shift reg 3
s3 <= stateimpar3(31) when sys_clk3 ='0' else statepar3(31);
--XOR Nodes
k1n <= state1(65) XOR state1(92);
k2n <= state2(161) XOR state2(176);
k3n <= state3(242) XOR state3(287);
t1n <= k1n XOR ((state1(90) AND state1(91)) XOR state2(170));
t2n <= k2n XOR ((state2(174) AND state2(175)) XOR state3(263));
t3n <= k3n XOR ((state3(285) AND state3(286)) XOR state1(68));
key_stream <= k1n XOR k2n XOR k3n;
key_out <= key_stream;
end Behavioral;
```

A.3. Trivium FPLP

```
----------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREFPLP - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium 1 bit FPLP.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test para
-- compararlos con la salida
----------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_corefplp is
    Port ( sys_clk : in  STD_LOGIC; --Reloj principal
           resetn :in  STD_LOGIC;
--Control de la carga
           cntrl : in STD_LOGIC_VECTOR(1 downto 0);
--Clave secreta
           key  : in  STD_LOGIC_VECTOR(79 downto 0);
--Vector Inicialización
           iv   : in  STD_LOGIC_VECTOR(79 downto 0);
-- Secuencia cifrante
           key_out : out  STD_LOGIC);
end trivium_corefplp;
architecture Behavioral of trivium_corefplp is
SIGNAL key_flip : STD_LOGIC_VECTOR(79 downto 0);
SIGNAL iv_flip : STD_LOGIC_VECTOR(79 downto 0);
SIGNAL state1 : STD_LOGIC_VECTOR(4 downto 0) := (OTHERS => '0');
SIGNAL state2 : STD_LOGIC_VECTOR(4 downto 0) := (OTHERS => '0');
SIGNAL state3 : STD_LOGIC_VECTOR(4 downto 0) := (OTHERS => '0');
SIGNAL state : STD_LOGIC_VECTOR(287 downto 0) := (OTHERS => '0');
SIGNAL statepar1 : STD_LOGIC_VECTOR(46 downto 0) := (OTHERS =>
'0');
SIGNAL stateimpar1 : STD_LOGIC_VECTOR(46 downto 0) := (OTHERS =>
'0');
SIGNAL statepar2 : STD_LOGIC_VECTOR(41 downto 0) := (OTHERS =>
'0');
```

```
SIGNAL stateimpar2 : STD_LOGIC_VECTOR(41 downto 0) := (OTHERS =>
'0');
SIGNAL statepar3 : STD_LOGIC_VECTOR(55 downto 0) := (OTHERS =>
'0');
SIGNAL stateimpar3 : STD_LOGIC_VECTOR(55 downto 0) := (OTHERS =>
'0');  -
SIGNAL k1n,k2n,k3n,t1n,t2n,t3n,key_stream_n : STD_LOGIC := '0';
---XOR
SIGNAL qout,qout2 : STD_LOGIC :='0';   ---Reloj/2
SIGNAL sys_clk3  : STD_LOGIC;   ---Reloj/2
-- Conversor 80-bit Big Endian a Little Endian (Revertir cada bit
-- del byte)
function little_endian (b: std_logic_vector) return
std_logic_vector is
       variable result : std_logic_vector(79 downto 0);
--ex 0x0123456789 -> 0x084C2A6E19
   begin
       for i in 0 to 9 loop
          result(((i*8)+7) downto (i*8)) := b(i*8) &
                                            b((i*8) + 1) &
                                            b((i*8) + 2) &
                                              b((i*8) + 3) &
                                              b((i*8) + 4) &
                                              b((i*8) + 5) &
                                              b((i*8) + 6) &
                                              b((i*8) + 7);
       end loop;
       return result;
   end;
begin
-- Definición procesos de reloj
   SYS_CLK3_process :process  (sys_clk,resetn)
   begin
        if(resetn = '0') then
           qout2  <= '0';
       elsif(sys_clk'event AND sys_clk='1') then
             qout2 <= NOT(qout2) ;
        end if;
   end process;
sys_clk3 <= NOT(qout2);
state(92 downto 0) <= "0000000000000" & key_flip;
state(176 downto 93) <= X"0" & iv_flip;
state(287 downto 177) <= "111" & X"00000000000000000000000000";
SECOND_TRIVIUM_RISE : process (sys_clk3,cntrl,resetn)
--3 registros de desplazamiento LFSR con realimentación de los
-- nodos
begin
   if(resetn = '0') then
       statepar1  <= (OTHERS => '0');
       statepar2  <= (OTHERS => '0');
       statepar3  <= (OTHERS => '0');
   elsif(sys_clk3'event AND sys_clk3='0') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
       if (cntrl = "10") then  ---Estado inicial
       -- FIRST PAR
```

```
          for i in 0 to 46 loop
              statepar1 (i) <= state (2*i);
          end loop;
          for i in 0 to 41 loop
              statepar2 (i) <= state (92+1+(2*i));
          end loop;
       -- THIRD
          for i in 0 to 55 loop
              statepar3 (i) <= state (176+1+(2*i));
          end loop;
       elsif ((cntrl = "11") OR (cntrl = "01")) then
-- registro 47 bits
          statepar1(46 downto 0) <= statepar1(45 downto 0) & t3n;
-- registro 42 bits
          statepar2(41 downto 0) <= statepar2(40 downto 0) & t1n;
-- registro 55 bits + 1 bits mas
          statepar3(55 downto 0) <= statepar3(54 downto 0) & t2n;
      end if;
   end if;
end process;
SECOND_TRIVIUM_FALL : process (sys_clk3,cntrl,resetn)
--3 registros de desplazamiento LFSR con realimentación de los
-- nodos (t1,t2,t3).
begin
  if(resetn = '0') then
     stateimpar1  <= (OTHERS => '0');
     stateimpar2  <= (OTHERS => '0');
     stateimpar3  <= (OTHERS => '0');
  elsif(sys_clk3'event AND sys_clk3='1') then
    if (cntrl = "10") then   ---Configuración estado inicial
-- FIRST
      for i in 0 to 45 loop
        stateimpar1 (i) <= state (1+(2*i));
      end loop;
-- SECOND
      for i in 0 to 41 loop
         stateimpar2 (i) <= state (94+(2*i));
      end loop;
-- THIRD PAR
      for i in 0 to 54 loop
       stateimpar3 (i) <= state (178+(2*i));
      end loop;
    elsif ((cntrl = "11") OR (cntrl = "01")) then
--registro 47 bits
      stateimpar1(46 downto 0) <= stateimpar1(45 downto 0) &
t3n;
      stateimpar2(41 downto 0) <= stateimpar2(40 downto 0) &
t1n;
-- --registro 56 bits
           stateimpar3(55 downto 0) <= stateimpar3(54 downto 0)
& t2n; --
    end if;
  end if;
end process;
-- Cambiar valores de entrada a “little endian” para que la
```

```
-- salida verifique
-- los vectores de test.
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
-- MODIFICACION
--XOR 65 IMPAR
state1(0) <= stateimpar1(32) when sys_clk3 ='0' else
statepar1(32);
--XOR 92 PAR
state1(1) <= statepar1(46) when sys_clk3 ='0' else
stateimpar1(46);
--XOR 90 PAR
state1(2) <= statepar1(45) when sys_clk3 ='0' else
stateimpar1(45);
--XOR 91 IMPAR
state1(3) <= stateimpar1(45) when sys_clk3 ='0' else
statepar1(45);
--XOR 170 IMPAR
state2(4) <= stateimpar2(38) when sys_clk3 ='0' else
statepar2(38);
--XOR 161 PAR
k1n <= state1(0) XOR state1(1);
state2(0) <= statepar2(34) when sys_clk3 ='0' else
stateimpar2(34);
--XOR 176 IMPAR
state2(1) <= stateimpar2(41) when sys_clk3 ='0' else
statepar2(41);
--XOR 174 IMPAR
state2(2) <= stateimpar2(40) when sys_clk3 ='0' else
statepar2(40);
--XOR 175 PAR
state2(3) <= statepar2(41) when sys_clk3 ='0' else
stateimpar2(41);
--XOR 263 PAR
state3(4) <= statepar3(43) when sys_clk3 ='0' else
stateimpar3(43);
--XOR 242 IMPAR
k2n <= state2(0) XOR state2(1);
state3(0) <= stateimpar3(32) when sys_clk3 ='0' else
statepar3(32);
--XOR 287 PAR
state3(1) <= statepar3(55) when sys_clk3 ='0' else
stateimpar3(55);
--XOR 285 PAR
state3(2) <= statepar3(54) when sys_clk3 ='0' else
stateimpar3(54);
--XOR 286 IMPAR
state3(3) <= stateimpar3(54) when sys_clk3 ='0' else
statepar3(54);
--XOR 68 PAR
state1(4) <= statepar1(34) when sys_clk3 ='0' else
stateimpar1(34);
k3n <= state3(0) XOR state3(1);
t1n <= k1n XOR ((state1(2) AND state1(3)) XOR state2(4));
t2n <= k2n XOR ((state2(2) AND state2(3)) XOR state3(4));
```

```
t3n <= k3n XOR ((state3(2) AND state3(3)) XOR state1(4));
key_stream_n <= k1n XOR k2n XOR k3n;
key_out <= key_stream_n;
end Behavioral;
```

A.4. Trivium 2 bits

```
----------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREx2 - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium 2 bit.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test para
-- compararlos con la salida
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_corex2 is
    Port ( sys_clk : in  STD_LOGIC; --Reloj principal
           resetn :in  STD_LOGIC;
           cntrl : in STD_LOGIC_VECTOR(1 downto 0); --Control Bus
           key  : in  STD_LOGIC_VECTOR(79 downto 0);--Clave Secreta
-- Vector inicialización
           iv   : in  STD_LOGIC_VECTOR(79 downto 0);
--Secuencia cifrante.
           key_out : out  STD_LOGIC_VECTOR(1 downto 0));
end trivium_corex2;
architecture Behavioral of trivium_corex2 is
--Reg. int. LFSR
signal state : STD_LOGIC_VECTOR(295 downto 0) := (OTHERS => '0');
--Sec. cifrante.
signal key_stream : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS =>
'0');
--Nodos realimentación.
signal k1 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
signal k2 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
signal k3 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
```

```
signal t1 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
signal t2 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
signal t3 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
signal key_flip : STD_LOGIC_VECTOR(79 downto 0);
signal iv_flip : STD_LOGIC_VECTOR(79 downto 0);
-- Conversor 80-bit Big Endian a Little Endian (Revertir cada bit
-- del byte)
--ej 0x0123456789 -> 0x084C2A6E19
function little_endian (b: std_logic_vector) return
std_logic_vector is
      variable result : std_logic_vector(79 downto 0);
   begin
      for i in 0 to 9 loop
         result(((i*8)+7) downto (i*8)) := b(i*8) &
                                           b((i*8) + 1) &
                                           b((i*8) + 2) &
                                           b((i*8) + 3) &
                                           b((i*8) + 4) &
                                           b((i*8) + 5) &
                                           b((i*8) + 6) &
                                           b((i*8) + 7);
      end loop;
      return result;
   end;
begin
--Core LFSR con 3 registros de desplazamiento con realimentación
MAIN_TRIVIUM : process (sys_clk,cntrl,resetn)
begin
  if(resetn = '0') then
          state(287 downto 0) <= (OTHERS => '0');
  elsif(sys_clk'event AND sys_clK='1') then
     if (cntrl = "10") then   ---Intial state setup based on spec
        state(92 downto 0) <= "0000000000000" & key_flip;
        state(176 downto 93) <= X"0" & iv_flip;
        state(287 downto 177) <= "111" &
X"00000000000000000000000000";
     elsif ((cntrl = "11") OR (cntrl = "01")) then
---shift register
      state(92 downto 0) <= state(90 downto 0) & t3(1 downto 0);
      state(176 downto 93) <= state(174 downto 93) & t1(1 downto
0);
      state(287 downto 177) <= state(285 downto 177) & t2(1
downto 0);
     end if;
  end if;
end process;
--XOR Nodos
G1: for i in 0 to 1 generate
    k1(i) <= state(64+i) XOR state(91+i);
end generate G1;
G2: for i in 0 to 1 generate
      k2(i) <= state(160+i) XOR state(175+i);
end generate G2;
G3: for i in 0 to 1 generate
      k3(i) <= state(241+i) XOR state(286+i);
```

```
end generate G3;
G4: for i in 0 to 1 generate
      t1(i) <= k1(i) XOR ((state(89+i) AND state(90+i)) XOR
state(169+i));
end generate G4;
G5: for i in 0 to 1 generate
      t2(i) <= k2(i) XOR ((state(173+i) AND state(174+i)) XOR
state(262+i));
end generate G5;
G6: for i in 0 to 1 generate
      t3(i) <= k3(i) XOR ((state(284+i) AND state(285+i)) XOR
state(67+i));
end generate G6;
G7: for i in 0 to 1 generate
      key_stream(i) <= (k1(i) XOR k2(i) XOR k3(i));
end generate G7;
--Cambia valores de entrada para que las salidas coincidan con
los esperados
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
key_out(1 downto 0) <= key_stream(1 downto 0);
end Behavioral;
```

A.5. Trivium MPLP 2 bits

```
----------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREx2MPLP - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripcion: Cifrador Trivium 2 bit MPLP.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Additional Comments: Incluye bits de control para la carga de
-- key/IV.
-- Incluye función de conversión de los vectores de test para
-- compararlos con la salida
----------------------------------------------------------------
entity trivium_corex2mplp is
   Port ( sys_clk : in  std_logic; -- Reloj principal
      sys_clk3: in std_logic;
      resetn :in  std_logiC;
      cntrl : in std_logic_vector(1 downto 0);
--Control de la carga
```

```
      key  : in  std_logic_vector(79 downto 0);
--Clave secreta
      iv   : in  std_logic_vector(79 downto 0);
--Vector Inicializ.
      key_out : out  std_logic_vector (1 downto 0));
-- Secuencia cifrante
end trivium_corex2mplp;
architecture Behavioral of trivium_corex2mplp is
  signal key_flip : STD_LOGIC_VECTOR(79 downto 0);
  signal iv_flip : STD_LOGIC_VECTOR(79 downto 0);
  signal state1 : STD_LOGIC_VECTOR(92 downto 64) := (OTHERS =>
'0');  ---
  signal state2 : STD_LOGIC_VECTOR(176 downto 157) := (OTHERS =>
'0');  ---
  signal state3 : STD_LOGIC_VECTOR(287 downto 241) := (OTHERS =>
'0');  ---
---XOR Nodos de realimentación
  signal s1,s2,s3: STD_LOGIC_VECTOR(1 downto 0) := (OTHERS =>
'0');
  signal state : STD_LOGIC_VECTOR(287 downto 0) := (OTHERS =>
'0');
---PAR LFSR
  signal statepar1 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS =>
'0');
---IMPAR LFSR
  signal stateimpar1 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS
=> '0');
---PAR LFSR
  signal statepar2 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS =>
'0');
---IMPAR LFSR
  signal stateimpar2 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS
=> '0');
---PAR LFSR
  signal statepar3 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS =>
'0');
---IMPAR LFSR
  signal stateimpar3 : STD_LOGIC_VECTOR(31 downto 0) := (OTHERS
=> '0');
---Salida secuencia cifrante
  signal key_stream : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS =>
'0');
---XOR Nodos de realimentación
  signal k1 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
  signal k2 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
  signal k3 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
  signal t1 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
  signal t2 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
  signal t3 : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS => '0');
  signal qout,qout2 : STD_LOGIC :='0';   ---Clock *2
   -- signal sys_clk3  : STD_LOGIC;   ---Clock *2
-- Conversor 80-bit Big Endian a Little Endian Convert.
function little_endian (b: std_logic_vector) return
std_logic_vector is
   variable result : std_logic_vector(79 downto 0);
```

```
--ex 0x0123456789 -> 0x084C2A6E19
   begin
      for i in 0 to 9 loop
         result(((i*8)+7) downto (i*8)) :=  b(i*8) & b((i*8) + 1)
& b((i*8) + 2) & b((i*8) + 3) & b((i*8) + 4) & b((i*8) + 5) &
b((i*8) + 6) & b((i*8) + 7);
      end loop;
return result;
end;
begin
-- Definición procesos de reloj
SYS_CLK3_process :process  (sys_clk,resetn)
   begin
      IF(resetn = '0') then
         qout2  <= '0';
      ELSIF(sys_clk'event AND sys_clk='1') then
         qout2 <= NOT(qout2) ;
      END IF;
end process;
sys_clk3 <= NOT(qout2);
state(92 downto 0) <= "0000000000000" & key_flip;
state(176 downto 93) <= X"0" & iv_flip;
state(287 downto 177) <= "111" & X"00000000000000000000000000";
SECOND_TRIVIUM_RISE : process (sys_clk3,resetn)
--3 registros LFSR con realimentación de los nodos
   begin
      if(resetn = '0') then
         statepar1  <= (OTHERS => '0');
         statepar2  <= (OTHERS => '0');
         statepar3  <= (OTHERS => '0');
      elsif(sys_clk3'event AND sys_clk3='0') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
         if (cntrl = "10") then
-- FIRST PAR
         for i in 0 to 15 loop
            statepar1 (2*i) <= state (i*4);
            statepar1 (2*i+1) <= state (i*4+1);
         end loop;
         for i in 0 to 15 loop
               statepar2 (2*i) <= state (93+(4*i));
               statepar2 (2*i+1) <= state (93+1+(4*i));
         end loop;
-- THIRD
         for i in 0 to 15 loop
               statepar3 (2*i) <= state (177+(4*i));
               statepar3 (2*i+1) <= state (177+1+(4*i));
         end loop;
         elsif ((cntrl = "11") OR (cntrl = "01")) then
-- registro 47 bits
           statepar1(31 downto 0) <= statepar1(29 downto 0) & t3;
-- registro 42 bits
           statepar2(31 downto 0) <= statepar2(29 downto 0) & t1;
-- registro 54 bits + 1 bits mas
           statepar3(31 downto 0) <= statepar3(29 downto 0) & t2;
         end if;
```

```
   end if;
end process;
SECOND_TRIVIUM_FALL : process (sys_clk3,resetn)
--3 registros LFSR con realimentación de los nodos
   begin
      if(resetn = '0') then
         stateimpar1  <= (OTHERS => '0');
         stateimpar2  <= (OTHERS => '0');
         stateimpar3  <= (OTHERS => '0');
      elsif(sys_clk3'event AND sys_clk3='1') then
         if (cntrl = "10") then
 ---Intial state setup based on spec
-- FIRST
            for i in 0 to 15 loop
               stateimpar1 (2*i) <= state ((i*4)+2);
               stateimpar1 (2*i+1) <= state ((i*4)+3);
            end loop;
-- SECOND
            for i in 0 to 15 loop
               stateimpar2 (2*i) <= state (95+(4*i));
               stateimpar2 (2*i+1) <= state (95+1+(4*i));
            end loop;
-- THIRD PAR
            for i in 0 to 15 loop
               stateimpar3 (2*i) <= state (179+(4*i));
               stateimpar3 (2*i+1) <= state (179+1+(4*i));
            end loop;
         elsif ((cntrl = "11") OR (cntrl = "01")) then
            stateimpar1(31 downto 0) <= stateimpar1(29 downto 0)
& t3 ;
            stateimpar2(31 downto 0) <= stateimpar2(29 downto 0)
& t1 ;
            stateimpar3(31 downto 0) <= stateimpar3(29 downto 0)
& t2 ;
         end if;
      end if;
end process;
--Cambia valores de entrada para que las salidas coincidan con
-- los esperados
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
-- MODIFICACION
MAIN_TRIVIUM : process (sys_clk,cntrl,resetn)
--3 registros de desplazamiento LFSR con realimentación de los
-- nodos
begin
   if(resetn = '0') then
      state1  <= (OTHERS => '0');
      state2  <= (OTHERS => '0');
      state3  <= (OTHERS => '0');
   elsif(sys_clk'event AND sys_clk='1') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
      if (cntrl = "10") then
        state1 <= state(92 downto 64) ;    -- registro 93 bits
        state2 <= state(176 downto 157) ;  -- registro 84 bits
```

```
        state3 <= state(287 downto 241) ; -- registro 111 bits
      elsif ((cntrl = "11") OR (cntrl = "01")) then
         state1(92 downto 64) <= state1(90 downto 64) & s1;
         state2(176 downto 157) <= state2(174 downto 157) & s2;
         state3(287 downto 241) <= state3(285 downto 241) & s3;
      end if;
   end if;
end process;
-- MODIFICACION
s1 <= stateimpar1(31 downto 30) when sys_clk3 ='0' else
statepar1(31 downto 30);--Nodo salida
s2 <= stateimpar2(31 downto 30) when sys_clk3 ='0' else
statepar2(31 downto 30);--Nodo salida
s3 <= stateimpar3(31 downto 30) when sys_clk3 ='0' else
statepar3(31 downto 30);--Nodo salida
-- MODIFICACION
G1: for i in 0 to 1 generate
    k1(i) <= state1(64+i) XOR state1(91+i);
end generate G1;
G2: for i in 0 to 1 generate
    k2(i) <= state2(160+i) XOR state2(175+i);
end generate G2;
G3: for i in 0 to 1 generate
    k3(i) <= state3(241+i) XOR state3(286+i);
end generate G3;
G4: for i in 0 to 1 generate
    t1(i) <= k1(i) XOR ((state1(89+i) AND state1(90+i)) XOR
state2(169+i));
end generate G4;
G5: for i in 0 to 1 generate
   t2(i) <= k2(i) XOR ((state2(173+i) AND state2(174+i)) XOR
state3(262+i));
end generate G5;
G6: for i in 0 to 1 generate
    t3(i) <= k3(i) XOR ((state3(284+i) AND state3(285+i)) XOR
state1(67+i));
end generate G6;
G7: for i in 0 to 1 generate
      key_stream(i) <= (k1(i) XOR k2(i) XOR k3(i));
end generate G7;
key_out <= key_stream(1 downto 0);
end Behavioral;
```

A.6. Trivium FPLP 2 bits

```
------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREx2FPLP - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium 2 bit FPLP.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test
------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_corex2fplp is
   Port ( sys_clk : in  std_logic; -- Reloj principal
        resetn :in  std_logiC;
        cntrl : in std_logic_vector(1 downto 0);
--Control de la carga
        key  : in  std_logic_vector(79 downto 0);
--Clave secreta
        iv   : in  std_logic_vector(79 downto 0);
--Vector Inicializacion
        key_out : out  std_logic_vector (1 downto 0));
-- Secuencia cifrante
end trivium_corex2fplp;
architecture Behavioral of trivium_corex2fplp is
   signal key_flip : STD_LOGIC_VECTOR(79 downto 0);
   signal iv_flip : STD_LOGIC_VECTOR(79 downto 0);
   signal state1 : STD_LOGIC_VECTOR(4 downto 0) := (OTHERS =>
'0');  ---
   signal state1L : STD_LOGIC_VECTOR(4 downto 0) := (OTHERS =>
'0');  ---
   signal state2 : STD_LOGIC_VECTOR(4 downto 0) := (OTHERS =>
'0');  ---
   signal state2L : STD_LOGIC_VECTOR(4 downto 0) := (OTHERS =>
'0');  ---
   signal state3 : STD_LOGIC_VECTOR(4 downto 0) := (OTHERS =>
'0');  ---
   signal state3L : STD_LOGIC_VECTOR(4 downto 0) := (OTHERS =>
'0');  ---
```

```
   signal state : STD_LOGIC_VECTOR(287 downto 0) := (OTHERS =>
'0');  ---
   signal statepar1 : STD_LOGIC_VECTOR(46 downto 0) := (OTHERS =>
'0');
---IMPAR LFSR
   signal stateimpar1 : STD_LOGIC_VECTOR(46 downto 0) := (OTHERS
=> '0');
---PAR LFSR
   signal statepar2 : STD_LOGIC_VECTOR(41 downto 0) := (OTHERS =>
'0');
---IMPAR LFSR
   signal stateimpar2 : STD_LOGIC_VECTOR(41 downto 0) := (OTHERS
=> '0');
---PAR LFSR
   signal statepar3 : STD_LOGIC_VECTOR(55 downto 0) := (OTHERS =>
'0');
---IMPAR LFSR
   signal stateimpar3 : STD_LOGIC_VECTOR(55 downto 0) := (OTHERS
=> '0');
---Salida secuencia cifrante
   signal key_stream_n : STD_LOGIC_VECTOR(1 downto 0) := (OTHERS
=> '0');
---XOR Nodos de realimentación
   signal k1n,k2n,k3n,t1n,t2n,t3n : STD_LOGIC := '0';  --
   signal k1n_1,k2n_1,k3n_1,t1n_1,t2n_1,t3n_1 : STD_LOGIC := '0';
   signal qout,qout2 : STD_LOGIC :='0';   ---Reloj/2
   signal sys_clk3  : STD_LOGIC;   ---Reloj/2
--Nueva salida secuencia cifrante
   signal key_out_n2 : std_logic_vector (1 downto 0) := (OTHERS
=> '0');
--Nueva salida secuencia cifrante
   signal key_stream_n2 : std_logic_vector (1 downto 0) :=
(OTHERS => '0');
-- Conversor 80-bit Big Endian a Little Endian
-- (Revertir cada bit del byte)
function little_endian (b: std_logic_vector) return
std_logic_vector is
   variable result : std_logic_vector(79 downto 0);
--ex 0x0123456789 -> 0x084C2A6E19
   begin
      for i in 0 to 9 loop
         result(((i*8)+7) downto (i*8)) :=  b(i*8) & b((i*8) + 1)
& b((i*8) + 2) & b((i*8) + 3) & b((i*8) + 4) & b((i*8) + 5) &
b((i*8) + 6) & b((i*8) + 7);
      end loop;
return result;
end;
begin
-- Definición procesos de reloj
SYS_CLK3_process :process (sys_clk,resetn)
   begin
      if(resetn = '0') then
         qout2  <= '0';
      elsif(sys_clk'event and sys_clk='1') then
         qout2 <= NOT(qout2) ;
```

```
      end if;
end process;
sys_clk3 <= NOT(qout2);
state(92 downto 0) <= "0000000000000" & key_flip;
state(176 downto 93) <= X"0" & iv_flip;
state(287 downto 177) <= "111" & X"00000000000000000000000000";
SECOND_TRIVIUM_RISE : process (sys_clk3,cntrl,resetn) ---
--3 registros de desplazamiento LFSR con realimentación de los
-- nodos
   begin --Core must be loaded with key and
      if(resetn = '0') then
         statepar1  <= (OTHERS => '0');
         statepar2  <= (OTHERS => '0');
         statepar3  <= (OTHERS => '0');
      elsif(sys_clk3'event AND sys_clk3='0') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
         if (cntrl = "10") then
-- FIRST PAR
         for i in 0 to 22 loop
            statepar1 (2*i) <= state (i*4);
            statepar1 (2*i+1) <= state (i*4+1);
            statepar1 (46) <= state (92);
         end loop;
         for i in 0 to 20 loop
               statepar2 (2*i) <= state (93+(4*i));
               statepar2 (2*i+1) <= state (93+1+(4*i));
         end loop;
-- THIRD
         for i in 0 to 27 loop
               statepar3 (2*i) <= state (177+(4*i));
               statepar3 (2*i+1) <= state (177+1+(4*i));
         end loop;
         elsif ((cntrl = "11") OR (cntrl = "01")) then
 -- registro 47 bits
            statepar1(46 downto 0) <= statepar1(44 downto 0) &
t3n & t3n_1;   -- registro 42 bits
            statepar2(41 downto 0) <= statepar2(39 downto 0) &
t1n & t1n_1;  -- registro 55 bits + 1 bits mas
            statepar3(55 downto 0) <= statepar3(53 downto 0)  &
t2n & t2n_1;
         end if;
   end if;
end process;
SECOND_TRIVIUM_FALL : process (sys_clk3,cntrl,resetn)
--3 registros de desplazamiento LFSR con realimentación de los
-- nodos
  begin  --Core must be loaded with key and
    if(resetn = '0') then
         stateimpar1  <= (OTHERS => '0');
         stateimpar2  <= (OTHERS => '0');
         stateimpar3  <= (OTHERS => '0');
    elsif(sys_clk3'event AND sys_clk3='1') then
      if (cntrl = "10") then
 ---Configuración estado Inicial
-- FIRST
```

```
         for i in 0 to 22 loop
               stateimpar1 (2*i) <= state ((i*4)+2);
               stateimpar1 (2*i+1) <= state ((i*4)+3);
         end loop;
-- SECOND
         for i in 0 to 20 loop
               stateimpar2 (2*i) <= state (95+(4*i));
               stateimpar2 (2*i+1) <= state (95+1+(4*i));
         end loop;
-- THIRD PAR
         for i in 0 to 26 loop
               stateimpar3 (2*i) <= state (179+(4*i));
               stateimpar3 (2*i+1) <= state (179+1+(4*i));
               stateimpar3 (54) <= state (287);
         end loop;
     elsif ((cntrl = "11") OR (cntrl = "01")) then
--- registro 47 bits
        stateimpar1(46 downto 0) <= stateimpar1(44 downto 0) &
t3n & t3n_1 ;   -- registro 42 bits
        stateimpar2(41 downto 0) <= stateimpar2(39 downto 0) &
t1n & t1n_1 ;-- registro 56 bits
        stateimpar3(55 downto 0) <= stateimpar3(53 downto 0) &
t2n & t2n_1 ;
         end if;
      end if;
end process;
--Cambia valores de entrada para que las salidas coincidan con
-- los esperados
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
-- MODIFICACION
--XOR Nodos 65
state1(0) <= statepar1(33) when sys_clk3 ='0' else
stateimpar1(33);
--XOR Nodos 64
state1L(0) <= statepar1(32) when sys_clk3 ='0' else
stateimpar1(32);
--XOR Nodos 92
state1(1) <= statepar1(46) when sys_clk3 ='0' else
stateimpar1(46);
--XOR Nodes 91
state1L(1) <= stateimpar1(45) when sys_clk3 ='0' else
statepar1(45);
--XOR Nodos 90
state1(2) <= stateimpar1(44) when sys_clk3 ='0' else
statepar1(44);
--XOR Nodos 89
state1L(2) <= statepar1(45) when sys_clk3 ='0' else
stateimpar1(45);
--XOR Nodos 91
state1(3) <= stateimpar1(45) when sys_clk3 ='0' else
statepar1(45);
--XOR Nodos 90
state1L(3) <= stateimpar1(44) when sys_clk3 ='0' else
statepar1(44);
```

```
--XOR Nodos 170
state2(4) <= statepar2(39) when sys_clk3 ='0' else
stateimpar2(39);
--XOR Nodos 169
state2L(4) <= statepar2(38) when sys_clk3 ='0' else
stateimpar2(38);
k1n <= state1(0) XOR state1(1);
k1n_1 <= state1L(0) XOR state1L(1);
--XOR Nodos 161
state2(0) <= statepar2(34) when sys_clk3 ='0' else
stateimpar2(34);
--XOR Nodos 160
state2L(0) <= stateimpar2(33) when sys_clk3 ='0' else
statepar2(33);
--XOR Nodos 176
state2(1) <= stateimpar2(41) when sys_clk3 ='0' else
statepar2(41);
--XOR Nodos 175
state2L(1) <= stateimpar2(40) when sys_clk3 ='0' else
statepar2(40);
--XOR Nodos 174
state2(2) <= statepar2(41) when sys_clk3 ='0' else
stateimpar2(41);
--XOR Nodos 173
state2L(2) <= statepar2(40) when sys_clk3 ='0' else
stateimpar2(40);
--XOR Nodos 175
state2(3) <= stateimpar2(40) when sys_clk3 ='0' else
statepar2(40);
--XOR Nodos 174
state2L(3) <= statepar2(41) when sys_clk3 ='0' else
stateimpar2(41);
--XOR Nodos 163
state3(4) <= stateimpar3(42) when sys_clk3 ='0' else
statepar3(42);
--XOR Nodos 262
state3L(4) <= statepar3(43) when sys_clk3 ='0' else
stateimpar3(43);
k2n <= state2(0) XOR state2(1);
k2n_1 <= state2L(0) XOR state2L(1);
--XOR Nodos 242
state3(0) <= statepar3(33) when sys_clk3 ='0' else
stateimpar3(33);
--XOR Nodos 241
state3L(0) <= statepar3(32) when sys_clk3 ='0' else
stateimpar3(32);
--XOR Nodos 287
state3(1) <= stateimpar3(54) when sys_clk3 ='0' else
statepar3(54);
--XOR Nodos 286
state3L(1) <= statepar3(55) when sys_clk3 ='0' else
stateimpar3(55);
--XOR Nodos 285
state3(2) <= statepar3(54) when sys_clk3 ='0' else
stateimpar3(54);
```

```
--XOR Nodos 284
state3L(2) <= stateimpar3(53) when sys_clk3 ='0' else
statepar3(53);
--XOR Nodos 286
state3(3) <= statepar3(55) when sys_clk3 ='0' else
stateimpar3(55);
--XOR Nodos 285
state3L(3) <= statepar3(54) when sys_clk3 ='0' else
stateimpar3(54);
--XOR Nodos 68
state1(4) <= statepar1(34) when sys_clk3 ='0' else
stateimpar1(34);
--XOR Nodos  67
state1L(4) <= stateimpar1(33) when sys_clk3 ='0' else
statepar1(33);
k3n <= state3(0) XOR state3(1);
k3n_1 <= state3L(0) XOR state3L(1);
t1n <= k1n XOR ((state1(2) AND state1(3)) XOR state2(4));
t1n_1 <= k1n_1 XOR ((state1L(2) AND state1L(3)) XOR state2L(4));
t2n <= k2n XOR ((state2(2) AND state2(3)) XOR state3(4));
t2n_1 <= k2n_1 XOR ((state2L(2) AND state2L(3)) XOR state3L(4));
t3n <= k3n XOR ((state3(2) AND state3(3)) XOR state1(4));
t3n_1 <= k3n_1 XOR ((state3L(2) AND state3L(3)) XOR state1L(4));
key_stream_n(1) <= k1n XOR k2n XOR k3n;
key_stream_n(0) <= k1n_1 XOR k2n_1 XOR k3n_1;
key_out <= key_stream_n(1 downto 0);
end Behavioral;
```

A.7. Trivium 8 bits

```
----------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREx8 - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium 8 bits.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test
----------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_corex8 is
    Port ( sys_clk : in  STD_LOGIC; --Reloj principal
           resetn :in  STD_LOGIC;
           cntrl : in STD_LOGIC_VECTOR(1 downto 0);
-- Control Bus
           key  : in  STD_LOGIC_VECTOR(79 downto 0);
-- Clave Secreta 80-bit
           iv   : in  STD_LOGIC_VECTOR(79 downto 0);
--Vector inicialización
           key_out : out  STD_LOGIC_VECTOR(7 downto 0)); --
Secuencia cifrante
end trivium_corex8;
architecture Behavioral of trivium_corex8 is
signal state : STD_LOGIC_VECTOR(295 downto 0) := (OTHERS => '0');
--Reg. int. LFSR
signal key_stream : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS =>
'0');
--Sec. cifrante.
signal k1 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
--Nodos realimentación.
signal k2 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
--Nodos realimentación.
signal k3 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
--Nodos realimentación.
signal t1 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
--Nodos realimentación.
signal t2 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
--Nodos realimentación.
signal t3 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
--Nodos realimentación.
signal key_flip : STD_LOGIC_VECTOR(79 downto 0);
signal iv_flip : STD_LOGIC_VECTOR(79 downto 0);
-- Conversor 80-bit Big Endian a Little Endian Convert.
--ej 0x0123456789 -> 0x084C2A6E19
function little_endian (b: std_logic_vector) return
std_logic_vector is
       variable result : std_logic_vector(79 downto 0);
   begin
       for i in 0 to 9 loop
          result(((i*8)+7) downto (i*8)) := b(i*8) &
                                            b((i*8) + 1) &
                                            b((i*8) + 2) &
                                            b((i*8) + 3) &
                                            b((i*8) + 4) &
                                            b((i*8) + 5) &
                                            b((i*8) + 6) &
                                            b((i*8) + 7);
       end loop;
       return result;
   end;
begin
MAIN_TRIVIUM : process (sys_clk,cntrl,resetn)
-- Core LFSR con 3 registros de
```

```
-- desplazamiento con realimentación de los nodos (t1,t2,t3).
begin
   if(resetn = '0') then
     state(287 downto 0) <= (OTHERS => '0');
   elsif(sys_clk'event AND sys_clK='1') then
     if (cntrl = "10") then   ---
          state(92 downto 0) <= "0000000000000" & key_flip;
          state(176 downto 93) <= X"0" & iv_flip;
          state(287 downto 177) <= "111" &
X"00000000000000000000000000";
      elsif ((cntrl = "11") OR (cntrl = "01")) then
---Desplazar
          state(92 downto 0) <= state(84 downto 0) & t3(7 downto
0);
          state(176 downto 93) <= state(168 downto 93) & t1(7
downto 0);
          state(287 downto 177) <= state(279 downto 177) & t2(7
downto 0);
       end if;
    end if;
end process;
--XOR Nodos
G1: for i in 0 to 7 generate
    k1(i) <= state(58+i) XOR state(85+i);
end generate G1;
G2: for i in 0 to 7 generate
      k2(i) <= state(154+i) XOR state(169+i);
end generate G2;
G3: for i in 0 to 7 generate
      k3(i) <= state(235+i) XOR state(280+i);
end generate G3;
G4: for i in 0 to 7 generate
      t1(i) <= k1(i) XOR ((state(83+i) AND state(84+i)) XOR
state(163+i));
end generate G4;
G5: for i in 0 to 7 generate
      t2(i) <= k2(i) XOR ((state(167+i) AND state(168+i)) XOR
state(256+i));
end generate G5;
G6: for i in 0 to 7 generate
      t3(i) <= k3(i) XOR ((state(278+i) AND state(279+i)) XOR
state(61+i));
end generate G6;
G7: for i in 0 to 7 generate
      key_stream(i) <= (k1(i) XOR k2(i) XOR k3(i));
end generate G7;
--Cambia valores de entrada para que las salidas coincidan con
los esperados
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
key_out(7 downto 0) <= key_stream(7 downto 0);
end Behavioral;
```

A.8. Trivium MPLP 8 bits

```
----------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREx8MPLP - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium MPLP 8 bits.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_corex8mplp is
   Port ( sys_clk : in  std_logic; --Reloj principal
      resetn :in  std_logiC;
      cntrl : in std_logic_vector(1 downto 0); --Control Bus
      key  : in  std_logic_vector(79 downto 0); );
--Clave Secreta 80-bit
      iv   : in  std_logic_vector(79 downto 0);
--Vector inicialización
      key_out : out  std_logic_vector (7 downto 0));
-- Secuencia cifrante
end trivium_corex8mplp;
architecture Behavioral of trivium_corex8mplp is

signal key_flip : STD_LOGIC_VECTOR(79 downto 0);
signal iv_flip : STD_LOGIC_VECTOR(79 downto 0);
signal state1 : STD_LOGIC_VECTOR(92 downto 48) := (OTHERS =>
'0');  ---
signal state2 : STD_LOGIC_VECTOR(176 downto 141) := (OTHERS =>
'0');  ---
signal state3 : STD_LOGIC_VECTOR(287 downto 225) := (OTHERS =>
'0');  ---
signal s1,s2,s3: STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
---Regitro interno LFSR
signal state : STD_LOGIC_VECTOR(287 downto 0) := (OTHERS => '0');
---Regitro PAR-1 interno LFSR
signal statepar1 : STD_LOGIC_VECTOR(23 downto 0) := (OTHERS =>
'0');
```

```
---Regitro IMPAR-1 interno LFSR
signal stateimpar1 : STD_LOGIC_VECTOR(23 downto 0) := (OTHERS =>
'0');
---Regitro PAR-2 interno LFSR
signal statepar2 : STD_LOGIC_VECTOR(23 downto 0) := (OTHERS =>
'0');
---Regitro IMPAR-2 interno LFSR
signal stateimpar2 : STD_LOGIC_VECTOR(23 downto 0) := (OTHERS =>
'0');
---Regitro PAR-3 interno LFSR
signal statepar3 : STD_LOGIC_VECTOR(23 downto 0) := (OTHERS =>
'0');
---Regitro IMPAR-3 interno LFSR
signal stateimpar3 : STD_LOGIC_VECTOR(23 downto 0) := (OTHERS =>
'0');
---Salida secuencia cifrante
signal key_stream : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS =>
'0');
--Señales realimentación
signal k1 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
signal k2 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
signal k3 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
signal t1 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
signal t2 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
signal t3 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');

signal qout,qout2 : STD_LOGIC :='0';   ---Clock *2
signal sys_clk3  : STD_LOGIC;   ---Clock *2
-- Conversor 80-bit Big Endian a Little Endian Convert.
function little_endian (b: std_logic_vector) return
std_logic_vector is
   variable result : std_logic_vector(79 downto 0);
--ex 0x0123456789 -> 0x084C2A6E19
   begin
      for i in 0 to 9 loop
         result(((i*8)+7) downto (i*8)) :=  b(i*8) & b((i*8)+1) &
b((i*8)+2) & b((i*8)+3) & b((i*8)+4) & b((i*8)+5) & b((i*8)+6) &
b((i*8)+7);
      end loop;
return result;
end;
begin
-- Definición relojes
SYS_CLK3_process :process  (sys_clk,resetn)
   begin
      if(resetn = '0') then
         qout2  <= '0';
      elsif(sys_clk'event AND sys_clk='1') then
         qout2 <= NOT(qout2) ;
      end if;
end process;
sys_clk3 <= NOT(qout2);

--Cambia valores de entrada para que las salidas coincidan con
-- los esperados
```

```
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
state(92 downto 0) <= "0000000000000" & key_flip;
state(176 downto 93) <= X"0" & iv_flip;
state(287 downto 177) <= "111" & X"00000000000000000000000000";
SECOND_TRIVIUM_RISE : process (sys_clk3,cntrl,resetn)
---Core LFSR con 3 registros de
-- desplazamiento con realimentación de los nodos (t1,t2,t3).
begin --Core must be loaded with key and
   if(resetn = '0') then
      statepar1  <= (OTHERS => '0');
      statepar2  <= (OTHERS => '0');
      statepar3  <= (OTHERS => '0');
   elsif(sys_clk3'event AND sys_clk3='0') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
     if (cntrl = "10") then      --Intial state setup spec
-- FIRST PAR, entran 8 bits de una vez
     for i in 0 to 2 loop
        statepar1 (8*i) <= state (i*16);
        statepar1 (8*i+1) <= state (i*16+1);
        statepar1 (8*i+2) <= state (i*16+2);
        statepar1 (8*i+3) <= state (i*16+3);
        statepar1 (8*i+4) <= state (i*16+4);
        statepar1 (8*i+5) <= state (i*16+5);
        statepar1 (8*i+6) <= state (i*16+6);
        statepar1 (8*i+7) <= state (i*16+7);
     end loop;
     for i in 0 to 2 loop
           statepar2 (8*i) <= state (93+(16*i));
           statepar2 (8*i+1) <= state (93+1+(16*i));
           statepar2 (8*i+2) <= state (93+2+(16*i));
           statepar2 (8*i+3) <= state (93+3+(16*i));
           statepar2 (8*i+4) <= state (93+4+(16*i));
           statepar2 (8*i+5) <= state (93+5+(16*i));
           statepar2 (8*i+6) <= state (93+6+(16*i));
           statepar2 (8*i+7) <= state (93+7+(16*i));
     end loop;
-- THIRD
     for i in 0 to 2 loop
           statepar3 (8*i) <= state (177+(16*i));
           statepar3 (8*i+1) <= state (177+1+(16*i));
           statepar3 (8*i+2) <= state (177+2+(16*i));
           statepar3 (8*i+3) <= state (177+3+(16*i));
           statepar3 (8*i+4) <= state (177+4+(16*i));
           statepar3 (8*i+5) <= state (177+5+(16*i));
           statepar3 (8*i+6) <= state (177+6+(16*i));
           statepar3 (8*i+7) <= state (177+7+(16*i));
     end loop;
     elsif ((cntrl = "11") OR (cntrl = "01")) then
---shift register mode -- registro 24 bits
      statepar1(23 downto 0) <= statepar1(15 downto 0) & t3(7
downto 0);
-- registro 24 bits.
      statepar2(23 downto 0) <= statepar2(15 downto 0) & t1(7
downto 0);  -- registro 24 bits
```

```
      statepar3(23 downto 0) <= statepar3(15 downto 0)  & t2(7
downto 0); --
     end if;
   end if;
end process;
SECOND_TRIVIUM_FALL : process (sys_clk3,cntrl,resetn)
-- Core LFSR con 3 registros de
-- desplazamiento con realimentación de los nodos (t1,t2,t3).
begin
   if(resetn = '0') then
      stateimpar1  <= (OTHERS => '0');
      stateimpar2  <= (OTHERS => '0');
      stateimpar3  <= (OTHERS => '0');
   elsif(sys_clk3'event AND sys_clk3='1') then
     if (cntrl = "10") then       ---
-- FIRST
        for i in 0 to 2 loop
           stateimpar1 (8*i) <= state ((i*16)+8);
           stateimpar1 (8*i+1) <= state ((i*16)+8+1);
           stateimpar1 (8*i+2) <= state ((i*16)+8+2);
           stateimpar1 (8*i+3) <= state ((i*16)+8+3);
           stateimpar1 (8*i+4) <= state ((i*16)+8+4);
           stateimpar1 (8*i+5) <= state ((i*16)+8+5);
           stateimpar1 (8*i+6) <= state ((i*16)+8+6);
           stateimpar1 (8*i+7) <= state ((i*16)+8+7);
        end loop;
-- SECOND
        for i in 0 to 2 loop
           stateimpar2 (8*i) <= state (101+(16*i));
           stateimpar2 (8*i+1) <= state (101+1+(16*i));
           stateimpar2 (8*i+2) <= state (101+2+(16*i));
           stateimpar2 (8*i+3) <= state (101+3+(16*i));
           stateimpar2 (8*i+4) <= state (101+4+(16*i));
           stateimpar2 (8*i+5) <= state (101+5+(16*i));
           stateimpar2 (8*i+6) <= state (101+6+(16*i));
           stateimpar2 (8*i+7) <= state (101+7+(16*i));
        end loop;
-- THIRD PAR
        for i in 0 to 2 loop
             stateimpar3 (8*i) <= state (185+(16*i));
           stateimpar3 (8*i+1) <= state (185+1+(16*i));
           stateimpar3 (8*i+2) <= state (185+2+(16*i));
           stateimpar3 (8*i+3) <= state (185+3+(16*i));
           stateimpar3 (8*i+4) <= state (185+4+(16*i));
           stateimpar3 (8*i+5) <= state (185+5+(16*i));
           stateimpar3 (8*i+6) <= state (185+6+(16*i));
           stateimpar3 (8*i+7) <= state (185+7+(16*i));
        end loop;
    elsif ((cntrl = "11") OR (cntrl = "01")) then
-- registro 24 bit
      stateimpar1(23 downto 0) <= stateimpar1(15 downto 0) & t3(7
downto 0);
-- registro 24 bit
      stateimpar2(23 downto 0) <= stateimpar2(15 downto 0) & t1(7
downto 0);
```

```
-- registro 24 bits
      stateimpar3(23 downto 0) <= stateimpar3(15 downto 0) & t2(7
downto 0);
    end if;
  end if;
end process;
MAIN_TRIVIUM : process (sys_clk,cntrl,resetn)
---Core LFSR con 3 registros de
-- desplazamiento con realimentación de los nodos (t1,t2,t3).
begin
   if(resetn = '0') then
      state1  <= (OTHERS => '0');
      state2  <= (OTHERS => '0');
      state3  <= (OTHERS => '0');
   elsif(sys_clk'event AND sys_clk='1') then
   -- PRIMERO DESPLAZO POR IMPAR!!!!
      if (cntrl = "10") then
         state1 <= state(92 downto 48) ;    -- registro 93 bits
         state2 <= state(176 downto 141) ;  -- registro 84 bits
         state3 <= state(287 downto 225) ; -- registro 111 bits
      elsif ((cntrl = "11") OR (cntrl = "01")) then   ---shift
register mo
         state1(92 downto 48) <= state1(84 downto 48) & s1;   --
93 bits
         state2(176 downto 141) <= state2(168 downto 141) & s2;
-- 84 bits
         state3(287 downto 225) <= state3(279 downto 225) & s3; -
- 111 bits
      end if;
   end if;
end process;
-- MODIFICACION
s1 <= stateimpar1(23 downto 16) when sys_clk3 ='0' else
statepar1(23 downto 16);--Nodo salida shift reg 1
s2 <= stateimpar2(23 downto 16) when sys_clk3 ='0' else
statepar2(23 downto 16);--Nodo salida shift reg 1
s3 <= stateimpar3(23 downto 16) when sys_clk3 ='0' else
statepar3(23 downto 16);--Nodo salida shift reg 1
-- MODIFICACION
G1: for i in 0 to 7 generate
    k1(i) <= state1(58+i) XOR state1(85+i);
end generate G1;
G2: for i in 0 to 7 generate
      k2(i) <= state2(154+i) XOR state2(169+i);
end generate G2;
G3: for i in 0 to 7 generate
      k3(i) <= state3(235+i) XOR state3(280+i);
end generate G3;
G4: for i in 0 to 7 generate
      t1(i) <= k1(i) XOR ((state1(83+i) AND state1(84+i)) XOR
state2(163+i));
end generate G4;
G5: for i in 0 to 7 generate
      t2(i) <= k2(i) XOR ((state2(167+i) AND state2(168+i)) XOR
state3(256+i));
```

```
end generate G5;
G6: for i in 0 to 7 generate
      t3(i) <= k3(i) XOR ((state3(278+i) AND state3(279+i)) XOR
state1(61+i));
end generate G6;
G7: for i in 0 to 7 generate
      key_stream(i) <= (k1(i) XOR k2(i) XOR k3(i));
end generate G7;
-----------------------------------
key_out <= key_stream(7 downto 0);
end Behavioral;
```

A.9. Trivium FPLP 8 bits

```
----------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREx8FPLP - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium FPLP 8 bits.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_corex8FPLP is
   Port ( sys_clk : in  std_logic; --Reloj principal
      resetn :in  std_logiC;
      cntrl : in std_logic_vector(1 downto 0); --Control Bus
      key  : in  std_logic_vector(79 downto 0); );
--Clave Secreta 80-bit.
      iv   : in  std_logic_vector(79 downto 0);
--Vector inicialización
      key_out : out  std_logic_vector (7 downto 0));
--Secuencia cifrante
end trivium_corex8FPLP;
architecture Behavioral of trivium_corex8FPLP is
   signal key_flip : STD_LOGIC_VECTOR(79 downto 0);
   signal iv_flip : STD_LOGIC_VECTOR(79 downto 0);
```

```
   signal state1 : STD_LOGIC_VECTOR(288 downto 0) := (OTHERS =>
'0');
---Regitro interno LFSR
   signal state : STD_LOGIC_VECTOR(288 downto 0) := (OTHERS =>
'0');
---Regitro PAR-1 interno LFSR
   signal statepar1 : STD_LOGIC_VECTOR(47 downto 0) := (OTHERS =>
'0');
---Regitro IMPAR-1 interno LFSR
   signal stateimpar1 : STD_LOGIC_VECTOR(47 downto 0) := (OTHERS
=> '0');
---Regitro PAR-2 interno LFSR
   signal statepar2 : STD_LOGIC_VECTOR(47 downto 0) := (OTHERS =>
'0');
---Regitro IMPAR-2 interno LFSR
   signal stateimpar2 : STD_LOGIC_VECTOR(47 downto 0) := (OTHERS
=> '0');
---Regitro PAR-3 interno LFSR
   signal statepar3 : STD_LOGIC_VECTOR(55 downto 0) := (OTHERS =>
'0');
---Regitro IMPAR-3 interno LFSR
   signal stateimpar3 : STD_LOGIC_VECTOR(55 downto 0) := (OTHERS
=> '0');
---Salida secuencia cifrante
   signal key_stream : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS =>
'0');
--Señales realimentación
   signal k1 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
   signal k2 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
   signal k3 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
   signal t1 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
   signal t2 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
   signal t3 : STD_LOGIC_VECTOR(7 downto 0) := (OTHERS => '0');
   signal qout,qout2 : STD_LOGIC :='0';   ---Reloj/2
   signal sys_clk3  : STD_LOGIC;   --- Reloj/2
-- Conversor 80-bit Big Endian a Little Endian Convert.
function little_endian (b: std_logic_vector) return
std_logic_vector is
   variable result : std_logic_vector(79 downto 0);
--ex 0x0123456789 -> 0x084C2A6E19
   begin
      for i in 0 to 9 loop
         result(((i*8)+7) downto (i*8)) :=  b(i*8) & b((i*8)+1) &
b((i*8)+2) & b((i*8)+3) & b((i*8)+4) & b((i*8)+5) & b((i*8)+6) &
b((i*8)+7);
      end loop;
return result;
end;
begin
-- Definición relojes
SYS_CLK3_process :process  (sys_clk,resetn)
   begin
      if(resetn = '0') then
         qout2  <= '0';
      elsif(sys_clk'event AND sys_clk='1') then
```

```
         qout2 <= NOT(qout2) ;
      end if;
end process;
sys_clk3 <= NOT(qout2);
-- Cambia valores de entrada para que las salidas coincidan con
-- los esperados
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
state(92 downto 0) <= "0000000000000" & key_flip;
state(176 downto 93) <= X"0" & iv_flip;
state(287 downto 177) <= "111" & X"00000000000000000000000000";
SECOND_TRIVIUM_RISE : process (sys_clk3,cntrl,resetn)
---Core LFSR con 3 registros de
--desplazamiento con realimentación de los nodos (t1,t2,t3).
   begin
      if(resetn = '0') then
         statepar1  <= (OTHERS => '0');
         statepar2  <= (OTHERS => '0');
         statepar3  <= (OTHERS => '0');
      elsif(sys_clk3'event AND sys_clk3='0') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
         if (cntrl = "10") then
-- PRIMERO PAR, entran 8 bits de una vez
         for i in 0 to 5 loop
            statepar1 (8*i) <= state (i*16);
            statepar1 (8*i+1) <= state (i*16+1);
            statepar1 (8*i+2) <= state (i*16+2);
            statepar1 (8*i+3) <= state (i*16+3);
            statepar1 (8*i+4) <= state (i*16+4);
            statepar1 (8*i+5) <= state (i*16+5);
            statepar1 (8*i+6) <= state (i*16+6);
            statepar1 (8*i+7) <= state (i*16+7);
         end loop;
         for i in 0 to 5 loop
               statepar2 (8*i) <= state (93+(16*i));
               statepar2 (8*i+1) <= state (93+1+(16*i));
               statepar2 (8*i+2) <= state (93+2+(16*i));
               statepar2 (8*i+3) <= state (93+3+(16*i));
               statepar2 (8*i+4) <= state (93+4+(16*i));
               statepar2 (8*i+5) <= state (93+5+(16*i));
               statepar2 (8*i+6) <= state (93+6+(16*i));
               statepar2 (8*i+7) <= state (93+7+(16*i));
         end loop;
-- THIRD
         for i in 0 to 6 loop
               statepar3 (8*i) <= state (177+(16*i));
               statepar3 (8*i+1) <= state (177+1+(16*i));
               statepar3 (8*i+2) <= state (177+2+(16*i));
               statepar3 (8*i+3) <= state (177+3+(16*i));
               statepar3 (8*i+4) <= state (177+4+(16*i));
               statepar3 (8*i+5) <= state (177+5+(16*i));
               statepar3 (8*i+6) <= state (177+6+(16*i));
               statepar3 (8*i+7) <= state (177+7+(16*i));
         end loop;
         elsif ((cntrl = "11") OR (cntrl = "01")) then
```

```
-- registro 48 bits. SE AÑADE UN BIT MAS AL AGRUPAR EN 8 BITS
          statepar1(47 downto 0) <= statepar1(39 downto 0) & t3(7
downto 0);   -- registro 48 bits. SE AÑADE UN BIT MAS AL AGRUPAR
EN 8 BITS
          statepar2(47 downto 0) <= statepar2(39 downto 0) & t1(7
downto 0);  -- registro 56 bits
          statepar3(55 downto 0) <= statepar3(47 downto 0)  & t2(7
downto 0);
          end if;
   end if;
end process;
SECOND_TRIVIUM_FALL : process (sys_clk3,cntrl,resetn)
---Core LFSR con 3 registros de
--desplazamiento con realimentación de los nodos (t1,t2,t3).
   begin  --Core must be loaded with key and
      if(resetn = '0') then
         stateimpar1  <= (OTHERS => '0');
         stateimpar2  <= (OTHERS => '0');
         stateimpar3  <= (OTHERS => '0');
      elsif(sys_clk3'event AND sys_clk3='1') then
         if (cntrl = "10") then
---Intial state setup based on spec
-- FIRST
            for i in 0 to 5 loop
               stateimpar1 (8*i) <= state ((i*16)+8);
               stateimpar1 (8*i+1) <= state ((i*16)+8+1);
               stateimpar1 (8*i+2) <= state ((i*16)+8+2);
               stateimpar1 (8*i+3) <= state ((i*16)+8+3);
               stateimpar1 (8*i+4) <= state ((i*16)+8+4);
               stateimpar1 (8*i+5) <= state ((i*16)+8+5);
               stateimpar1 (8*i+6) <= state ((i*16)+8+6);
               stateimpar1 (8*i+7) <= state ((i*16)+8+7);
            end loop;
-- SECOND
            for i in 0 to 4 loop
               stateimpar2 (8*i) <= state (101+(16*i));
               stateimpar2 (8*i+1) <= state (101+1+(16*i));
               stateimpar2 (8*i+2) <= state (101+2+(16*i));
               stateimpar2 (8*i+3) <= state (101+3+(16*i));
               stateimpar2 (8*i+4) <= state (101+4+(16*i));
               stateimpar2 (8*i+5) <= state (101+5+(16*i));
               stateimpar2 (8*i+6) <= state (101+6+(16*i));
               stateimpar2 (8*i+7) <= state (101+7+(16*i));
            end loop;
-- THIRD PAR
            for i in 0 to 6 loop
               stateimpar3 (8*i) <= state (185+(16*i));
               stateimpar3 (8*i+1) <= state (185+1+(16*i));
               stateimpar3 (8*i+2) <= state (185+2+(16*i));
               stateimpar3 (8*i+3) <= state (185+3+(16*i));
               stateimpar3 (8*i+4) <= state (185+4+(16*i));
               stateimpar3 (8*i+5) <= state (185+5+(16*i));
               stateimpar3 (8*i+6) <= state (185+6+(16*i));
               stateimpar3 (8*i+7) <= state (185+7+(16*i));
            end loop;
```

```
         elsif ((cntrl = "11") OR (cntrl = "01")) then
-- registro 48 bits generation
      stateimpar1(47 downto 0) <= stateimpar1(39 downto 0) & t3(7
downto 0);   -- registro 48 bits.
      stateimpar2(47 downto 0) <= stateimpar2(39 downto 0) & t1(7
downto 0);
-- registro 56 bits
      stateimpar3(55 downto 0) <= stateimpar3(47 downto 0) & t2(7
downto 0);
         end if;
      end if;
end process;
-- MODIFICACION
G1: for i in 0 to 7 generate
   k1(i) <= state1(58+i) XOR state1(85+i);
end generate G1;
G2: for i in 0 to 7 generate
   k2(i) <= state1(154+i) XOR state1(169+i);
end generate G2;
G3: for i in 0 to 7 generate
   k3(i) <= state1(235+i) XOR state1(280+i);
end generate G3;
G4: for i in 0 to 7 generate
   t1(i) <= k1(i) XOR ((state1(83+i) AND state1(84+i)) XOR
state1(163+i));
end generate G4;
G5: for i in 0 to 7 generate
   t2(i) <= k2(i) XOR ((state1(167+i) AND state1(168+i)) XOR
state1(256+i));
end generate G5;
G6: for i in 0 to 7 generate
   t3(i) <= k3(i) XOR ((state1(278+i) AND state1(279+i)) XOR
state1(61+i));
end generate G6;
G7: for i in 0 to 7 generate
      key_stream(i) <= (k1(i) XOR k2(i) XOR k3(i));
end generate G7;
-- de 58 a 68
GP1: for i in 0 to 10 generate
   GP11: if i < 6 generate
       state1(i + 58) <= stateimpar1(i+26) when sys_clk3 ='0'
else statepar1(i+26);
   end generate GP11;
   GP12: if i > 5 generate
       state1(i+58) <= statepar1(i+26) when sys_clk3 ='0' else
stateimpar1(i+26);--XOR Nodes 281
   end generate GP12;
end generate GP1;
--de 83 a 92
GP2: for i in 0 to 9 generate
   GP21: if i < 5 generate
       state1(i + 83) <= statepar1(i+43) when sys_clk3 ='0' else
stateimpar1(i+43);
   end generate GP21;
   GP22: if i > 4 generate
```

```
       state1(i+83) <= stateimpar1(i+35) when sys_clk3 ='0' else
statepar1(i+35);--XOR Nodes 281
   end generate GP22;
end generate GP2;
-- de 154 a 161
GP4: for i in 0 to 7 generate
   GP41: if i < 3 generate
       state1(154+i) <= stateimpar2(29+i) when sys_clk3 ='0' else
statepar2(29+i);--XOR Nodes 154
   end generate GP41;
   GP42: if i > 2 generate
       state1(154+i) <= statepar2(29+i) when sys_clk3 ='0' else
stateimpar2(29+i);--XOR Nodes 154
   end generate GP42;
end generate GP4;
-- de 163 a 172
GP5: for i in 0 to 9 generate
   GP51: if i < 2 generate
       state1(163+i) <= statepar2(38+i) when sys_clk3 ='0' else
stateimpar2(38+i);--XOR Nodes 165
   end generate GP51;
   GP52: if (i > 1) generate
       state1(163+i) <= stateimpar2(30+i) when sys_clk3 ='0' else
statepar2(30+i);--XOR Nodes 165
   end generate GP52;
end generate GP5;
-- de 173 a 172
GP3: for i in 0 to 3 generate
       state1(173+i) <= statepar2(40+i) when sys_clk3 ='0' else
stateimpar2(40+i);--XOR Nodes 165
end generate GP3;
-- de 235 a 242
GP8: for i in 0 to 7 generate
   GP81: if i < 6 generate
      state1(235+i) <= stateimpar3(26+i) when sys_clk3 ='0' else
statepar3(26+i);--XOR Nodes 235
   end generate GP81;
   GP82: if i > 5 generate
      state1(235+i) <= statepar3(26+i) when sys_clk3 ='0' else
stateimpar3(26+i);--XOR Nodes 235
   end generate GP82;
end generate GP8;
-- de 256 a 263
GP7: for i in 0 to 7 generate
   GP71: if i < 1 generate
      state1(256+i) <= stateimpar3(39+i) when sys_clk3 ='0' else
statepar3(39+i);--XOR Nodes 256
   end generate GP71;
   GP72: if i > 0 generate
      state1(256+i) <= statepar3(39+i) when sys_clk3 ='0' else
stateimpar3(39+i);--XOR Nodes 256
   end generate GP72;
end generate GP7;
-- de 278 a 287
GP9: for i in 0 to 9 generate
```

```
   GP91: if i < 3 generate
      state1(278+i) <= statepar3(53+i) when sys_clk3 ='0' else
stateimpar3(53+i);--XOR Nodes 278
   end generate GP91;
   GP92: if i > 2 generate
      state1(278+i) <= stateimpar3(45+i) when sys_clk3 ='0' else
statepar3(45+i);--XOR Nodes 281
   end generate GP92;
end generate GP9;
key_out <= key_stream(7 downto 0);
end Behavioral;
```

A.10. Trivium 16 bits

```
----------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREx16 - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium 16 bits.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_corex16 is
    Port ( sys_clk : in  STD_LOGIC; --Reloj principal
           resetn :in  STD_LOGIC;
           cntrl : in STD_LOGIC_VECTOR(1 downto 0); --Control Bus
           key  : in  STD_LOGIC_VECTOR(79 downto 0);
-- Clave Secreta
           iv   : in  STD_LOGIC_VECTOR(79 downto 0);
-- Vector inicial
           key_out : out  STD_LOGIC_VECTOR(15 downto 0));
--Secuencia cifrante
end trivium_corex16;
architecture Behavioral of trivium_corex16 is
-- Reg. int. LFSR
SIGNAL state : STD_LOGIC_VECTOR(295 downto 0) := (OTHERS => '0');
```

```
SIGNAL key_stream : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS =>
'0');  --
SIGNAL k1 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');  --
Nodos
SIGNAL k2 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');  --
Nodos
SIGNAL k3 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');  --
SIGNAL t1 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');  --
SIGNAL t2 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');  --
SIGNAL t3 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');  --
SIGNAL key_flip : STD_LOGIC_VECTOR(79 downto 0);
SIGNAL iv_flip : STD_LOGIC_VECTOR(79 downto 0);
-- Conversor 80-bit Big Endian a Little Endian Convert.
--ej 0x0123456789 -> 0x084C2A6E19
function little_endian (b: std_logic_vector) return
std_logic_vector is
       variable result : std_logic_vector(79 downto 0);
   begin
       for i in 0 to 9 loop
          result(((i*8)+7) downto (i*8)) := b(i*8) &
                                            b((i*8) + 1) &
                                            b((i*8) + 2) &
                                            b((i*8) + 3) &
                                            b((i*8) + 4) &
                                            b((i*8) + 5) &
                                            b((i*8) + 6) &
                                            b((i*8) + 7);
       end loop;
       return result;
   end;
begin
--Core LFSR con 3 registros de desplazamiento con realimentación
-- de los nodos (t1,t2,t3).
MAIN_TRIVIUM : process (sys_clk,cntrl,resetn)
begin
   if(resetn = '0') then
     state(287 downto 0) <= (OTHERS => '0');
   elsif(sys_clk'event AND sys_clK='1') then
      if (cntrl = "10") then
          state(92 downto 0) <= "0000000000000" & key_flip;
          state(176 downto 93) <= X"0" & iv_flip;
          state(287 downto 177) <= "111" &
X"00000000000000000000000000";
      elsif ((cntrl = "11") OR (cntrl = "01")) then
          state(92 downto 0) <= state(76 downto 0) & t3(15 downto
0);
          state(176 downto 93) <= state(160 downto 93) & t1(15
downto 0);
          state(287 downto 177) <= state(271 downto 177) & t2(15
downto 0
      end if;
    end if;
end process;
--XOR Nodes5
G1: for i in 0 to 15 generate
```

```
    k1(i) <= state(50+i) XOR state(77+i);
end generate G1;
G2: for i in 0 to 15 generate
      k2(i) <= state(146+i) XOR state(161+i);
end generate G2;
G3: for i in 0 to 15 generate
      k3(i) <= state(227+i) XOR state(272+i);
end generate G3;
G4: for i in 0 to 15 generate
      t1(i) <= k1(i) XOR ((state(75+i) AND state(76+i)) XOR
state(155+i));
end generate G4;
G5: for i in 0 to 15 generate
      t2(i) <= k2(i) XOR ((state(159+i) AND state(160+i)) XOR
state(248+i));
end generate G5;
G6: for i in 0 to 15 generate
      t3(i) <= k3(i) XOR ((state(270+i) AND state(271+i)) XOR
state(53+i));
end generate G6;
G7: for i in 0 to 15 generate
      key_stream(i) <= (k1(i) XOR k2(i) XOR k3(i));
end generate G7;
--Cambia valores de entrada para que las salidas coincidan con
-- los esperados.
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
key_out(15 downto 0) <= key_stream(15 downto 0);
-- MODIFICACION
end Behavioral;
```

A.11. Trivium MPLP 16 bits

```
----------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREx16MPLP - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium MPLP 16 bits.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test
```

```
------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_corex16FPLP is
   Port ( sys_clk : in  std_logic; -- Reloj principal
      resetn :in  std_logiC;
      cntrl : in std_logic_vector(1 downto 0); -- Control Bus
      key  : in  std_logic_vector(79 downto 0);
-- Clave Secreta 80-bit
      iv   : in  std_logic_vector(79 downto 0);
-- Vector inicialización
      key_out : out  std_logic_vector (15 downto 0));
-- Secuencia cifrante
end trivium_corex16FPLP;
 architecture Behavioral of trivium_corex16FPLP is
signal key_flip : std_logic_vector (79 downto 0);
signal iv_flip : std_logic_vector (79 downto 0);
--
signal state1 : std_logic_vector (92 downto 32) := (OTHERS =>
'0');  ---
signal state2 : std_logic_vector (176 downto 125) := (OTHERS =>
'0');  -
signal state3 : std_logic_vector (287 downto 209) := (OTHERS =>
'0'); --
signal s1,s2,s3: std_logic_vector (15 downto 0) := (OTHERS =>
'0');
signal state : std_logic_vector (287 downto 0) := (OTHERS =>
'0');  --
signal statepar1 : std_logic_vector (15 downto 0) := (OTHERS =>
'0');  --
signal stateimpar1 : std_logic_vector (15 downto 0) := (OTHERS =>
'0');
signal statepar2 : std_logic_vector (15 downto 0) := (OTHERS =>
'0');  --   signal stateimpar2 : std_logic_vector (15 downto 0)
:= (OTHERS => '0');
signal statepar3 : std_logic_vector (15 downto 0) := (OTHERS =>
'0');  --
signal stateimpar3 : std_logic_vector (15 downto 0) := (OTHERS =>
'0');
---Salida secuencia cifrante
signal key_stream : std_logic_vector (15 downto 0) := (OTHERS =>
'0');
--Señales realimentación
signal k1 : std_logic_vector (15 downto 0) := (OTHERS => '0');  -
signal k2 : std_logic_vector (15 downto 0) := (OTHERS => '0');  -
signal k3 : std_logic_vector (15 downto 0) := (OTHERS => '0');  -
signal t1 : std_logic_vector (15 downto 0) := (OTHERS => '0');  -
signal t2 : std_logic_vector (15 downto 0) := (OTHERS => '0');  -
signal t3 : std_logic_vector (15 downto 0) := (OTHERS => '0');  -
signal qout,qout2 : STD_LOGIC :='0';   ---Reloj/2
signal sys_clk3  : STD_LOGIC;   ---Reloj/2
-- Conversor 80-bit Big Endian a Little Endian Convert.
```

```
function little_endian (b: std_logic_vector) return
std_logic_vector is -
 variable result : std_logic_vector(79 downto 0);
   begin
      for i in 0 to 9 loop
         result(((i*8)+7) downto (i*8)) :=  b(i*8) & b((i*8)+1) &
b((i*8)+2) & b((i*8)+3) & b((i*8)+4) & b((i*8)+5) & b((i*8)+6) &
b((i*8)+7);
      end loop;
return result;
end;
begin
-- Definición relojes
SYS_CLK3_process :process  (sys_clk,resetn)
   begin
      if(resetn = '0') then
         qout2  <= '0';
      elsif(sys_clk'event AND sys_clk='1') then
         qout2 <= NOT(qout2) ;
      end if;
end process;
sys_clk3 <= NOT(qout2);
--Cambia valores de entrada para que las salidas coincidan con
-- los esperados
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
state(92 downto 0) <= "0000000000000" & key_flip;
state(176 downto 93) <= X"0" & iv_flip;
state(287 downto 177) <= "111" & X"00000000000000000000000000";
SECOND_TRIVIUM_RISE : process (sys_clk3,cntrl,resetn)
   begin
      if(resetn = '0') then
         statepar1  <= (OTHERS => '0');
         statepar2  <= (OTHERS => '0');
         statepar3  <= (OTHERS => '0');
      elsif(sys_clk3'event AND sys_clk3='0') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
         if (cntrl = "10") then  --
-- FIRST PAR, entran 8 bits de una vez
         for i in 0 to 0 loop
            statepar1 (16*i) <= state (i*32);
            statepar1 (16*i+1) <= state (i*32+1);
            statepar1 (16*i+2) <= state (i*32+2);
            statepar1 (16*i+3) <= state (i*32+3);
            statepar1 (16*i+4) <= state (i*32+4);
            statepar1 (16*i+5) <= state (i*32+5);
            statepar1 (16*i+6) <= state (i*32+6);
            statepar1 (16*i+7) <= state (i*32+7);
            statepar1 (16*i+8) <= state (i*32+8);
            statepar1 (16*i+9) <= state (i*32+9);
            statepar1 (16*i+10) <= state (i*32+10);
            statepar1 (16*i+11) <= state (i*32+11);
            statepar1 (16*i+12) <= state (i*32+12);
            statepar1 (16*i+13) <= state (i*32+13);
            statepar1 (16*i+14) <= state (i*32+14);
```

```
            statepar1 (16*i+15) <= state (i*32+15);
         end loop;
         for i in 0 to 0 loop
               statepar2 (16*i) <= state (93+(32*i));
               statepar2 (16*i+1) <= state (93+1+(32*i));
               statepar2 (16*i+2) <= state (93+2+(32*i));
               statepar2 (16*i+3) <= state (93+3+(32*i));
               statepar2 (16*i+4) <= state (93+4+(32*i));
               statepar2 (16*i+5) <= state (93+5+(32*i));
               statepar2 (16*i+6) <= state (93+6+(32*i));
               statepar2 (16*i+7) <= state (93+7+(32*i));
               statepar2 (16*i+8) <= state (93+8+(32*i));
               statepar2 (16*i+9) <= state (93+9+(32*i));
               statepar2 (16*i+10) <= state (93+10+(32*i));
               statepar2 (16*i+11) <= state (93+11+(32*i));
               statepar2 (16*i+12) <= state (93+12+(32*i));
               statepar2 (16*i+13) <= state (93+13+(32*i));
               statepar2 (16*i+14) <= state (93+14+(32*i));
               statepar2 (16*i+15) <= state (93+15+(32*i));
         end loop;
-- THIRD
         for i in 0 to 0 loop
               statepar3 (16*i) <= state (177+(32*i));
               statepar3 (16*i+1) <= state (177+1+(32*i));
               statepar3 (16*i+2) <= state (177+2+(32*i));
               statepar3 (16*i+3) <= state (177+3+(32*i));
               statepar3 (16*i+4) <= state (177+4+(32*i));
               statepar3 (16*i+5) <= state (177+5+(32*i));
               statepar3 (16*i+6) <= state (177+6+(32*i));
               statepar3 (16*i+7) <= state (177+7+(32*i));
               statepar3 (16*i+8) <= state (177+8+(32*i));
               statepar3 (16*i+9) <= state (177+9+(32*i));
               statepar3 (16*i+10) <= state (177+10+(32*i));
               statepar3 (16*i+11) <= state (177+11+(32*i));
               statepar3 (16*i+12) <= state (177+12+(32*i));
               statepar3 (16*i+13) <= state (177+13+(32*i));
               statepar3 (16*i+14) <= state (177+14+(32*i));
               statepar3 (16*i+15) <= state (177+15+(32*i));
         end loop;
         elsif ((cntrl = "11") OR (cntrl = "01")) then
-- registro 16 bits.
            statepar1(15 downto 0) <= t3(15 downto 0);
            statepar2(15 downto 0) <= t1(15 downto 0);
            statepar3(15 downto 0) <= t2(15 downto 0);
         end if;
   end if;
end process;
SECOND_TRIVIUM_FALL : process (sys_clk3,cntrl,resetn)
-- Core LFSR con 3 registros de
-- desplazamiento con realimentación de los nodos (t1,t2,t3).
   begin  --Core must be loaded with key and
      if(resetn = '0') then
         stateimpar1  <= (OTHERS => '0');
         stateimpar2  <= (OTHERS => '0');
         stateimpar3  <= (OTHERS => '0');
```

```
      elsif(sys_clk3'event AND sys_clk3='1') then
         if (cntrl = "10") then  --
-- FIRST
            for i in 0 to 0 loop
               stateimpar1 (16*i) <= state ((i*32)+16);
               stateimpar1 (16*i+1) <= state ((i*32)+16+1);
               stateimpar1 (16*i+2) <= state ((i*32)+16+2);
               stateimpar1 (16*i+3) <= state ((i*32)+16+3);
               stateimpar1 (16*i+4) <= state ((i*32)+16+4);
               stateimpar1 (16*i+5) <= state ((i*32)+16+5);
               stateimpar1 (16*i+6) <= state ((i*32)+16+6);
               stateimpar1 (16*i+7) <= state ((i*32)+16+7);
               stateimpar1 (16*i+8) <= state ((i*32)+16+8);
               stateimpar1 (16*i+9) <= state ((i*32)+16+9);
               stateimpar1 (16*i+10) <= state ((i*32)+16+10);
               stateimpar1 (16*i+11) <= state ((i*32)+16+11);
               stateimpar1 (16*i+12) <= state ((i*32)+16+12);
               stateimpar1 (16*i+13) <= state ((i*32)+16+13);
               stateimpar1 (16*i+14) <= state ((i*32)+16+14);
               stateimpar1 (16*i+15) <= state ((i*32)+16+15);
            end loop;
-- SECOND
            for i in 0 to 0 loop
               stateimpar2 (16*i) <= state (109+(32*i));
               stateimpar2 (16*i+1) <= state (109+1+(32*i));
               stateimpar2 (16*i+2) <= state (109+2+(32*i));
               stateimpar2 (16*i+3) <= state (109+3+(32*i));
               stateimpar2 (16*i+4) <= state (109+4+(32*i));
               stateimpar2 (16*i+5) <= state (109+5+(32*i));
               stateimpar2 (16*i+6) <= state (109+6+(32*i));
               stateimpar2 (16*i+7) <= state (109+7+(32*i));
               stateimpar2 (16*i+8) <= state (109+8+(32*i));
               stateimpar2 (16*i+9) <= state (109+9+(32*i));
               stateimpar2 (16*i+10) <= state (109+10+(32*i));
               stateimpar2 (16*i+11) <= state (109+11+(32*i));
               stateimpar2 (16*i+12) <= state (109+12+(32*i));
               stateimpar2 (16*i+13) <= state (109+13+(32*i));
               stateimpar2 (16*i+14) <= state (109+14+(32*i));
               stateimpar2 (16*i+15) <= state (109+15+(32*i));
            end loop;
-- THIRD PAR
            for i in 0 to 0 loop
               stateimpar3 (16*i) <= state (193+(32*i));
               stateimpar3 (16*i+1) <= state (193+1+(32*i));
               stateimpar3 (16*i+2) <= state (193+2+(32*i));
               stateimpar3 (16*i+3) <= state (193+3+(32*i));
               stateimpar3 (16*i+4) <= state (193+4+(32*i));
               stateimpar3 (16*i+5) <= state (193+5+(32*i));
               stateimpar3 (16*i+6) <= state (193+6+(32*i));
               stateimpar3 (16*i+7) <= state (193+7+(32*i));
               stateimpar3 (16*i+8) <= state (193+8+(32*i));
               stateimpar3 (16*i+9) <= state (193+9+(32*i));
               stateimpar3 (16*i+10) <= state (193+10+(32*i));
               stateimpar3 (16*i+11) <= state (193+11+(32*i));
               stateimpar3 (16*i+12) <= state (193+12+(32*i));
```

```
               stateimpar3 (16*i+13) <= state (193+13+(32*i));
               stateimpar3 (16*i+14) <= state (193+14+(32*i));
               stateimpar3 (16*i+15) <= state (193+15+(32*i));
            end loop;
         elsif ((cntrl = "11") OR (cntrl = "01")) then
            stateimpar1(15 downto 0) <= t3(15 downto 0);
            stateimpar2(15 downto 0) <= t1(15 downto 0);
            stateimpar3(15 downto 0) <= t2(15 downto 0);
         end if;
      end if;
end process;
MAIN_TRIVIUM : process (sys_clk,cntrl,resetn)
-- Core LFSR con 3 registros de
-- desplazamiento con realimentación de los nodos (t1,t2,t3).
begin
   if(resetn = '0') then
       state1  <= (OTHERS => '0');
       state2  <= (OTHERS => '0');
       state3  <= (OTHERS => '0');
    elsif(sys_clk'event AND sys_clk='1') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
       if (cntrl = "10") then
          state1 <= state(92 downto 32) ;
          state2 <= state(176 downto 125) ;
          state3 <= state(287 downto 209) ;
        elsif ((cntrl = "11") or (cntrl = "01")) then
          state1(92 downto 32) <= state1(76 downto 32) & s1;
          state2(176 downto 125) <= state2(160 downto 125) & s2;
        end if;
    end if;
end process;
-- MODIFICACION
s1 <= stateimpar1(15 downto 0) when sys_clk3 ='0' else
statepar1(15 downto 0);--Nodo salida shift reg 1
s2 <= stateimpar2(15 downto 0) when sys_clk3 ='0' else
statepar2(15 downto 0);--Nodo salida shift reg 1
s3 <= stateimpar3(15 downto 0) when sys_clk3 ='0' else
statepar3(15 downto 0);--Nodo salida shift reg 1
G1: for i in 0 to 15 generate
    k1(i) <= state1(50+i) XOR state1(77+i);
end generate G1;
G2: for i in 0 to 15 generate
      k2(i) <= state2(146+i) XOR state2(161+i);
end generate G2;
G3: for i in 0 to 15 generate
      k3(i) <= state3(227+i) XOR state3(272+i);
end generate G3;
G4: for i in 0 to 15 generate
      t1(i) <= k1(i) XOR ((state1(75+i) AND state1(76+i)) XOR
state2(155+i));
end generate G4;
G5: for i in 0 to 15 generate
      t2(i) <= k2(i) XOR ((state2(159+i) AND state2(160+i)) XOR
state3(248+i));
end generate G5;
```

```
G6: for i in 0 to 15 generate
      t3(i) <= k3(i) XOR ((state3(270+i) AND state3(271+i)) XOR
state1(53+i));
end generate G6;
G7: for i in 0 to 15 generate
      key_stream(i) <= (k1(i) XOR k2(i) XOR k3(i));
end generate G7;
key_out <= key_stream(15 downto 0);
end Behavioral;
```

A.12. Trivium FPLP 16 bits

```
----------------------------------------------------------------
-- Compañía: IMSE-cnm (CSIC-Universidad de Sevilla)
-- Diseñador: JMMG
-- Fecha:    2016
-- Nombre Diseño: Trivium
-- Nombre Módulo: TRIVIUM_COREx16FPLP - Behavioral
-- Nombre Proyecto: TESIS
-- Dispositivo a implementar: ASIC
-- Versión Herramienta: Synopsys
-- Descripción: Cifrador Trivium FPLP 16 bits.
-- Especificación:
-- http://www.ecrypt.eu.org/stream/ciphers/trivium/
-- Vectores de Test:
-- http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/
--/*checkout*/ecrypt/trunk/submissions/trivium/
--/unverified.test-vectors?rev=210
-- Revision: v1.0
-- Comentarios adicionales: Incluye bits de control para la carga
-- de key/IV.
-- Incluye función de conversión de los vectores de test
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity trivium_corex16FPLP is
   Port ( sys_clk : in  std_logic; --Reloj principal
      resetn :in  std_logiC;
      cntrl : in std_logic_vector(1 downto 0); --Control Bus
      key  : in  std_logic_vector(79 downto 0); );
--Clave Secreta 80-bit.
      iv   : in  std_logic_vector(79 downto 0);
--Vector inicialización
      key_out : out  std_logic_vector (15 downto 0));
--Secuencia cifrante
end trivium_corex16FPLP;
architecture Behavioral of trivium_corex16FPLP is
signal key_flip : STD_LOGIC_VECTOR(79 downto 0);
signal iv_flip : STD_LOGIC_VECTOR(79 downto 0);
---Regitro interno LFSR
```

```
signal state1 : STD_LOGIC_VECTOR(304 downto 0) := (OTHERS =>
'0');
---Regitro interno LFSR
signal state : STD_LOGIC_VECTOR(304 downto 0) := (OTHERS => '0');
---Regitro PAR-1 interno LFSR
signal statepar1 : STD_LOGIC_VECTOR(47 downto 0) := (OTHERS =>
'0');
---Regitro IMPAR-1 interno LFSR
signal stateimpar1 : STD_LOGIC_VECTOR(47 downto 0) := (OTHERS =>
'0');
---Regitro PAR-2 interno LFSR
signal statepar2 : STD_LOGIC_VECTOR(47 downto 0) := (OTHERS =>
'0');
---Regitro IMPAR-2 interno LFSR
signal stateimpar2 : STD_LOGIC_VECTOR(47 downto 0) := (OTHERS =>
'0');
---Regitro PAR-3 interno LFSR
signal statepar3 : STD_LOGIC_VECTOR(63 downto 0) := (OTHERS =>
'0');
---Regitro IMPAR-3 interno LFSR
signal stateimpar3 : STD_LOGIC_VECTOR(63 downto 0) := (OTHERS =>
'0');
---Salida secuencia cifrante
signal key_stream : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS =>
'0');
--Señales realimentación
signal k1 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');
signal k2 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');
signal k3 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');
signal t1 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');
signal t2 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');
signal t3 : STD_LOGIC_VECTOR(15 downto 0) := (OTHERS => '0');

signal qout,qout2 : STD_LOGIC :='0';   --- Reloj/2
signal sys_clk3  : STD_LOGIC;   --- Reloj/2
-- Conversor 80-bit Big Endian a Little Endian Convert.
function little_endian (b: std_logic_vector) return
std_logic_vector is
   variable result : std_logic_vector(79 downto 0);
--ex 0x0123456789 -> 0x084C2A6E19
   begin
      for i in 0 to 9 loop
         result(((i*8)+7) downto (i*8)) :=  b(i*8) & b((i*8)+1) &
b((i*8)+2) & b((i*8)+3) & b((i*8)+4) & b((i*8)+5) & b((i*8)+6) &
b((i*8)+7);
      end loop;
return result;
end;
begin
-- Definición relojes
SYS_CLK3_process :process  (sys_clk,resetn)
   begin
      if(resetn = '0') then
         qout2  <= '0';
      elsif(sys_clk'event AND sys_clk='1') then
```

```
         qout2 <= NOT(qout2) ;
      end if;
end process;
sys_clk3 <= NOT(qout2);
--Cambia valores de entrada para que las salidas coincidan con
-- los esperados
key_flip <= little_endian(key);
iv_flip  <= little_endian(iv);
state(92 downto 0) <= "0000000000000" & key_flip;
state(176 downto 93) <= X"0" & iv_flip;
state(287 downto 177) <= "111" & X"00000000000000000000000000";
SECOND_TRIVIUM_RISE : process (sys_clk3,cntrl,resetn)
-- Core LFSR con 3 registros de
-- desplazamiento con realimentación de los nodos (t1,t2,t3).
begin --Core must be loaded with key and
   if(resetn = '0') then
      statepar1  <= (OTHERS => '0');
      statepar2  <= (OTHERS => '0');
      statepar3  <= (OTHERS => '0');
   elsif(sys_clk3'event AND sys_clk3='0') then
-- PRIMERO DESPLAZO POR IMPAR!!!!
     if (cntrl = "10") then
-- FIRST PAR, entran 8 bits de una vez
       for i in 0 to 2 loop
          statepar1 (16*i) <= state (i*32);
          statepar1 (16*i+1) <= state (i*32+1);
          statepar1 (16*i+2) <= state (i*32+2);
          statepar1 (16*i+3) <= state (i*32+3);
          statepar1 (16*i+4) <= state (i*32+4);
          statepar1 (16*i+5) <= state (i*32+5);
          statepar1 (16*i+6) <= state (i*32+6);
          statepar1 (16*i+7) <= state (i*32+7);
          statepar1 (16*i+8) <= state (i*32+8);
          statepar1 (16*i+9) <= state (i*32+9);
          statepar1 (16*i+10) <= state (i*32+10);
          statepar1 (16*i+11) <= state (i*32+11);
          statepar1 (16*i+12) <= state (i*32+12);
          statepar1 (16*i+13) <= state (i*32+13);
          statepar1 (16*i+14) <= state (i*32+14);
          statepar1 (16*i+15) <= state (i*32+15);
     end loop;
       for i in 0 to 2 loop
          statepar2 (16*i) <= state (93+(32*i));
          statepar2 (16*i+1) <= state (93+1+(32*i));
          statepar2 (16*i+2) <= state (93+2+(32*i));
          statepar2 (16*i+3) <= state (93+3+(32*i));
          statepar2 (16*i+4) <= state (93+4+(32*i));
          statepar2 (16*i+5) <= state (93+5+(32*i));
          statepar2 (16*i+6) <= state (93+6+(32*i));
          statepar2 (16*i+7) <= state (93+7+(32*i));
          statepar2 (16*i+8) <= state (93+8+(32*i));
          statepar2 (16*i+9) <= state (93+9+(32*i));
          statepar2 (16*i+10) <= state (93+10+(32*i));
          statepar2 (16*i+11) <= state (93+11+(32*i));
          statepar2 (16*i+12) <= state (93+12+(32*i));
```

```
            statepar2 (16*i+13) <= state (93+13+(32*i));
            statepar2 (16*i+14) <= state (93+14+(32*i));
            statepar2 (16*i+15) <= state (93+15+(32*i));
         end loop;
-- THIRD
         for i in 0 to 3 loop
            statepar3 (16*i) <= state (177+(32*i));
            statepar3 (16*i+1) <= state (177+1+(32*i));
            statepar3 (16*i+2) <= state (177+2+(32*i));
            statepar3 (16*i+3) <= state (177+3+(32*i));
            statepar3 (16*i+4) <= state (177+4+(32*i));
            statepar3 (16*i+5) <= state (177+5+(32*i));
            statepar3 (16*i+6) <= state (177+6+(32*i));
            statepar3 (16*i+7) <= state (177+7+(32*i));
            statepar3 (16*i+8) <= state (177+8+(32*i));
            statepar3 (16*i+9) <= state (177+9+(32*i));
            statepar3 (16*i+10) <= state (177+10+(32*i));
            statepar3 (16*i+11) <= state (177+11+(32*i));
            statepar3 (16*i+12) <= state (177+12+(32*i));
            statepar3 (16*i+13) <= state (177+13+(32*i));
            statepar3 (16*i+14) <= state (177+14+(32*i));
            statepar3 (16*i+15) <= state (177+15+(32*i));
         end loop;
      elsif ((cntrl = "11") OR (cntrl = "01")) then
-- registro 48 bits. SE AÑADE UN BIT MAS AL AGRUPAR EN 8 BITS
          statepar1(47 downto 0) <= statepar1(31 downto 0) & t3(15
downto 0);
-- registro 48 bits
         statepar2(47 downto 0) <= statepar2(31 downto 0) & t1(15
downto 0);
-- registro 64 bits
         statepar3(63 downto 0) <= statepar3(47 downto 0)  & t2(15
downto 0);
-- registro 64 bits
          end if;
   end if;
end process;
SECOND_TRIVIUM_FALL : PROCESS (sys_clk3,cntrl,resetn)
-- Core LFSR con 3 registros de
-- desplazamiento con realimentación de los nodos (t1,t2,t3).
begin  --Core must be loaded with key and
   if(resetn = '0') then
      stateimpar1  <= (OTHERS => '0');
      stateimpar2  <= (OTHERS => '0');
      stateimpar3  <= (OTHERS => '0');
   elsif(sys_clk3'event AND sys_clk3='1') then
      if (cntrl = "10") then  ---Intial state setup based on spec
-- FIRST
         for i in 0 to 2 loop
            stateimpar1 (16*i) <= state ((i*32)+16);
            stateimpar1 (16*i+1) <= state ((i*32)+16+1);
            stateimpar1 (16*i+2) <= state ((i*32)+16+2);
            stateimpar1 (16*i+3) <= state ((i*32)+16+3);
            stateimpar1 (16*i+4) <= state ((i*32)+16+4);
            stateimpar1 (16*i+5) <= state ((i*32)+16+5);
```

```
            stateimpar1 (16*i+6) <= state ((i*32)+16+6);
            stateimpar1 (16*i+7) <= state ((i*32)+16+7);
            stateimpar1 (16*i+8) <= state ((i*32)+16+8);
            stateimpar1 (16*i+9) <= state ((i*32)+16+9);
            stateimpar1 (16*i+10) <= state ((i*32)+16+10);
            stateimpar1 (16*i+11) <= state ((i*32)+16+11);
            stateimpar1 (16*i+12) <= state ((i*32)+16+12);
            stateimpar1 (16*i+13) <= state ((i*32)+16+13);
            stateimpar1 (16*i+14) <= state ((i*32)+16+14);
            stateimpar1 (16*i+15) <= state ((i*32)+16+15);
          end loop;
-- SECOND
          for i in 0 to 2 loop
            stateimpar2 (16*i) <= state (109+(32*i));
            stateimpar2 (16*i+1) <= state (109+1+(32*i));
            stateimpar2 (16*i+2) <= state (109+2+(32*i));
            stateimpar2 (16*i+3) <= state (109+3+(32*i));
            stateimpar2 (16*i+4) <= state (109+4+(32*i));
            stateimpar2 (16*i+5) <= state (109+5+(32*i));
            stateimpar2 (16*i+6) <= state (109+6+(32*i));
            stateimpar2 (16*i+7) <= state (109+7+(32*i));
            stateimpar2 (16*i+8) <= state (109+8+(32*i));
            stateimpar2 (16*i+9) <= state (109+9+(32*i));
            stateimpar2 (16*i+10) <= state (109+10+(32*i));
            stateimpar2 (16*i+11) <= state (109+11+(32*i));
            stateimpar2 (16*i+12) <= state (109+12+(32*i));
            stateimpar2 (16*i+13) <= state (109+13+(32*i));
            stateimpar2 (16*i+14) <= state (109+14+(32*i));
            stateimpar2 (16*i+15) <= state (109+15+(32*i));
          end loop;
-- THIRD PAR
          for i in 0 to 3 loop
            stateimpar3 (16*i) <= state (193+(32*i));
            stateimpar3 (16*i+1) <= state (193+1+(32*i));
            stateimpar3 (16*i+2) <= state (193+2+(32*i));
            stateimpar3 (16*i+3) <= state (193+3+(32*i));
            stateimpar3 (16*i+4) <= state (193+4+(32*i));
            stateimpar3 (16*i+5) <= state (193+5+(32*i));
            stateimpar3 (16*i+6) <= state (193+6+(32*i));
            stateimpar3 (16*i+7) <= state (193+7+(32*i));
            stateimpar3 (16*i+8) <= state (193+8+(32*i));
            stateimpar3 (16*i+9) <= state (193+9+(32*i));
            stateimpar3 (16*i+10) <= state (193+10+(32*i));
            stateimpar3 (16*i+11) <= state (193+11+(32*i));
            stateimpar3 (16*i+12) <= state (193+12+(32*i));
            stateimpar3 (16*i+13) <= state (193+13+(32*i));
            stateimpar3 (16*i+14) <= state (193+14+(32*i));
            stateimpar3 (16*i+15) <= state (193+15+(32*i));
        end loop;
      elsif ((cntrl = "11") OR (cntrl = "01")) then
       stateimpar1(47 downto 0) <= stateimpar1(31 downto 0) & t3(15
downto 0);
       stateimpar2(47 downto 0) <= stateimpar2(31 downto 0) & t1(15
downto 0);
```

```
     stateimpar3(63 downto 0) <= stateimpar3(47 downto 0) & t2(15
downto 0);
         end if;
       end if;
end process;
-- MODIFICACION
G1: for i in 0 to 15 generate
    k1(i) <= state1(50+i) XOR state1(77+i);
end generate G1;
G2: for i in 0 to 15 generate
      k2(i) <= state1(146+i) XOR state1(161+i);
end generate G2;
G3: for i in 0 to 15 generate
      k3(i) <= state1(227+i) XOR state1(272+i);
end generate G3;
G4: for i in 0 to 15 generate
      t1(i) <= k1(i) XOR ((state1(75+i) AND state1(76+i)) XOR
state1(155+i));
end generate G4;
G5: for i in 0 to 15 generate
      t2(i) <= k2(i) XOR ((state1(159+i) AND state1(160+i)) XOR
state1(248+i));
end generate G5;
G6: for i in 0 to 15 generate
      t3(i) <= k3(i) XOR ((state1(270+i) AND state1(271+i)) XOR
state1(53+i));
end generate G6;
G7: for i in 0 to 15 generate
      key_stream(i) <= (k1(i) XOR k2(i) XOR k3(i));
end generate G7;
-- de 50 a 68
GP1: for i in 0 to 18 generate
   GP11: if i < 14 generate -- de 50 a 63
       state1(i + 50) <= stateimpar1(i+18) when sys_clk3 ='0'
else statepar1(i+18);
   end generate GP11;
   GP12: if i > 13 generate -- de 64 a 68
       state1(i+50) <= statepar1(i+18) when sys_clk3 ='0' else
stateimpar1(i+18);--XOR Nodes 281
   end generate GP12;
end generate GP1;
--de 75 a 92
GP2: for i in 0 to 17 generate
   GP21: if i < 5 generate
       state1(i + 75) <= statepar1(i+43) when sys_clk3 ='0' else
stateimpar1(i+43);
   end generate GP21;
   GP22: if i > 4 generate
       state1(i+75) <= stateimpar1(i+27) when sys_clk3 ='0' else
statepar1(i+27);--XOR Nodes 281
   end generate GP22;
end generate GP2;
-- de 146 a 172
GP4: for i in 0 to 26 generate
   GP41: if i < 11 generate
```

```
       state1(146+i) <= stateimpar2(21+i) when sys_clk3 ='0' else
statepar2(21+i);--XOR Nodes 154
   end generate GP41;
-- de 157 a 172
   GP42: if i > 10 generate
       state1(146+i) <= statepar2(21+i) when sys_clk3 ='0' else
stateimpar2(21+i);--XOR Nodes 154
   end generate GP42;
end generate GP4;
-- de 173 a 176
GP5: for i in 0 to 3 generate
       state1(173+i) <= stateimpar2(32+i) when sys_clk3 ='0' else
statepar2(32+i);--XOR Nodes 165
end generate GP5;
-- de 227 a 242
GP8: for i in 0 to 15 generate
   GP81: if i < 14 generate
      state1(227+i) <= stateimpar3(18+i) when sys_clk3 ='0' else
statepar3(18+i);--XOR Nodes 235
   end generate GP81;
   GP82: if i > 13 generate
      state1(227+i) <= statepar3(18+i) when sys_clk3 ='0' else
stateimpar3(18+i);--XOR Nodes 235
   end generate GP82;
end generate GP8;
-- de 248 a 263
GP7: for i in 0 to 15 generate
   GP71: if i < 9 generate
      state1(248+i) <= statepar3(39+i) when sys_clk3 ='0' else
stateimpar3(39+i);--XOR Nodes 256
   end generate GP71;
   GP72: if i > 8 generate
      state1(248+i) <= stateimpar3(23+i) when sys_clk3 ='0' else
statepar3(23+i);--XOR Nodes 256
   end generate GP72;
end generate GP7;
-- de 270 a 288
GP9: for i in 0 to 18 generate
   GP91: if i < 3 generate
      state1(270+i) <= stateimpar3(45+i) when sys_clk3 ='0' else
statepar3(45+i);--XOR Nodes 278
   end generate GP91;
   GP92: if i > 2 generate
      state1(270+i) <= statepar3(45+i) when sys_clk3 ='0' else
stateimpar3(45+i);--XOR Nodes 281
   end generate GP92;
end generate GP9;
key_out <= key_stream(15 downto 0);
end Behavioral;
```

APÉNDICE B. SCRIPTS Y CÓDIGOS *VHDL*

A continuación se presentan algunos de los *scripts* de síntesis y de implementación del circuito integrado así como ficheros de simulación y otros empleados en la fase de *front-end* y *back-end*.

B.1. *Script* RTL para síntesis en *Synopsys*

```
read_file -format vhdl
{/mnt/cnm/jmiguel/CJESUS/DESIGN_VISION/SRC/trivium_core.vhd}
analyze -library WORK -format vhdl
{/mnt/cnm/jmiguel/CJESUS/DESIGN_VISION/SRC/trivium_core.vhd}
elaborate TRIVIUM_CORE -architecture BEHAVIORAL -library WORK
check_design -multiple_designs
write -hierarchy -format ddc -output
/mnt/cnm/jmiguel/CJESUS/DESIGN_VISION/SYN/DDC/trivium_core.ddc
create_clock -name "clk" -period 40 -waveform { 0 20  } { SYS_CLK
}
set_dont_touch_network  [ find clock clk ]
set_dont_touch_network RESETN
set_ideal_network RESETN
set_max_area 10000
set_operating_conditions -library fsa0a_c_sc_tc TCCOM
set_wire_load_model -name G5K -library fsa0a_c_sc_tc
set_wire_load_mode top
compile -map_effort medium -area_effort medium -incremental_mapping
report_qor > ./SYN/RPT/report_qor_default.txt
report_qor
report_constraint -all_violators
report_timing > ./SYN/RPT/report_timing.txt
report_timing
report_area > ./SYN/RPT/report_area_default.txt
report_area
write -hierarchy -format ddc -output
/mnt/cnm/jmiguel/CJESUS/DESIGN_VISION/SYN/DDC/trivium_core_compiled
.ddc
write -hierarchy -format verilog -output
/mnt/cnm/jmiguel/CJESUS/DESIGN_VISION/SYN/NETLIST/trivium_core_comp
iled.v
write -hierarchy -format vhdl -output
```

```
/mnt/cnm/jmiguel/CJESUS/DESIGN_VISION/SYN/NETLIST/trivium_core_comp
iled.vhd
write_sdf ./SYN/SCR/trivium_core.sdf
write_sdc ./SYN/SCR/trivium_core.sdc
report_power -analysis_effort low
report_clock -nosplit
```

B.2. *Testbench* utilizado en simulaciones *ModelSim*

```
--------------------------------------------------------------
-- Company:
-- Engineer:
-- Create Date:   09:23:39 09/18/20166
-- Design Name:
-- Module Name:
-- Project Name:  TRIVIUM
-- Target Device:
-- Tool versions:
-- Description:
-- VHDL Test Bench Created by ISE for module: TRIVIUM_CORE
-- Dependencies:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- Notes:
-- simulation model.
--Set 6, vector#  3:
-- key = 0F62B5085BAE0154A7FA
-- IV = 288FF65DC42B92F960C7
-- stream[0..63] =          A4386C6D7624983FEA8DBE7314E5FE1F
--                          9D102004C2CEC99AC3BFBF003A66433F
--                          3089A98FAD8512C49D7AABC0639F90C5
 --                         FFED06F9D35AA8C86630E76A838E26D7
-- stream[65472..65535] =   04BB52CDF852E04B178FE3B07AF57EC1
--                          06F3180B9B0D59B2192D42BCC35CEF68
--                          96555D57316FF9153C359A8C43EF14CF
--                          7BE1F94D57A52669181D183DD5A4137F
-- stream[65536..65599] =   613009063D291C419D0194D59ADED624
--                          9D9365DAE8D6A62864CF649F5842A214
--                          57BFAD03153DB891E63AC9A859BB9151
--                          1C475A8BD44756480FFBF14AA766B443
-- stream[131008..131071] = CB18518E27F7F95A5207AE008C760F33
--                          C26947E5231847AD32A5ADC1AC74DF45
--                          9526B62A2CD6956D14D3F48677AC338B
--                          13CD7B7A1B3A0C834E64AC03307F8830
-- xor-digest =             88353FC92945C5AF3C04CBF04D467981
--                          3A4E87D9239097CA8CB22CE02C2BF352
--                          DFB5134F17A1AD32684F35C6ADCC560F
--                            AA7AE9BB19F8D8DA96D89C648C2E48C8
--------------------------------------------------------------------
LIBRARY ieee;
use IEEE.std_logic_1164.all;
```

```
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;
ENTITY TRIVIUM_CORE_TB IS
END TRIVIUM_CORE_TB;
ARCHITECTURE behavior OF TRIVIUM_CORE_TB IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT trivium_core_mlp
PORT(
    SYS_CLK : IN  std_logic;
    RESETN : IN  std_logic;
    CNTRL : IN std_logic_vector(1 downto 0);
    KEY : IN  std_logic_vector(79 downto 0);
    IV : IN  std_logic_vector(79 downto 0);
    KEY_OUT_N : OUT  std_logic
        );
    END COMPONENT;
COMPONENT trivium_core
PORT(
    SYS_CLK : IN  std_logic;
    RESETN : IN  std_logic;
    CNTRL : IN std_logic_vector(1 downto 0);
    KEY : IN  std_logic_vector(79 downto 0);
    IV : IN  std_logic_vector(79 downto 0);
    KEY_OUT : OUT  std_logic
        );
    END COMPONENT;
--Inputs
   signal SYS_CLK : std_logic := '0';
   signal RESETN : std_logic := '0';
   signal CNTRL : std_logic_vector(1 downto 0) := "00";
   signal KEY : std_logic_vector(79 downto 0) := (others => '0');
   signal IV : std_logic_vector(79 downto 0) := (others => '0');
--Outputs
   signal KEY_OUT_N : std_logic;
   signal KEY_OUT : std_logic;
   -- Clock period definitions
   constant SYS_CLK_period : time := 40 ns;

   signal cycle_cnt : std_logic_vector(16 downto 0) := (OTHERS =>
'0');
   signal correct_output : std_logic_vector(127 downto 0) :=
(OTHERS => '0');
   signal error_N : std_logic := '0';
   signal error : std_logic := '0';
   signal hab_error : std_logic := '0';
   signal result_N : std_logic_vector(127 downto 0) := (OTHERS =>
'0');
   signal result : std_logic_vector(127 downto 0) := (OTHERS =>
'0');
function littler_endian (b: std_logic_vector(127 downto 0)) return
std_logic_vector is
```

```
variable result : std_logic_vector(127 downto 0);
begin
        for i in 0 to 15 loop
                result(((i*8)+7) downto (i*8)) := b(i*8) &
                                                  b((i*8) + 1) &
                                                  b((i*8) + 2) &
                                                  b((i*8) + 3) &
                                                  b((i*8) + 4) &
                                                  b((i*8) + 5) &
                                                  b((i*8) + 6) &
                                                  b((i*8) + 7);
       end loop;
   return result;
end;
begin
-- Instantiate the Unit Under Test (UUT)
   uut_or: trivium_core PORT MAP (
          SYS_CLK => SYS_CLK,
          RESETN => RESETN,
          CNTRL => CNTRL,
          KEY => KEY,
          IV => IV,
           KEY_OUT => KEY_OUT
        );
   uut: trivium_core_mlp PORT MAP (
          SYS_CLK => SYS_CLK,
          RESETN => RESETN,
          CNTRL => CNTRL,
          KEY => KEY,
          IV => IV,
          KEY_OUT_N => KEY_OUT_N
        );
   -- Clock process definitions
   SYS_CLK_process :process
   begin
      SYS_CLK <= '0';
      wait for SYS_CLK_period/2;
      SYS_CLK <= '1';
      wait for SYS_CLK_period/2;
   end process;
   -- Stimulus process
   stim_proc: process
   begin
      wait for 15 ns;
      RESETN <= '1';
-- hold reset state for 100 ns.
      wait for 100 ns;
      KEY <= X"0F62B5085BAE0154A7FA";
      IV  <= X"288FF65DC42B92F960C7";
      CNTRL <= "10";
      hab_error <= '0';
      wait for SYS_CLK_period*2; -- cambio x1-> x2
      --wait for SYS_CLK_period; -- cambio x1-> x2
      CNTRL <= "11";
      wait for SYS_CLK_period*1152;
```

```
      CNTRL <= "01";
      wait for SYS_CLK_period*126;
      wait for SYS_CLK_period/2;
      correct_output <=
littler_endian(X"A4386C6D7624983FEA8DBE7314E5FE1F");
       hab_error <= '1';
      wait for SYS_CLK_period;
      correct_output <= X"00000000000000000000000000000000";
      hab_error <= '0';
      wait for SYS_CLK_period*126;
      wait for SYS_CLK_period/2;
-- correct_output <=
littler_endian(X"9D102004C2CEC99AC3BFBF003A66433F");
     correct_output <=
littler_endian(X"0D102004C2CEC99AC3BFBF003A66433F");
      hab_error <= '1';
      wait for SYS_CLK_period;
      correct_output <= X"00000000000000000000000000000000";
      hab_error <= '0';
      wait for SYS_CLK_period*127;
      wait for SYS_CLK_period/2;
      correct_output <=
littler_endian(X"3089A98FAD8512C49D7AABC0639F90C5");
      hab_error <= '1';
      wait for SYS_CLK_period;
      hab_error <= '0';
      correct_output <= X"00000000000000000000000000000000";
      --wait for SYS_CLK_period*127;
      --correct_output <=
      --littler_endian(X"3089A98FAD8512C49D7AABC0639F90C5");
      -- wait for SYS_CLK_period;
      -- correct_output <= X"00000000000000000000000000000000";
      -- insert stimulus here
      wait;
   end process;
   ---Result Shift Register
   res_shift_reg : process
        begin
          wait until ((CNTRL = "11") OR (CNTRL = "01"));
 --wait for 300 ns;    --  150  ns
          loop
          result_N <= result_N(126 downto 0) & KEY_OUT_N;
          result <= result(126 downto 0) & KEY_OUT;
          cycle_cnt <= cycle_cnt + 1;
          wait for SYS_CLK_period;
        end loop;
        END PROCESS;
---Output error
   output_error_N : process (SYS_CLK)
        begin
         IF(SYS_CLK'event AND SYS_CLK='0') then
           if hab_error = '1' then
              if correct_output = result then
                 error <= '0';
              else
```

```
                    error <= '1';
                  end if;
                  if correct_output = result_n then
                    error_N <= '0';
                  else
                    error_N <= '1';
                  end if;
              end if;
        end if;
    end process;
end;
```

B.3. *Script* para el cálculo de consumo en *Synopsys*

```
set power_preserve_rtl_hier_names "true"
read_file -format vhdl
{/mnt/cnm/jmiguel/CJESUS/AMS035_DESIGNVISION/SRC/trivium_core.vhd}
analyze -library work -format vhdl
{/mnt/cnm/jmiguel/CJESUS/AMS035_DESIGNVISION/SRC/trivium_core.vhd}
elaborate TRIVIUM_CORE -architecture BEHAVIORAL -library work
set_fix_multiple_port_nets -feedthroughs
uniquify
check_design -multiple_designs
write -hierarchy -format ddc -output
/mnt/cnm/jmiguel/CJESUS/AMS035_DESIGNVISION/SYN/DDC/trivium_core.dd
c
create_clock -name "clk" -period 40 -waveform { 0 20 } { sys_clk  }
set_dont_touch_network  [ find clock clk ]
set_dont_touch_network resetn
set_ideal_network resetn
set_max_area 10000
set_operating_conditions -library c35_CORELIB TYPICAL
set_wire_load_model -name 10k -library c35_CORELIB
compile -map_effort medium -area_effort medium -incremental_mapping
compile_ultra
report_qor > ./SYN/RPT/report_qor_default.txt
report_qor
report_constraint -all_violators
report_timing > ./SYN/RPT/report_timing.txt
report_timing
report_area > ./SYN/RPT/report_area_default.txt
report_area
report_cell > ./SYN/RPT/report_cell_default.txt
report_cell
report_reference > ./SYN/RPT/report_reference_default.txt
report_reference
# change_names -rules vhdl -hierarchy
write -hierarchy -format ddc -output
/mnt/cnm/jmiguel/CJESUS/AMS035_DESIGNVISION/SYN/DDC/trivium_core_co
mpiled.ddc
write -hierarchy -format vhdl -output
/mnt/cnm/jmiguel/CJESUS/AMS035_DESIGNVISION/SYN/NETLIST/trivium_cor
e_compiled.vhd
write -hierarchy -format verilog -output
```

```
/mnt/cnm/jmiguel/CJESUS/AMS035_DESIGNVISION/SYN/NETLIST/trivium_cor
e_compiled.v
write_sdf ./SYN/SCR/trivium_core.sdf
write_sdc ./SYN/SCR/trivium_core.sdc
report_power -analysis_effort low
report_power > ./SYN/RPT/report_power_default.txt
```

B.4. Código VHDL del Bloque deTriviums

```
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity bloque_triviums is
    Port ( clk     :in STD_LOGIC; --System or User clock
           clk2     :in STD_LOGIC; --System or User clock
           resetn   :in STD_LOGIC;
           cntrl    :in STD_LOGIC_VECTOR(1 downto 0);
           serie    :in STD_LOGIC;
           load1    :in STD_LOGIC;  -- Load Key and IV
           slct     :in STD_LOGIC_VECTOR(2 downto 0);  -- Select
--trivium
           key_out_mux :out STD_LOGIC_VECTOR(15 downto 0)
          );
end bloque_triviums;
architecture Behavioral of bloque_triviums is
signal key: STD_LOGIC_VECTOR(79 downto 0); --Secret 80-bit key
input port
signal iv : STD_LOGIC_VECTOR(79 downto 0); --Secret 80-bit key
input port
signal selclk,reg_selclk : STD_LOGIC_VECTOR(11 downto 0); --Ctrl
CLK 11-bit
signal sys_clk0, sys_clk1, sys_clk2: std_logic; -- 1 BIT
signal sys_clk3, sys_clk4, sys_clk5: std_logic; -- 2 BITS
signal sys_clk6, sys_clk7, sys_clk8: std_logic; -- 8 BITS
signal SYS_CLK9, SYS_CLK10, SYS_CLK11: STD_LOGIC; -- 16 BITS
signal key_out, key_outmplp,key_outfplp : std_logic;
signal key_out2, key_out2mplp, key_out2fplp :std_logic_vector(1
downto 0);
signal key_out8, key_out8mplp, key_out8fplp :std_logic_vector(7
downto 0);
signal key_out16, key_out16mplp, key_out16fplp :
std_logic_vector(15 downto 0);
signal out0_mux16b41, out1_mux16b41 : STD_LOGIC_VECTOR(15 downto
0);
signal in0,in1,in2,in3,in4 : std_logic_vector(15 downto 0);
signal sys_clkm, sys_clk3_1, sys_clk3_2, sys_clk3_8, sys_clk3_16:
std_logic;
-----------------------------------------------------------
-- MULTIPLEXER
-----------------------------------------------------------
    COMPONENT MUX16B41
```

```
    PORT(
         in00   : IN  std_logic_vector(15 downto 0);
         in01   : IN  std_logic_vector(15 downto 0);
         in10   : IN  std_logic_vector(15 downto 0);
         in11   : IN  std_logic_vector(15 downto 0);
         sel    : IN  std_logic_vector(1 downto 0);
         out16b : OUT std_logic_vector(15 downto 0)
        );
    END COMPONENT;
    COMPONENT MUX16B21
    PORT(
         in0   : IN  std_logic_vector(15 downto 0);
         in1   : IN  std_logic_vector(15 downto 0);
         sel   : IN  std_logic;
         out16b: OUT std_logic_vector(15 downto 0)
        );
    END COMPONENT;
------------------------------------------------------------
-- 1BITS
------------------------------------------------------------
    COMPONENT trivium_core
    PORT(
         sys_clk : IN  std_logic;
         resetn  : IN  std_logic;
         cntrl   : IN  std_logic_vector(1 downto 0);
         key     : IN  std_logic_vector(79 downto 0);
         iv      : IN  std_logic_vector(79 downto 0);
         key_out : OUT std_logic
        );
    END COMPONENT;
    COMPONENT trivium_corefplp
    PORT(
         sys_clk : IN  std_logic;
         resetn  : IN  std_logic;
         cntrl   : IN  std_logic_vector(1 downto 0);
         key     : IN  std_logic_vector(79 downto 0);
         iv      : IN  std_logic_vector(79 downto 0);
         key_out : OUT std_logic
        );
    END COMPONENT;
    COMPONENT trivium_coremplp
    PORT(
         sys_clk :  IN  std_logic;
         sys_clk3 : IN  std_logic;
         resetn  :  IN  std_logic;
         cntrl   :  IN  std_logic_vector(1 downto 0);
         key     :  IN  std_logic_vector(79 downto 0);
         iv      :  IN  std_logic_vector(79 downto 0);
         key_out :  OUT std_logic
        );
    END COMPONENT;
------------------------------------------------------------
-- 2BITS
------------------------------------------------------------
    COMPONENT trivium_corex2
```

```
    PORT(
         sys_clk : IN  std_logic;
         resetn  : IN  std_logic;
         cntrl   : IN  std_logic_vector(1 downto 0);
         key     : IN  std_logic_vector(79 downto 0);
         iv      : IN  std_logic_vector(79 downto 0);
         key_out : OUT std_logic_vector(1 downto 0)
        );
    END COMPONENT;
    COMPONENT trivium_corex2fplp
    PORT(
         sys_clk : IN  std_logic;
         resetn  : IN  std_logic;
         cntrl   : IN  std_logic_vector(1 downto 0);
         key     : IN  std_logic_vector(79 downto 0);
         iv      : IN  std_logic_vector(79 downto 0);
         key_out : OUT std_logic_vector(1 downto 0)
        );
    END COMPONENT;
    COMPONENT trivium_corex2mplp
    PORT(
         sys_clk : IN  std_logic;
         sys_clk3 : IN  std_logic;
         resetn  : IN  std_logic;
         cntrl   : IN  std_logic_vector(1 downto 0);
         key     : IN  std_logic_vector(79 downto 0);
         iv      : IN  std_logic_vector(79 downto 0);
         key_out : OUT std_logic_vector(1 downto 0)
        );
    END COMPONENT;
-------------------------------------------------------------
-- 8 BITS
-------------------------------------------------------------
    COMPONENT trivium_corex8
    PORT(
         sys_clk : IN  std_logic;
         resetn  : IN  std_logic;
         cntrl   : IN  std_logic_vector(1 downto 0);
         key     : IN  std_logic_vector(79 downto 0);
         iv      : IN  std_logic_vector(79 downto 0);
         key_out : OUT std_logic_vector(7 downto 0)
        );
    END COMPONENT;
    COMPONENT trivium_corex8fplp
    PORT(
         sys_clk : IN  std_lOGIC;
         resetn  : IN  std_logic;
         cntrl   : IN  std_logic_vector(1 downto 0);
         key     : IN  std_logic_vector(79 downto 0);
         iv      : IN  std_logic_vector(79 downto 0);
         key_out : OUT std_logic_vector(7 downto 0)
        );
    END COMPONENT;
    COMPONENT trivium_corex8mplp
    PORT(
```

```
        sys_clk : IN  std_logic;
        sys_clk3 : IN  std_logic;
        resetn  : IN  std_logic;
        cntrl   : IN  std_logic_vector(1 downto 0);
        key     : IN  std_logic_vector(79 downto 0);
        iv      : IN  std_logic_vector(79 downto 0);
        key_out : OUT std_logic_vector(7 downto 0)
       );
    END COMPONENT;
-------------------------------------------------------------------
-- 16 BITS
-------------------------------------------------------------------
    COMPONENT trivium_corex16
    PORT(
        sys_clk : IN  std_logic;
        resetn  : IN  std_logic;
        cntrl   : IN  std_logic_vector(1 downto 0);
        key     : IN  std_logic_vector(79 downto 0);
        iv      : IN  std_logic_vector(79 downto 0);
        key_out : OUT std_logic_vector(15 downto 0)
       );
    END COMPONENT;
    COMPONENT trivium_corex16fplp
    PORT(
        sys_clk : IN  std_logic;
        resetn  : IN  std_logic;
        cntrl   : IN  std_logic_vector(1 downto 0);
        key     : IN  std_logic_vector(79 downto 0);
        iv      : IN  std_logic_vector(79 downto 0);
        key_out : OUT std_logic_vector(15 downto 0)
       );
    END COMPONENT;
    COMPONENT trivium_corex16mplp
    PORT(
        sys_clk : IN  std_logic;
        sys_clk3 : IN  std_logic;
        resetn  : IN  std_logic;
        cntrl   : IN  std_logic_vector(1 downto 0);
        key     : IN  std_logic_vector(79 downto 0);
        iv      : IN  std_logic_vector(79 downto 0);
        key_out : OUT std_logic_vector(15 downto 0)
       );
    END COMPONENT;
begin
    sys_clk0 <=  clk AND reg_selclk(0);
    sys_clk1 <=  clk AND reg_selclk(1);
    sys_clk2 <=  clk AND reg_selclk(2);
    sys_clk3 <=  clk AND reg_selclk(3);
    sys_clk4 <=  clk AND reg_selclk(4);
    sys_clk5 <=  clk AND reg_selclk(5);
    sys_clk6 <=  clk AND reg_selclk(6);
    sys_clk7 <=  clk AND reg_selclk(7);
    sys_clk8 <=  clk AND reg_selclk(8);
    sys_clk9 <=  clk AND reg_selclk(9);
    sys_clk10 <= clk AND reg_selclk(10);
```

```
    sys_clk11 <= clk AND reg_selclk(11);
    sys_clk3_1  <= sys_clkm and reg_selclk(3);
    sys_clk3_2  <= sys_clkm and reg_selclk(5);
    sys_clk3_8  <= sys_clkm and reg_selclk(8);
    sys_clk3_16 <= sys_clkm and reg_selclk(11);
    in0 <= ("0000000000000000");
    in1 <= (key_out & key_outfplp & key_outmplp & "0000000000000");
    in2 <= ("00" & key_out2 & key_out2mplp & key_out2fplp &
key_out8mplp);
    in3 <= (key_out8 & key_out8fplp);
    in4 <= (key_out8fplp & key_out8mplp);
  --sys_clk3 generation
   SYS_CLK3_process :process  (clk,resetn)
   begin
      IF(resetn = '0') then
         sys_clkm  <= '1';
      ELSIF(clk'event AND clk='1') then
         sys_clkm <= NOT(sys_clkm) ;
      END IF;
   end process;
    PLoad1:process (resetn, clk2)
    begin
    if resetn = '0' then
       iv(79 downto 0) <= (OTHERS =>'0');
       key(79 downto 0) <= (OTHERS =>'0');
       selclk(11 downto 0) <= (OTHERS =>'0');
       reg_selclk(11 downto 0) <= (OTHERS =>'0');
    elsif clk2'event and clk2  = '1' then
       if load1 = '1' then
           reg_selclk(11 downto 0) <= (OTHERS =>'0');
           selclk(11 downto 1) <= selclk(10 downto 0);
           selclk(0) <= key(79);
           key(79 downto 1) <= key(78 downto 0);
           key(0) <= iv(79);
           iv(79 downto 1) <= iv(78 downto 0);
           iv(0) <= serie;
       else
           reg_selclk <= selclk(11 downto 0);
       end if;
    end if;
    end process;
MUX16B41_0:mux16b41 PORT MAP (
        in00 => in0,
        in01 => in1,
        in10 => in2,
        in11 => in3,
        sel => slct(1 downto 0),
        out16b => out0_mux16b41
    );
    MUX16B41_1:mux16b41 PORT MAP (
        in00 => in4,
        in01 => key_out16,
        in10 => key_out16mplp,
        in11 => key_out16fplp,
        sel => slct(1 downto 0),
```

```
        out16b => out1_mux16b41
    );
    mux16b21_0:mux16b21 PORT MAP (
        in0 => out0_mux16b41,
        in1 => out1_mux16b41,
        sel => slct(2),
        out16b => key_out_mux
    );
    dut_tv:trivium_core PORT MAP (
        sys_clk => sys_clk0,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_ouT
    );
    dut_tv_fplp:trivium_corefplp PORT MAP (
        sys_clk => sys_clk1,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_outfplp
    );
    dut_tv_mplp:trivium_coremplp PORT MAP (
        sys_clk => sys_clk2,
        sys_clk3 => sys_clk3_1,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_outmplp
    );
-----------------------------------------------------------------
-- 2 BITS
-----------------------------------------------------------------
    dut_tvx2:trivium_corex2 PORT MAP (
        sys_clk => sys_clk3,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_out2
    );
    dut_tvx2fplp:trivium_corex2fplp PORT MAP (
        sys_clk => sys_clk4,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_out2fplp
    );
    dut_tvx2mplp:trivium_corex2mplp PORT MAP (
        sys_clk => sys_clk5,
        sys_clk3 => sys_clk3_2,
```

```
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_out2mplp
    );
-----------------------------------------------------------------
-- 8 BITS
-----------------------------------------------------------------
    dut_tvx8:trivium_corex8 PORT MAP (
        sys_clk => sys_clk6,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_out8
    );
    dut_tvx8fplp:trivium_corex8fplp PORT MAP (
        sys_clk => sys_clk7,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_out8fplp
    );
    dut_tvx8mplp:trivium_corex8mplp PORT MAP (
        sys_clk => sys_clk8,
        sys_clk3 => sys_clk3_8,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_out8mplp
    );
-----------------------------------------------------------------
-- 16 BITS
-----------------------------------------------------------------
    dut_tvx16:trivium_corex16 PORT MAP (
        sys_clk => sys_clk9,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_out16
    );
    dut_tvx16fplp:trivium_corex16fplp PORT MAP (
        sys_clk => sys_clk10,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_out16fplp
    );
    dut_tvx16mplp:trivium_corex16mplp PORT MAP (
        sys_clk => sys_clk11,
```

```
        sys_clk3 => sys_clk3_16,
        resetn => resetn,
        cntrl => cntrl,
        key => key,
        iv => iv,
        key_out => key_out16mplp
    );
end Behavioral;
```

B.5. *Testbench* para *ModelSim* del Bloque Triviums

```
LIBRARY ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;
entity bloque_triviums_tb is
end bloque_triviums_tb;
ARCHITECTURE behavior OF bloque_triviums_tb IS
  -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT bloque_triviums
    Port ( clk     :in STD_LOGIC; --System or User clock
           clk2     :in STD_LOGIC; --System or User clock
           resetn   :in STD_LOGIC;
           cntrl    :in STD_LOGIC_VECTOR(1 downto 0);
           serie    :in STD_LOGIC;
           load1    :in STD_LOGIC;  -- Load Key and IV
           slct     :in STD_LOGIC_VECTOR(2 downto 0);  -- Select
trivium
           key_out_mux :out STD_LOGIC_VECTOR(15 downto 0)
         );
    END COMPONENT;
   --Inputs
   signal clk   : std_logic := '0';
   signal clk2   : std_logic := '0';
   signal resetn : std_logic := '0';
   signal cntrl  : std_logic_vector(1 downto 0) := "00";
   signal serie  : std_logic;
   signal load1  : std_logic;
   signal slct   : std_logic_vector(2 downto 0);
   signal selclk  : std_logic_vector(11 downto 0) :=  (others =>
'0');
   signal key    : std_logic_vector(79 downto 0) := (others =>
'0');
   signal iv     : std_logic_vector(79 downto 0) := (others =>
'0');
   --Outputs
   signal key_out_mux : std_logic_vector(15 downto 0);
   -- Clock period definitions
   constant clk_period : time := 50 ns;
   constant clk2_period : time := 50 ns;
   signal result : std_logic_vector(127 downto 0) := (OTHERS =>
```

```
'0');
   signal cycle_cnt : std_logic_vector(16 downto 0) := (OTHERS =>
'0');
   signal correct_output : std_logic_vector(127 downto 0) :=
(OTHERS => '0');
   signal error : std_logic := '0';
   signal hab_error : std_logic := '0';
   signal load_start:  std_logic;
   signal load_finish: std_logic;
function littler_endian (b: std_logic_vector(127 downto 0)) return
std_logic_vector is
variable result : std_logic_vector(127 downto 0);
begin
   for i in 0 to 15 loop
     result(((i*8)+7) downto (i*8)) := b(i*8) & b((i*8) + 1) &
                                       b((i*8) + 2) & b((i*8) + 3) &
                                      b((i*8) + 4) & b((i*8) + 5) &
                                      b((i*8) + 6) & b((i*8) + 7);
    end loop;
   return result;
end;
begin
   -- Instantiate the Unit Under Test (UUT)
   uut: bloque_triviums PORT MAP (
          clk    => clk,
          clk2    => clk2,
          resetn  => resetn,
          cntrl   => cntrl,
          serie   => serie,
          load1   => load1,
          slct    => slct,
          key_out_mux => key_out_mux
        );
   -- Clock process definitions
   clk_process :process
   begin
       clK <= '0';
       wait for clk_period/2;
       clk <= '1';
       wait for clk_period/2;
   end process;
   clk2_process :process
   begin
       clk2 <= '0';
       wait for clk2_period/2;
       clk2 <= '1';
       wait for clk2_period/2;
   end process;
   -- Main process
   main_proc: process
   begin
      -- Set signal values
         resetn <= '0';
         cntrl  <= "00";
         slct   <= "000";
```

```
         selclk   <= "111111111111";
         key <= X"0F62B5085BAE0154A7FA";
         iv  <= X"288FF65DC42B92F960C7";
         load_start <= '0';
      wait for 15 ns;
         resetn <= '1';
      wait on clk until clk = '1';
         -- Serial load init
         load_start <= '1';
      wait on clk until clk = '1';
         load_start <= '0';
      wait on load_finish until load_finish = '1';
      wait on clk until clk = '1';
      wait for 187 ns;
      wait on clk until clk = '0';
         -- Test the four triviums
         slct   <= "000";
         cntrl <= "10";
         hab_error <= '0';
      wait for clk_period*2; -- cambio x1-> x3
      -- wait on clk until clk = '0';
         cntrl <= "11";
      wait for clk_period*1152;
         cntrl <= "01";
      wait for clk_period*126;
      wait for clk_period/2;
         correct_output <=
littler_endian(X"A4386C6D7624983FEA8DBE7314E5FE1F");
         hab_error <= '1';
      wait for clk_period;
         correct_output <= X"00000000000000000000000000000000";
         hab_error <= '0';
      wait for clk_period*127;
--correct_output<=
--littler_endian(X"9D102004C2CEC99AC3BFBF003A66433F");
correct_output<=littler_endian(X"0D102004C2CEC99AC3BFBF003A66433F")
;
         hab_error <= '1';
      wait for clk_period;
         correct_output <= X"00000000000000000000000000000000";
        hab_error <= '0';
      wait for clk_period*127;
correct_output<=littler_endian(X"3089A98FAD8512C49D7AABC0639F90C5")
;
         hab_error <= '1';
      wait for clk_period;
         hab_error <= '0';
         correct_output <= X"00000000000000000000000000000000";
      wait;
   end process;
   -- Serial load process
   serial:process
   begin
      wait on clk2 until clk2 = '0';
      if load_start = '1' then
```

```
            load_finish <= '0';
            load1 <= '1';
            for i in 11 downto 0 loop
                serie <= selclk(i);
                wait on clk2 until clk2 = '0';
            end loop;
            for i in 79 downto 0 loop
                serie <= key(i);
                wait on clk2 until clK2 = '0';
            end loop;
            for i in 79 downto 0 loop
                serie <= iv(i);
                wait on clk2 until clk2 = '0';
            end loop;
            load1 <= '0';
            load_finish <= '1';
        else
            load_finish <= '0';
            load1 <= '0';
        end if;
    end process;
    --Result Shift Register
    --res_shift_reg : process
    --begin
        --wait until ((cntrl = "11") OR (cntrl = "01"));
        --loop
           --result <= result(126 downto 0) & key_out_mux;
           --cycle_cnt <=  cycle_cnt + 1;
           --wait for clk1_period;
         --end loop;
    --END PROCESS;
    --Output error
        output_error : process (clk)
        begin
           IF(clk'event AND clk='1') then
              if hab_error = '1' then
                  if correct_output = result then
                      error <= '0';
                  else
                      error <= '1';
                  end if;
                    end if;
           end if;
        end process;
end behavior;
```

B.6. *Script* de comandos para *Encounter RTL-GDSII*

```
#############################################################
#  Encounter Command Logging File                           #
#  Created on Mon Apr 21 11:34:47 2016
#  Basado en un place & route manual                        #
#############################################################
loadConfig encounter.cities.conf 0
```

```
set_global timing_clock_phase_propagation both
commitConfig
fit
setDrawView fplan
getIoFlowFlag
setIoFlowFlag 0
floorPlan -coreMarginsBy die -site core -r 0.975591665458 0.700021
40.0 40.0 40.0 40.0
uiSetTool select
getIoFlowFlag
fit
clearGlobalNets
globalNetConnect VDD -type pgpin -pin VDD -inst *
globalNetConnect VSS -type pgpin -pin VSS -inst *
addRing -spacing_bottom 4 -width_left 10 -width_bottom 10 -
width_top 10 -spacing_top 4 -layer_bottom M1 -stacked_via_top_layer
M9 -width_right 10 -around core -jog_distance 0.14 -offset_bottom 5
-layer_top M1 -threshold 0.14 -offset_left 5 -spacing_right 4 -
spacing_left 4 -offset_right 5 -offset_top 5 -layer_right M2 -nets
{VSS VDD } -stacked_via_bottom_layer M1 -layer_left M2
placeDesign -prePlaceOpt
clearClockDomains
setClockDomains -all
timeDesign -preCTS -idealClock -pathReports -drvReports -
slackReports -numPaths 50 -prefix fallos_trivium_preCTS -outDir
timingReports
optDesign -preCTS
clockDesign -specFile clock_cities.const -outDir clock_report -
fixedInstBeforeCTS
saveClockNets -output cities.ctsntf
displayClockPhaseDelay -preRoute
setDrawView place
setDrawView ameba
setDrawView fplan
getFillerMode -quiet
addFiller -cell FILL64 FILL32 FILL16 FILL8 FILL4 FILL2 FILL1 -
prefix FILLER
sroute -connect { blockPin padPin padRing corePin floatingStripe }
-layerChangeRange { 1 8 } -blockPinTarget { nearestRingStripe
nearestTarget } -padPinPortConnect { allPort oneGeom } -
checkAlignedSecondaryPin 1 -blockPin useLef -allowJogging 1 -
crossoverViaBottomLayer 1 -allowLayerChange 1 -targetViaTopLayer 8
-crossoverViaTopLayer 8 -targetViaBottomLayer 1 -nets { VSS VDD }
wroute
clearClockDomains
setClockDomains -all
timeDesign -postRoute -pathReports -drvReports -slackReports -
numPaths 50 -prefix fallos_trivium_postRoute -outDir timingReports
clearClockDomains
setClockDomains -all
timeDesign -postRoute -hold -pathReports -slackReports -numPaths 50
-prefix fallos_trivium_postRoute -outDir timingReports
optDesign -postRoute
saveDesign cities_routed.enc
streamOut netlist/cities.gds -mapFile gds2.map -libName DesignLib -
```

```
stripes 1 -units 2000 -mode ALL
saveNetlist netlist/cities_routed.v -excludeLeafCell
isExtractRCModeSignoff
isExtractRCModeSignoff
rcOut -spef netlist/cities_routed.spef
delayCal -sdf netlist/cities_routed.sdf
```

B.7. *Script* de restricciones para síntesis en *Encounter RTL*

```
create_clock -name "clk" -period 40 -waveform {0 20} {SYS_CLK}
set_clock_uncertainty 0.1 clk
set_clock_latency 0.2 clk
set_clock_transition 0.1 clk
set_dont_touch_network  [ find clock clk ]
set_dont_touch_network RESETN
set_ideal_network RESETN
#set_ideal_network -no_propagate RESETN
set_max_area 1000
set_operating_conditions -library fsa0a_c_sc_tc TCCOM
set_wire_load_model -name G5K -library fsa0a_c_sc_tc
set_wire_load_mode top
```

B.8. *Script* para generación del fichero SAIF en *ModelSim*

```
vsim -voptargs=+acc -t ns -c -L
/cad/KITS/tsmc/tsmc_0.09lp/TSMCHOME/digital/Front_End/verilog/tcbn9
0lphp_150j/modelsim/mti_6.6d/tcbn90lphp -sdftyp
bloque_triviums_tb/uut=bloque_triviums.sdf -sdfnoerror -foreign
"dpfli_init /cad/SYNOPSYS_2006/sparcOS5/power/dpfli/dpfli.so"
work.bloque_triviums_tb
set_toggle_region bloque_triviums_tb/uut
run 1000
toggle_start
run 62000
toggle_stop
toggle_report bloque_triviums.saif 1e-9 bloque_triviums_tb/uut
quit
```

B.9. *Testbench* en simulaciones *ModelSim* para ASIC CITIES

```
LIBRARY ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use ieee.numeric_std.ALL;
entity cities_tb1_bt is
end cities_tb1_bt;
ARCHITECTURE behavior OF cities_tb1_bt IS
```

```
  -- Component Declaration for the Unit Under Test (UUT)
  component cities is
  port(
     -- Pines comunes a todos los bloques
     resetn:                in std_logic;
     -- Pines comunes a bloque_triviums y fallos_triviums
     cntrl:                 in std_logic_vector(1 downto 0);
     load:                  in std_logic; -- load1 y carga_key
     -- Pines de bloque_triviums
     clk_bt:                in std_logic;
     clk2_bt:               in std_logic;
     serie_bt:              in std_logic;
     --load1_bt:              in std_logic;
     slct_bt:               in std_logic_vector(2 downto 0);
     key_out_mux:           out std_logic_vector(15 downto 0);
     -- Pines de fallos_trivium
     clk_ft:                in std_logic;
     clk2_ft:               in std_logic;
     carga_ft:              in std_logic;
     --carga_key_ft:          in std_logic;
     key_in_ft:             in std_logic;
     select_tv_ft:          in std_logic_vector(2 downto 0);
     -- Pines de sistema_completo
     init_sc:               in std_logic;
     transmit_sc:           in std_logic;
     xin_sc:                in std_logic;
     mosi_receptor_sc:      in std_logic;
     nss_receptor_sc:       in std_logic;
     key_serial_sc:         in std_logic;
     load_key_serial_sc:    in std_logic;
     cambia_flanco_sc:      in std_logic;
     clke_sc:               in std_logic;
     clkr_sc:               in std_logic;
     mosi_emisor_sc:        out std_logic;
     nss_emisor_sc:         out std_logic;
     xout_sc:               out std_logic;
     valida_sc:             out std_logic;
     exito_sc:              out std_logic
   );
   end component;
   --Inputs
     signal resetn:                 std_logic := '0';
     -- Pines comunes a bloque_triviums y fallos_triviums
     signal cntrl:                  std_logic_vector(1 downto 0);
     signal load:                   std_logic := '0'; -- comun a
-- load1 y carga_key
     -- Pines de bloque_triviums
     signal clk_bt:                 std_logic := '0';
     signal clk2_bt:                std_logic := '0';
     signal serie_bt:               std_logic := '0';
     signal slct_bt:                std_logic_vector(2 downto 0);
     signal key_out_mux:            std_logic_vector(15 downto 0);
     signal clk_ft:                 std_logic := '0';
     signal clk2_ft:                std_logic := '0';
     signal carga_ft:               std_logic := '0';
```

```
     signal key_in_ft:              std_logic := '0';
     signal select_tv_ft:           std_logic_vector(2 downto 0);
     -- Pines de sistema_completo
     signal init_sc:                std_logic := '0';
     signal transmit_sc:            std_logic := '0';
     signal xin_sc:                 std_logic := '0';
     signal mosi_receptor_sc:       std_logic := '0';
     signal nss_receptor_sc:        std_logic := '0';
     signal key_serial_sc:          std_logic := '0';
     signal load_key_serial_sc:     std_logic := '0';
     signal cambia_flanco_sc:       std_logic := '0';
     signal clke_sc:                std_logic := '0';
     signal clkr_sc:                std_logic := '0';
     signal mosi_emisor_sc:         std_logic := '0';
     signal nss_emisor_sc:          std_logic := '0';
     signal xout_sc:                std_logic := '0';
     signal valida_sc:              std_logic := '0';
     signal exito_sc:               std_logic := '0';
   -- Señales intermedias
     signal clk, clk2: std_logic;
     signal serie, load1: std_logic;
     signal slct: std_logic_vector(2 downto 0);
     signal selclk  : std_logic_vector(11 downto 0) :=  (others =>
'0');
     signal key    : std_logic_vector(79 downto 0) := (others =>
'0');
     signal iv     : std_logic_vector(79 downto 0) := (others =>
'0');
   -- Clock period definitions
   constant clk_period : time := 50 ns;
   constant clk2_period : time := 50 ns;

   signal result : std_logic_vector(127 downto 0) := (OTHERS =>
'0');
   signal cycle_cnt : std_logic_vector(16 downto 0) := (OTHERS =>
'0');
   signal correct_output : std_logic_vector(127 downto 0) :=
(OTHERS => '0');

   signal error : std_logic := '0';
   signal hab_error : std_logic := '0';
   signal load_start:  std_logic;
   signal load_finish: std_logic;
function littler_endian (b: std_logic_vector(127 downto 0)) return
std_logic_vector is
variable result : std_logic_vector(127 downto 0);
begin
  for i in 0 to 15 loop
    result(((i*8)+7) downto (i*8)) := b(i*8) & b((i*8) + 1) &
                                      b((i*8) + 2) & b((i*8) + 3) &
                                      b((i*8) + 4) & b((i*8) + 5) &
                                      b((i*8) + 6) & b((i*8) + 7);
      end loop;
      return result;
end;
```

```
begin
   -- Instantiate the Unit Under Test (UUT)
   uut: cities PORT MAP (
     resetn => resetn,
     cntrl  => cntrl,
     load  => load,
     -- Pines de bloque_triviums
     clk_bt  => clk_bt,
     clk2_bt  => clk2_bt,
     serie_bt  => serie_bt,
     slct_bt  => slct_bt,
     key_out_mux  => key_out_mux,
     -- Pines de fallos_trivium
     clk_ft  => clk_ft,
     clk2_ft  => clk2_ft,
     carga_ft  => carga_ft,
     key_in_ft  => key_in_ft,
     select_tv_ft  => select_tv_ft,
     -- Pines de sistema_completo
     init_sc  => init_sc,
     transmit_sc  => transmit_sc,
     xin_sc  => xin_sc,
     mosi_receptor_sc  => mosi_receptor_sc,
     nss_receptor_sc  => mosi_receptor_sc,
     key_serial_sc  => key_serial_sc,
     load_key_serial_sc  => load_key_serial_sc,
     cambia_flanco_sc  => cambia_flanco_sc,
     clke_sc  => clke_sc,
     clkr_sc  => clkr_sc,
     mosi_emisor_sc  => mosi_emisor_sc,
     nss_emisor_sc  => nss_emisor_sc,
     xout_sc  => xout_sc,
     valida_sc  => valida_sc,
     exito_sc  => exito_sc);
   -- Asignacion de pines
     clk_bt <= clk;
     clk2_bt <= clk2;
     serie_bt <= serie;
     load <= load1;
     slct_bt <= slct;
     --key_out_mux <= key_out_mux;
   -- Clock process definitions
   clk_process :process
   begin
       clK <= '0';
       wait for clk_period/2;
       clk <= '1';
       wait for clk_period/2;
   end process;
   clk2_process :process
   begin
       clk2 <= '0';
       wait for clk2_period/2;
       clk2 <= '1';
       wait for clk2_period/2;
```

```
   end process;
   -- Main process
   main_proc: process
   begin
      -- Set signal values
         resetn <= '0';
         cntrl  <= "00";
         slct   <= "001";
         selclk   <= "111111111111";
         key <= X"0F62B5085BAE0154A7FA";
         iv  <= X"288FF65DC42B92F960C7";
         load_start <= '0';
      wait for 15 ns;
         resetn <= '1';
      wait on clk until clk = '1';
         -- Serial load init
         load_start <= '1';
      wait on clk until clk = '1';
         load_start <= '0';
      wait on load_finish until load_finish = '1';
      wait on clk until clk = '1';
      wait for 187 ns;
      wait on clk until clk = '0';
         -- Test the four triviums
         slct   <= "001";
         cntrl <= "10";
         hab_error <= '0';
      wait for clk_period*2; -- cambio x1-> x3
      -- wait on clk until clk = '0';
         cntrl <= "11";
      wait for clk_period*1152;
         cntrl <= "01";
      wait for clk_period*126;
      wait for clk_period/2;
         correct_output <=
littler_endian(X"A4386C6D7624983FEA8DBE7314E5FE1F");
         hab_error <= '1';
      wait for clk_period;
         correct_output <= X"00000000000000000000000000000000";
         hab_error <= '0';
      wait for clk_period*127;
         -- correct_output <=
littler_endian(X"9D102004C2CEC99AC3BFBF003A66433F");
         correct_output <=
littler_endian(X"0D102004C2CEC99AC3BFBF003A66433F");
         hab_error <= '1';
      wait for clk_period;
         correct_output <= X"00000000000000000000000000000000";
        hab_error <= '0';
      wait for clk_period*127;
         correct_output <=
littler_endian(X"3089A98FAD8512C49D7AABC0639F90C5");
         hab_error <= '1';
      wait for clk_period;
         hab_error <= '0';
```

```
        correct_output <= X"00000000000000000000000000000000";
      wait;
   end process;
   -- Serial load process
   serial:process
   begin
      wait on clk2 until clk2 = '0';
      if load_start = '1' then
          load_finish <= '0';
          load1 <= '1';
          for i in 11 downto 0 loop
              serie <= selclk(i);
              wait on clk2 until clk2 = '0';
          end loop;
          for i in 79 downto 0 loop
              serie <= key(i);
              wait on clk2 until clK2 = '0';
          end loop;
          for i in 79 downto 0 loop
              serie <= iv(i);
              wait on clk2 until clk2 = '0';
          end loop;
          load1 <= '0';
          load_finish <= '1';
      else
          load_finish <= '0';
          load1 <= '0';
      end if;
   end process;
   --Result Shift Register
   --res_shift_reg : process
   --begin
      --wait until ((cntrl = "11") OR (cntrl = "01"));
      --loop
         --result <= result(126 downto 0) & key_out_mux;
         --cycle_cnt <=  cycle_cnt + 1;
         --wait for clk1_period;
       --end loop;
   --END PROCESS;
   --Output error
      output_error : process (clk)
      begin
         if(clk'event AND clk='1') then
            if hab_error = '1' then
                if correct_output = result then
                    error <= '0';
                else
                    error <= '1';
                end if;
                  end if;
         end if;
      end process;
end behavior;
```

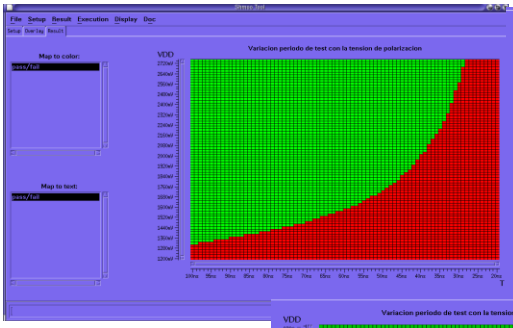
APÉNDICE C. RESULTADOS DE LAS MEDIDAS EXPERIMENTALES

A continuación se presentan algunos datos complementarios de las medidas experimentales llevadas a cabo en las distintas muestras de los circuitos integrados recibidos. En concreto se tienen 10 ASICs disponibles para la caracterización. Los datos que se van a mostrar en los siguientes apartados son los referidos a los resultados del test de máxima frecuencia, *shmo-plots* de la muestra 2 y los resultados del cálculo del consumo de potencia a tres frecuencias de reloj, 20 MHz, 40 MHz y 50 MHz.

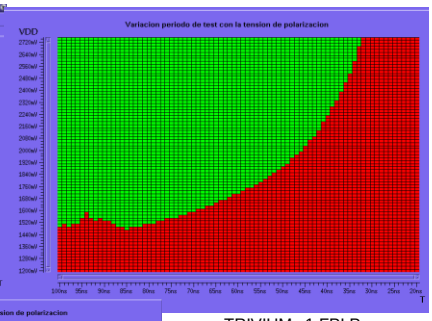
C.1. Medidas de frecuencia máxima en las muestras

Datos de frecuencia máxima (MHz) a temperatura ambiente.										
Condiciones de polarización típica. V_{DD_CORE}=1,2 V y V_{DD_RING}=2,5 V										
Trivium	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**
Trivium_x1	129	129	129	128	128	129	128	129	130	128
Trivium_x1mp	105	105	105	105	104	105	104	105	105	104
Trivium_x1fp	115	114	114	114	114	115	114	114	114	115
Trivium_x2[2]	124	123	122	122	120	123	122	123	123	122
Trivium_x2mp	119	118	118	118	118	118	117	119	119	118
Trivium_x2fp	117	117	117	117	116	117	118	117	118	116
Trivium_x8	116	114	115	115	114	116	115	115	115	116
Trivium_x8mp	101	101	101	101	99	102	102	100	101	102
Trivium_x8fp	86	87	86	86	85	86	86	86	86	86
Trivium_x16	99	99	98	98	97	99	99	98	99	99
Trivium_x16mp	97	98	97	97	96	98	98	97	98	99
Trivium_x16fp	76	75	77	75	74	76	75	74	74	75

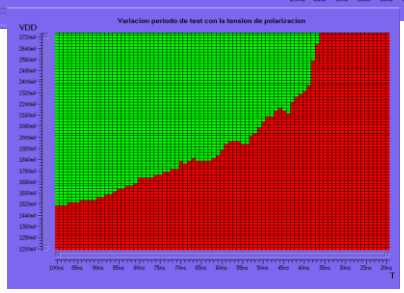
C.2. *Shmoo-Plots* de los cifradores de la muestra 2



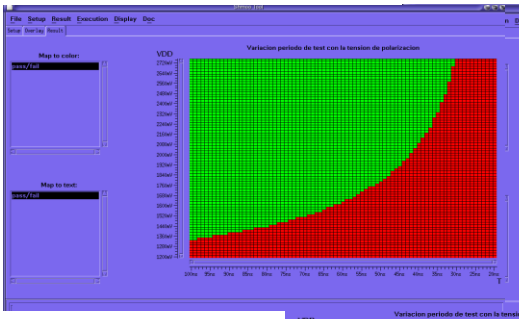
TRIVIUM x1 STANDARD



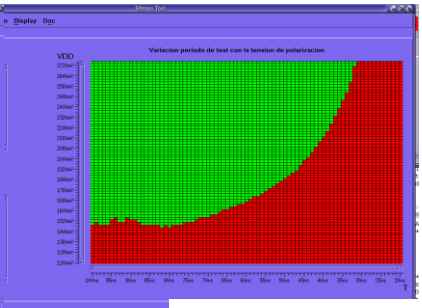
TRIVIUM x1 FPLP



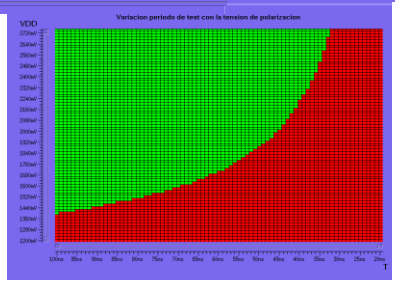
TRIVIUM x1 MPLP



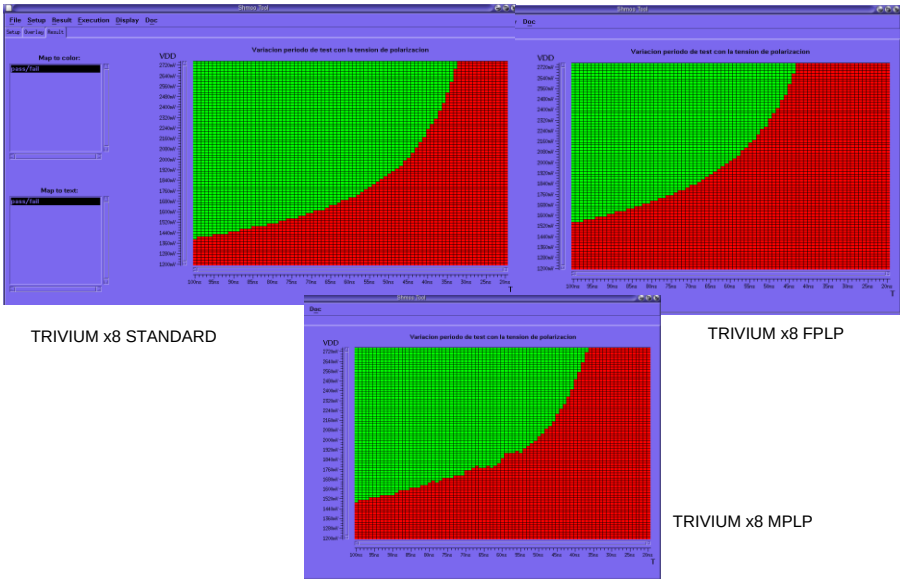
TRIVIUM x2 STANDARD



TRIVIUM x2 FPLP



TRIVIUM x2 MPLP



TRIVIUM x8 STANDARD

TRIVIUM x8 FPLP

TRIVIUM x8 MPLP

C.3. Medidas del consumo de potencia en las muestras

Datos del consumo de potencia (μW) a 50 MHz y polarización típica.										
Trivium	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**
Trivium_x1	573	558	565	554	580	580	565	580	599	565
Trivium_x1mp[1]	255	255	255	255	277	300	273	285	292	273
Trivium_x1fp	262	273	270	270	288	285	273	292	285	247
Trivium_x2[2]	779	771	779	756	768	790	786	768	779	779
Trivium_x2mp	307	318	318	300	322	318	307	315	322	315
Trivium_x2fp	281	300	303	285	281	307	281	281	270	285
Trivium_x8	513	543	543	528	543	539	550	554	535	532
Trivium_x8mp	491	502	491	491	476	494	498	491	502	487
Trivium_x8fp	491	517	509	498	509	517	513	513	502	498
Trivium_x16	569	573	569	562	569	550	550	584	573	565
Trivium_x16mp	543	562	565	580	554	565	565	577	584	573
Trivium_x16fp	674	685	685	704	678	659	659	696	719	685

Datos del consumo de potencia (μW) a 40 MHz y polarización típica.										
Trivium	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**
Trivium_x1	461	468	457	442	479	461	464	464	468	472
Trivium_x1mp	232	240	217	217	228	206	225	228	243	236
Trivium_x1fp	262	221	225	228	240	217	240	243	247	236
Trivium_x2	640	652	637	637	640	625	652	625	633	644
Trivium_x2mp	258	247	258	255	255	277	240	251	273	270
Trivium_x2fp	255	273	232	247	255	251	232	232	240	262
Trivium_x8	408	397	423	427	427	399	431	434	419	419
Trivium_x8mp	412	412	408	401	404	442	416	416	401	393
Trivium_x8fp	416	401	427	419	442	412	431	434	427	419
Trivium_x16	457	449	461	483	476	457	479	449	457	453
Trivium_x16mp	464	569	461	472	468	464	453	457	472	457
Trivium_x16fp	550	461	562	580	569	569	580	554	565	577

Datos del consumo de potencia (μW) a 20 MHz y polarización típica..										
Trivium	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**
Trivium_x1	236	221	240	225	240	228	240	232	225	232
Trivium_x1mp	120	109	139	109	105	120	112	112	109	116
Trivium_x1fp	112	105	124	101	116	105	120	127	97	124
Trivium_x2	303	341	303	315	333	318	322	315	326	311
Trivium_x2mp	120	120	127	127	139	131	135	135	146	127
Trivium_x2fp	120	135	116	97	135	139	131	112	116	116
Trivium_x8	206	213	217	225	221	221	225	225	202	225
Trivium_x8mp	187	202	191	206	213	202	210	210	187	195
Trivium_x8fp	195	183	191	202	221	210	202	228	195	210
Trivium_x16	236	221	232	243	221	221	228	236	213	232
Trivium_x16mp	225	273	217	232	221	228	232	228	213	228
Trivium_x16fp	277	225	262	288	281	281	296	300	266	277