Universidad Autónoma de San Luis Potosí

Facultad de Ciencias

Diseño e implementación de Circuitos Integrados de Aplicación Específica (ASICs)

TESIS

para obtener el grado de

Maestro en Ciencias Aplicadas

presenta:

José de Jesús de la Rosa de la Rosa

Directores de tesis:

Dra. Marcela Mejía Carlos
Dr. Miguel Ángel Lastras Montaño

San Luis Potosí, S.L.P. Diciembre de 2020

Universidad Autónoma de San Luis Potosí
Facultad de Ciencias

Tesis:

"Diseño e Implementación de Circuitos Integrados de Aplicación Específica (ASICs)"

Director de Tesis	Director de Tesis
Dra. Marcela Mejía Carlos	Dr. Miguel Ángel Lastras Montaño

Sinodal
Dr. Edgar Armando Cerda Méndez

Sinodal
Dr. Luis Felipe Lastras Martínez

Sinodal
Dr. José Salome Murguía Ibarra

Declaración de autoría y originalidad de la tesis

Yo, **José de Jesús de la Rosa de la Rosa**, estudiante del Posgrado en Ciencias Aplicadas de la Facultad de Ciencias de la Universidad Autónoma de San Luis Potosí, como autor de la tesis "**Diseño e Implementación de Circuitos Integrados de Aplicación Especifica (ASICs)**", declaro que esta tesis es una obra original, inédita, auténtica, personal, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales vigentes que protegen los derechos de autor y de propiedad intelectual e industrial. Las ideas, doctrinas, resultados y conclusiones a los que he llegado son de mi absoluta responsabilidad.

Resumen

Los circuitos integrados se han convertido en los principales componentes de la mayoría de dispositivos electrónicos que usamos en nuestra vida diaria. Éstos muestran importantes características, tales como bajo costo, alta confiabilidad, bajos requisitos de energía y altas velocidades de procesamiento, en comparación con los tubos de vacío y los transistores que los precedieron. Debido al enorme crecimiento y desarrollo de los circuitos integrados así como las herramientas para desarrollarlos, hoy en día se han convertido en una opción viable para el desarrollo particular de dispositivos electrónicos a bajo costo. En esta tesis se presenta el diseño e implementación de dos sistema: (1) un sistema de encriptación por sincronización de autómatas celulares (ESCA, por sus siglas en inglés), y (2) un diseño modular conformado por sumadores, un oscilador en forma de anillo y un decodificador (creados a partir de las celdas estándar INV, NAND y NOR). Ambos sistemas fueron implementados utilizando la metodología de *circuito integrado de aplicación específica* (ASIC, por sus siglas en inglés). El sistema de cifrado es un algoritmo de clave simétrica y ha sido desarrollado a partir de la evolución de los autómatas celulares con la regla 90 y, como lo indica su nombre, por la sincronización de un par de autómatas celulares. El sistema se compone de 3 etapas: permutaciones Ψ y Φ, un bloque para una transformación no lineal conocida como caja de sustitución y un bloque de pre-procesado (así como sus respectivos módulos inversos de los dos últimos bloques mencionados). Adicionalmente, el sistema hace uso de un generador de números pseudo-aleatorios en las etapas de las permutaciones. Este diseño fue implementado en tecnología CMOS de $0.5\,\mu$m empleando software de la compañía *Cadence*, *Mentor Graphics* y *Synopsys*. Los resultados de las simulaciones muestran un correcto funcionamiento del sistema a una frecuencia menor o igual a 33 MHz. El segundo diseño, consistió en el diseño *full-custom* de 3 celdas estándar (INV, NAND y NOR) para la elaboración de un oscilador en forma de anillo, un decodificador de 2 bits, y 2 medios sumadores de 2 bits. Estos últimos implementados compuertas NAND y con compuertas NOR. El diseño *full-custom* de las celdas y de los bloques mencionados se realizó empleando el software *Electric* y la simulación se llevó a cabo en el software *LTSpice*. Este diseño fue implementado en la tecnología pseudo-CMOS de $5\,\mu$m, utilizando unicamente transistores tipo nMOS. Debido a lo anterior, las implementaciones se realizaron utilizando una tecnología de pseudo-CMOS, en la cual se contemplan dos suministros de voltaje, mejorando así notoriamente el rendimiento de las compuertas, así como el voltaje pico a pico alcanzado. Ambos diseños cumplieron las verificaciones necesarias, y las simulaciones muestran un correcto funcionamiento.

Abstract

Integrated circuits (ICs) have become the main components of most electronic devices that we use in our daily lives. These show important characteristics such as low cost, high reliability, low power requirements, and high processing speeds, compared to the vacuum tubes and transistors that preceded them. Due to the huge growth and development of ICs, as well as the tools to develop them, today they have become a viable option for the development of low-cost electronic devices. This thesis presents the design and implementation of two systems: (1) the Encryption by Synchronization in Cellular Automata (ESCA), and (2) a modular design that consist of adders, a ring oscillator and a decoder (created from the standard cells INV, NAND and NOR), implemented in a Application Specific Integrated Circuit (ASIC) technology. The encryption system is a symmetric key algorithm and has been developed from the evolution of cellular automata using the *rule 90* and, as its name indicates, by the synchronization of a pair of cellular automata. The system consists of 3 stages: Ψ and Φ permutations, a block for a non-linear transformation (known as a substitution box or S-box), and a pre-processing block (as well as their respective inverse modules of the last two blocks mentioned above). Additionally, the system requires a pseudo-random number generator for the permutation stages. This design was implemented in $0.5\,\mu$m CMOS technology using software from *Cadence*, *Mentor Graphics*, and *Synopsys*. The results of the simulations show a correct operation of the system with a frequency up to 33 MHz. The second system consists of a *full-custom* design of 3 standard cells (INV, NAND and NOR) for the implementation of a ring oscillator, a 2-bit decoder and two 2-bit half adders, one of them implemented with NAND gates and another with NOR gates. The *full-custom* design of the cells and blocks was carried out using the *Electric* and the simulation was carried out in *LTSpice*. This design was implemented in a $5\,\mu$m pseudo-CMOS technology, in which only nMOS transistors were available. For this, we implemented the system using a pseudo-CMOS design style using two voltage supplies, thus significantly improving the performance of the gates, as well as the peak-to-peak voltage achieved. Both designs met the necessary requirements and their simulations show correct operation.

Agradecimientos

Quiero agradecer a toda mi familia por el apoyo brindado, en especial a mis padres Alberto y Teresa, quienes me han guiado a lo largo de mi vida forjando en mí, valores y principios que han sido de vital importancia para el cumplimiento de mis metas; por toda su paciencia y amor que siempre han tenido conmigo. A Diana Rangel, quien ha sido un gran apoyo durante todo el desarrollo del trabajo, así como una gran compañera en mi vida personal, gracias por todo. Mis más sinceros agradecimientos a mis asesores, la Dra. Marcela Mejía Carlos y el Dr. Miguel Ángel Lastras Montaño, quienes me dieron la oportunidad de trabajar con ellos, me brindaron todo el tiempo y disposición necesarios para el desarrollo del trabajo; por toda su paciencia y consejos hacia mi, que hasta la fecha, me siguen brindando. A todos mis profesores que compartieron sus conocimientos conmigo para que pudiera tener la mejor preparación, por todo el tiempo fuera de clases que tomaron para poder orientarme en mi formación académica y por sus valiosos consejos. A mis compañeros, de los cuales ahora tengo el gusto de considerarme ser su amigo y que han compartido conmigo mucho de su tiempo y amistad: Abril, Adán, Alfonso, Ángel, Armando, Blanca, David, Isaac, José, Raúl, Rodolfo y Rubí. Al Dr. Alfonso Lastras Martínez, por permitirme realizar el trabajo de tesis en el Instituto de Investigación en Comunicación Óptica y por el apoyo económico brindado, así como a todo el personal administrativo y técnico del instituto, quienes siempre tuvieron la disposición de apoyarme. Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo recibido durante todo el proceso de formación en el posgrado. A la UCSB por todos los recursos computacionales e infraestructura facilitados para la elaboración del trabajo. Al CIDESI por la infraestructura para la fabricación de los dispositivos.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El auge de la electrónica de semiconductores se encuentra entre las innovaciones más importantes del último medio siglo. Los circuitos integrados han revolucionado distintas áreas como la económica, tecnológica y social. Al día de hoy sería prácticamente imposible imaginar un mundo sin computadoras, teléfonos, internet, y muchos otros dispositivos que se han vuelto imprescindibles para la vida cotidiana. Grandes avances científicos, tales como los descubrimientos de las ondas gravitacionales [3] o el Bosón de Higgs [4], se verían imposibles de estudiar sin el uso de sistemas basados en circuitos integrados. En pocas palabras, los circuitos integrados forman parte de la base tecnológica de la actualidad.

Un circuito integrado es un dispositivo el cual esta constituido por una gran cantidad de circuitos electrónicos integrados en una placa de material semiconductor, generalmente silicio. La evolución de estos circuitos ha crecido enormemente, desde el primer circuito integrado que fue construido por Jack Kilby en 1958 en la empresa Texas Instruments, quien construyo un Flip-Flop empleando dos transistores hasta procesadores recientes, como el Apple M1 de 8 núcleos que contiene aproximadamente 16×10^9 transistores. Este aumento corresponde a un crecimiento anual aproximado del 53 % por año.

En el presente, un gran porcentaje de dispositivos electrónicos, son fabricados a partir de circuitos integrados en base a MOSFETs (*Metal Oxide Semiconductor Field Effect Transistors*), los cuales se pueden clasificar en dos tipos: nMOS y pMOS, dopados con material tipo n y tipo p, respectivamente. El MOSFET fue inventado por el ingeniero egipcio Mohamed M. Atalla y el ingeniero coreano Dawon Kahng en Bell Labs en 1959 y es el dispositivo semiconductor más común en circuitos digitales y analógicos. Fue el primer transistor realmente compacto que se podía miniaturizar y producir en masa para una amplia gama de aplicaciones. La escala y miniaturización de los MOSFETs ha impulsado el rápido crecimiento exponencial de la tecnología de semiconductores electrónicos desde la década de 1960, y ha permitido tener circuitos integrados de alta densidad [5].

La regla general que ha llegado a dominar la industria de circuitos integrados es la llamada *Ley de Moore*, que establece que el número de transistores en un circuito integrado se duplica cada dos años aproximadamente [6]. En 1965 Gordon E. Moore, cuantificó crecimiento de la nueva tecnología de semiconductores y la relación con la que disminuyen su tamaño. La mejora exponencial que describe la Ley de Moore, transformó las primeras computadoras domésticas de la década de 1970 en máquinas sofisticadas en las décadas de 1980 y 1990. A partir de eso, dio lugar al Internet de alta velocidad, teléfonos, vehículos y refrigeradores inteligentes, así como una gran variedad de dispositivos electrónicos. Esta observación ha sido tomada como un reto a cumplir por la industria de semiconductores y hasta el día de hoy ha sido cumplida. En la Figura 1.1 se muestra una gráfica correspondiente al crecimiento anteriormente descrito. Debido a la exactitud con la que la Ley de Moore predijo en el pasado el crecimiento de los circuitos integrados y a pesar de su complejidad, es visto como un método seguro para predecir las tendencias futuras.

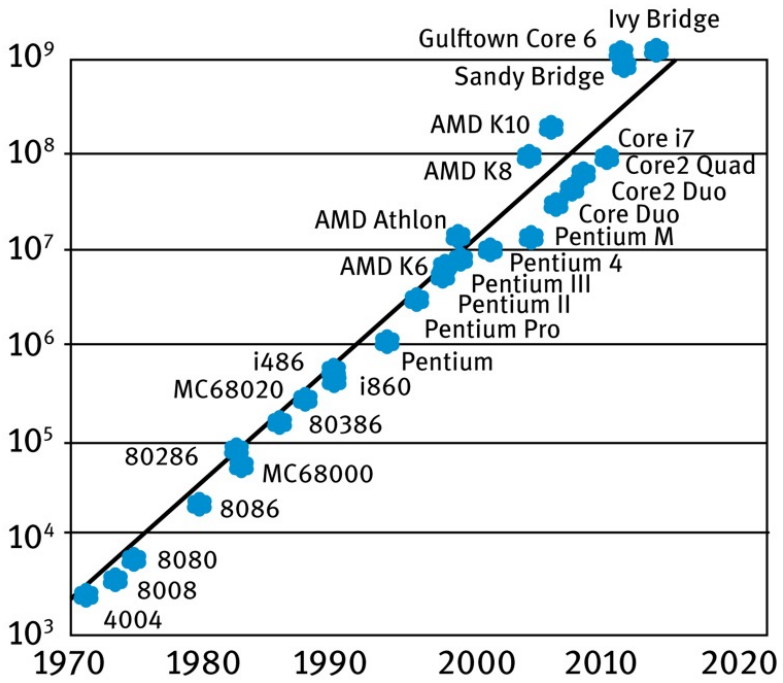


Figura 1.1: La Ley de Moore fue postulada por Gordon Moore, co-fundador de Intel, en 1965 y corregida en 1975. Hasta la actualidad sigue vigente [1].

En 1971 El CPU 4004 de Intel, contenía transistores con un *feature size* (ancho mínimo de un transistor para el proceso de fabricación) de 10 μm, sin embargo un procesador actual de teléfonos inteligentes como el Apple M1 bionic, es de 5 nm. El nivel de integración de los chips se ha clasificado en pequeña, mediana, gran y muy amplia escala. Los circuitos de integración a pequeña escala o SSI (*Small Scale Integration*), tienen menos de 10 compuertas, con aproximadamente media docena de transistores por compuerta. Los circuitos de integración de mediana escala o MSI (*Medium Scale Integration*), tienen hasta 1000 compuertas.

Los circuitos de integración a gran escala o LSI (*Large Scale Integration*), tienen hasta 10,000 compuertas. A partir de la década de 1980 en adelante, el termino integración a muy amplia escala o VLSI (*Very Large Scale Integration*) se usó para describir la mayoría de los circuitos integrados.

Los fabricantes introducen una nueva generación de procesos (o nodo tecnológico), cada 2 o 3 años y con esto aumentan la cantidad de transistores en la misma cantidad de área. Reducir las dimensiones de los transistores en un circuito integrado hace posible emplear mayor velocidad usando la misma potencia por unidad de área. Esta fue una observación de Robert Dennard de IBM, quien más tarde formalizó la llamada *Escala de Dennard*, la cual es la capacidad de reducir los voltajes de funcionamiento del dispositivo y escalar las frecuencias de reloj exponencialmente cada generación. Sin embargo, la Escala de Dennard llegó a su fin en 2004, debido a que la frecuencia era duplicada cada 34 meses aproximadamente y a pesar de que los transistores consumían menos energía, la enorme cantidad de ellos cambiando de estado a una velocidad muy alta, condujo a una crisis de eficiencia energética para lo diseños digitales y que plantea un desafío mas para el escalado de tecnología.

1.1. Transistores MOS

En la tecnología CMOS (*Complementary metal–oxide–semiconductor*) existen dos tipos de transistores, pFET y nFET (Transistores de efecto de campo de canal tipo n o p), y como su nombre lo indica, estos son controlados por un campo eléctrico. Una estructura MOS (Metal-Oxide-Semiconductor) tiene la forma tipo "*sandwich*" de tres materiales: un metal, un óxido, y un semiconductor. Cada transistor esta conformado por una puerta (*gate*) de material conductor, una capa aislante (típicamente de dióxido de silicio o de algún material dieléctrico de alto κ o constante dieléctrica que permitan una alta capacitancia en la puerta sin efectos de fuga de corriente) y finalmente de una capa de silicio que conforma el cuerpo (o *bulk*), el cual contiene zonas altamente dopadas con material tipo-p o tipo-n, que corresponden a la fuente (*source*) y drenador (*drain*) del transistor. En la Figura 1.2 se muestra la sección transversal típica de los transistores planares nMOS y pMOS.

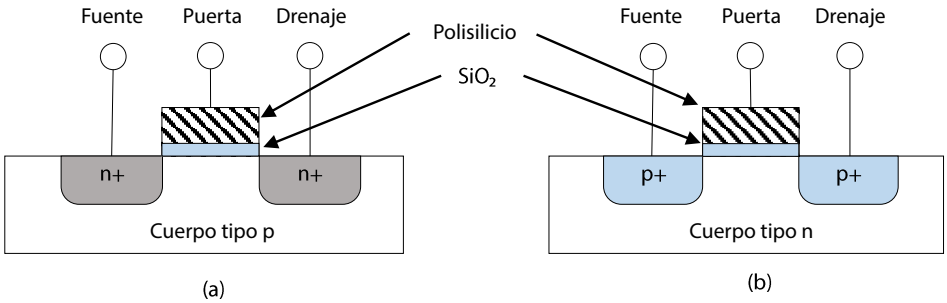


Figura 1.2: (a) Transistor nMOS y (b) transistor pMOS.

La puerta puede ser vista como una entrada para el control del transistor, debido a que afecta el flujo de corriente entre la fuente y el drenador. Considerando un transistor nMOS, el cuerpo generalmente esta aterrizado para que las uniones p-n que forman la fuente y el drenador con el cuerpo, estén inversamente polarizadas, por lo tanto, se puede decir que el transistor esta apagado. Si el voltaje de la puerta es incrementado, se crea un campo eléctrico, provocando que los electrones libres sean atraídos a la interfaz con el óxido de silicio. Si el voltaje en la puerta es lo suficientemente grande, los electrones superarán en número a los huecos, formado una delgada capa cerca de la interfaz, llamada canal (*channel*), que actúa como material semiconductor tipo-n. Por lo tanto, si se aplica una diferencia de potencial entre al fuente y el drenador, obtendremos una corriente entre dichos puntos. En este modo, se dice que el transistor esta en un estado de *encendido*. De manera práctica, los transistores pueden ser vistos como *switches* de encendido o apagado. Para el caso del nMOS, este se enciende cuando a la entrada de la puerta se encuentra un 1 lógico (o voltaje alto) y se apaga con un 0 lógico (o voltaje bajo). Para el caso del pMOS ocurre lo contrario: se enciende con 0 lógico y se apaga con 1 lógico.

1.2. ASICs en hardware de seguridad

Un circuito integrado de aplicación específica o ASIC (*Application-Specific Integrated Circuit*), es un chip diseñado para un propósito específico, creado para repetir la misma función eficientemente. Esto a diferencia de un chip de propósito general, los cuales pueden realizar distintas funciones pero de una manera menos eficiente. Cuando se intenta diseñar un sistema electrónico, generalmente se pueden obtener diversas soluciones técnicamente validas utilizando una combinación de chips o dispositivos, incluyendo componentes programables, analógicos, digitales y memorias, que no necesariamente son hechos a la medida. A veces puede ser interesante integrar solo las funcionalidades que necesita un chip dedicado a una aplicación y eso es lo que hace un ASIC: una combinación de funciones analógicas y digitales, en un solo chip. Un ASIC también podría ser la única solución posible cuando el sistema necesita cumplir ciertos requerimientos para lograr algún propósito especifico, por ejemplo, una alta eficiencia energética o un rendimiento muy alto.

Los ASICs son empleados en una gran variedad de dispositivos, comúnmente estos pueden formar parte de un sistema más complejo que un simple chip. Sus aplicaciones se encuentran en diversos dispositivos, tales como *wearables*, cámaras de vigilancia, sistemas de control para el hogar, lavadoras, drones y dispositivos biomédicos, entre otros. El rápido crecimiento de las aplicaciones digitales, el creciente número de usuarios de Internet y las amenazas tales como el acceso no autorizado a la información privada o robo de identidad, ha llevado a una creciente demanda de medidas de seguridad y dispositivos para proteger los datos del usuario ante cualquier amenaza. Dentro de la amplia gama de implementaciones de circuitos integrados, también existen chips que incorporan funciones de seguridad en hardware tales como sistemas de cifrado y generadores de números aleatorios, entre otros. Esto ocasionó que

la seguridad se convirta en un tema relevante para el diseño de ASICs capaces de prevenir ataques, robo de identidad o para la protección de información sensible [7].

En diversos trabajos como [8–10] se han desarrollado diseños de chips para la implementación de sistemas de cifrado. Cada diseño tiene distintos propósitos, como la reducción del consumo de potencia o el incremento de la frecuencia de operación. El interés por desarrollar mejores sistemas de cifrado implementados en tecnología CMOS, se debe en gran parte al continuo intento por extraer información sensible. Si esta tendencia continúa, más y mejores diseños serán necesarios para combatir este problema.

1.3. Sistemas de cifrado

Desde los inicios de la humanidad, la privacidad de la información ha sido importante en todos los aspectos de la vida. Puede ser aún más importante a medida que la tecnología continue controlando diversas operaciones en nuestro día a día. La tarea principal de la criptografía es posibilitar la comunicación de información con cierto nivel de seguridad, de tal manera que alguien ajeno no entienda su significado. Los sistemas criptográficos además de ser de gran utilidad en las comunicaciones, también son apropiados para proteger información sensible, como el contenido de dispositivos de almacenamiento o incluso archivos individuales.

Un algoritmo criptográfico es un método matemático empleado para cifrar información. Por lo general, estos sistemas emplean una o más claves (números o cadenas de caracteres) como condiciones iniciales o parámetros del algoritmo. En la literatura se pueden encontrar dos tipos de sistemas criptográficos, los simétricos (de clave secreta) y los asimétricos (de clave pública). Los sistemas simétricos convierten un mensaje en un texto cifrado del mismo tamaño y emplean una sola clave para cifrar o descifrar y son algoritmos utilizados para transferir grandes cantidades de información de modo seguro. Los sistemas asimétricos, a diferencia de los simétricos, usan claves distintas para cifrar (clave privada) y descifrar (clave pública) la información [11]. Estos sistemas suelen emplearse para distribuir las claves de los algoritmos simétricos.

Un ejemplo de los sistemas simétricos es el cifrado por bloques AES *(Advanced Encryption Standard)*, también conocido como *Rijndael*. Dicho sistema fue adoptado como un estándar de cifrado por el gobierno de los Estados Unidos en 2001 y fue anunciado por el NIST *(National Institute of Standards and Technology)*. Por otro lado, un ejemplo de un algoritmo criptográfico asimétrico es el sistema RSA (*Rivest, Shamir y Adleman*) empleado tanto para cifrar como para firmar digitalmente. El sistema DSA *(Digital Signature Algorithm)* es otro algoritmo criptográfico asimétrico, el cual es un estándar del Gobierno Federal de los Estados Unidos de América.

En el presente trabajo se realizó la implementación del sistema de Encriptación por Sin-

cronización de Autómatas Celulares o ESCA (*Encryption by Synchronization in Cellular Automata*). Como su nombre lo indica, dicho sistema está basado en la sincronización de autómatas celulares y su evolución a través de la regla 90 [12]. En [13, 14], se considera un sistema de cifrado compuesto por dos familias de permutaciones y un generador de números pseudoaleatorios o PRNG *(Pseudo-random Number Generator)*. En [15, 16] se propuso una implementación matricial y la adición de un módulo de pre-procesamiento que también es construido con la ayuda de los autómatas celulares, esto con la finalidad de conferir mayor seguridad contra los ataques de criptoanálisis. Por último, en [17] se presenta la implementación de una caja de sustitución o S-box *(Sustitution box)* basada en autómatas celulares, el cual es un componente no lineal en un sistema de cifrado, siendo este una parte importante en dichos sistemas. Dicha caja de sustitución formó parte del sistema final implementado en el presente trabajo.

1.4. Motivación

Debido a la poca infraestructura para el diseño e implementación de ASICs en México, el desarrollo de aplicaciones y la implementación de dichos circuitos integrados ha sido complicado para diversas universidades e instituciones públicas. Con el desarrollo del presente trabajo se propone demostrar la factibilidad y la diversidad de posibles aplicaciones para desarrollar dispositivos electrónicos en beneficio de la comunidad estudiantil y laboratorios de la Universidad Autónoma de San Luis Potosí, así como la introducción de este campo de las ciencias básicas que incluye el desarrollo de ASICs, con el fin de complementar o incluso incentivar la creación de nuevos cursos y especialidades dentro de la universidad. Para validar lo anteriormente mencionado, en el presente trabajo se plantea el diseño e implementación digital de dos ASICs: (1) el sistema de cifrado ESCA, un sistema simétrico basado en la sincronización de autómatas celulares capaz de cifrar bloques de 8 bits, y (2) un chip conformado por un oscilador en forma de anillo, un decodificador y dos sumadores. La implementación del sistema de cifrado se realizó empleando la tecnología CMOS 0.5 μm y celdas estándar para su diseño digital. Dicha implementación tiene la posibilidad de ser adaptable y reconfigurable, es decir, la ruta de datos podrá ser reconfigurada para la utilización de menos bloques, y a su vez, adaptar funciones mas pequeñas en ciertos bloques. El segundo ASIC se diseñó utilizando la tecnología CMOS de 5 μm, comenzando desde la elaboración de las celdas que conforman los módulos. El proceso tecnológico empleado para este chip, solo dispone de transistores nMOS, por lo que se emplearon diseños distintos a los conocidos en la tecnología CMOS.

Capítulo 2

Sistema de cifrado

El sistema de cifrado ESCA implementado en esta tesis consta de 7 módulos: Un generador de números pseudoaleatorios (PRNG), el módulo de cifrado, el de pre-procesamiento, la caja de sustitución y sus respectivos módulos inversos. En este capítulo se presenta una breve explicación de cómo son creados los bloques anteriormente mencionados, así como los fundamentos teóricos de los autómatas celulares y su sincronización, y la construcción de cada uno de los módulos del sistema. También se presenta una implementación híbrida del PRNG, creada para la optimización de su implementación digital y por último, se describe el sistema final para su implementación, así como sus modos de operación, entradas requeridas y la distribución de semillas.

2.1. Autómatas Celulares

Los autómatas celulares representan un modelo matemático de un sistema dinámico que evoluciona en pasos discretos. Éste se puede representar como una línea unidimensional que contiene casillas o celdas que pueden tomar distintos valores binarios y según transcurra el tiempo, el autómata evoluciona en función de sus estados anteriores y celdas vecinas [18]. También existen autómatas celulares con mayor número de estados por celdas, mayor cantidad de vecindades, plantillas más amplias y otros parámetros más. El conjunto de celdas de cada autómata logran una evolución de acuerdo a una determinada expresión matemática que es sensible a los estados de las células vecinas y se le conoce como regla local, que se refiere al algoritmo usado para calcular el estado de la siguiente celda, un ejemplo se muestra en la Figura 2.1, donde el estado de la casilla **A**, depende de los estados de las casillas **B** y **C**.

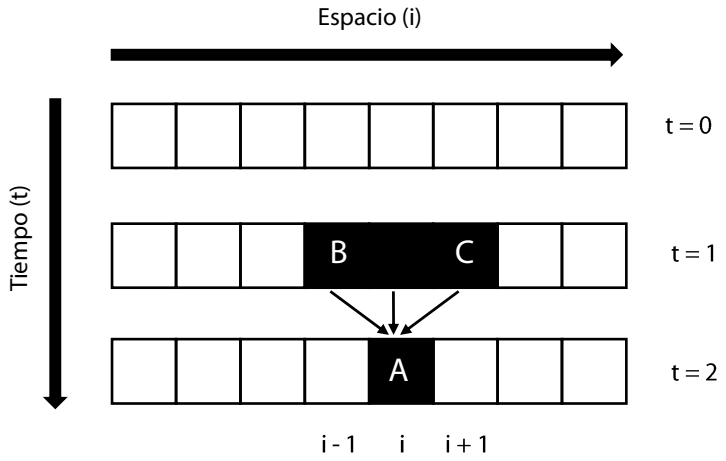


Figura 2.1: Evolución de un autómata celular a través de la regla 90, donde el estado de la casilla **A** depende del estado de las casillas **B** y **C**.

2.1.1. Sincronización de Autómatas Celulares

La sincronización de autómatas celulares ocurre cuando después de cierto número de iteraciones, dos sistemas llegan a un estado igual, de tal manera que la diferencia de ambos sea un vector nulo. Un par de autómatas celulares, donde la regla local corresponde a la regla 90, sincroniza si cada par consecutivo de las coordenadas acopladas son separadas por un bloque de $N = 2^n - 1$ sitios desacoplados [12]. El acoplamiento de los autómatas celulares ocurre cuando las coordenadas acopladas de un sistema son replicadas en otro. Por ejemplo, en la Figura 2.2 se muestran dos sistemas evolucionados de tamaño $N = 7$, donde se copiaron las coordenadas acopladas del autómata manejador al autómata de réplica y su evolución después de **N** pasos con la regla 90 dio lugar al mismo estado.

2.1.2. Unidad encriptadora básica

La implementación de las permutaciones, un PRNG y el pre-procesado, básicamente consisten en algoritmos de un solo tiempo. Para la generación de éstos, puede emplearse la unidad encriptadora básica (BUC), descrita en [19]. En la figura 2.3 se muestra la estructura de este bloque básico, donde las coordenadas 0 y N+1 se consideran como coordenadas acopladas, que son asignadas externamente en cada tiempo t. La unidad se forma a partir de la secuencia inicial $x^0 = (x_0^0, x_1^0, x_2^0, ..., x_{N-1}^0, x_N^0, x_{N+1}^0)$, que evoluciona del tiempo $t = 0$ al tiempo $t = N$ siguiendo la regla local 90.

En el bloque básico, se destacan sub-bloques con lineas punteadas, para distinguir palabras especificas. Las palabras **x** y **y** corresponden a las semillas iniciales del sistema de cifrado, las palabras **m** y **c**, corresponden al texto plano y cifrado, respectivamente, por último, la palabra

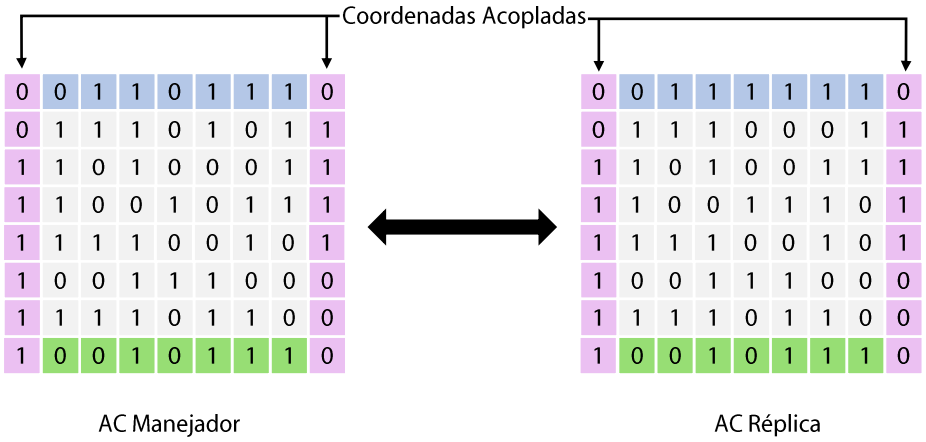


Figura 2.2: Sincronización de dos autómatas celulares donde el estado del autómata de réplica llega al mismo estado que el autómata manejador después de $N = 7$ pasos.

t corresponde al número pseudo-aleatorio. A continuación se describen las permutaciones para el cifrado (Ψ_x), descifrado (Φ_x) y PRNG ($h(x, y)$).

- Permutación $m = \Phi_x(c)$: Para obtener la palabra **m**, se hace evolucionar el autómata hacia adelante en el tiempo, teniendo como entrada las palabras **x** y **c**. La independencia en **t** es resultado de la sincronización [14, 20, 21].
- Permutación $c = \Psi_x(m)$: Para obtener la palabra **c**, se hace evolucionar el autómata hacia atrás en el tiempo, esta función depende de las entradas **x** y **m**.
- Función $t = h(x, y)$: Para obtener la palabra t, se hace evolucionar el autómata hacia atrás en el tiempo, utilizando como entradas las palabras **x** y **y**.

Con la ayuda de la BUC, es posible la generación de los algoritmos de un solo tiempo. Esto se realiza, observando la dependencia de cada bit de las palabra que se quiere obtener. A continuación se muestra dos ejemplos para la obtención del bit 4 de la palabra **t** y del bit 7 de la palabra **m** de tamaño de 15 bits, haciendo reducciones de las dependencias utilizando álgebra booleana.

Obtención del bit t_4 de la figura 2.4:

$$\begin{aligned} t_4 &= a_4 \oplus t_2 \\ &= a_2 \oplus a_{18} \oplus y_1 \oplus a_2 \\ &= y_1 \oplus y_3 \oplus a_{32} \\ &= y_1 \oplus y_3 \oplus y_5 \oplus x_4 \end{aligned} \tag{2.1}$$

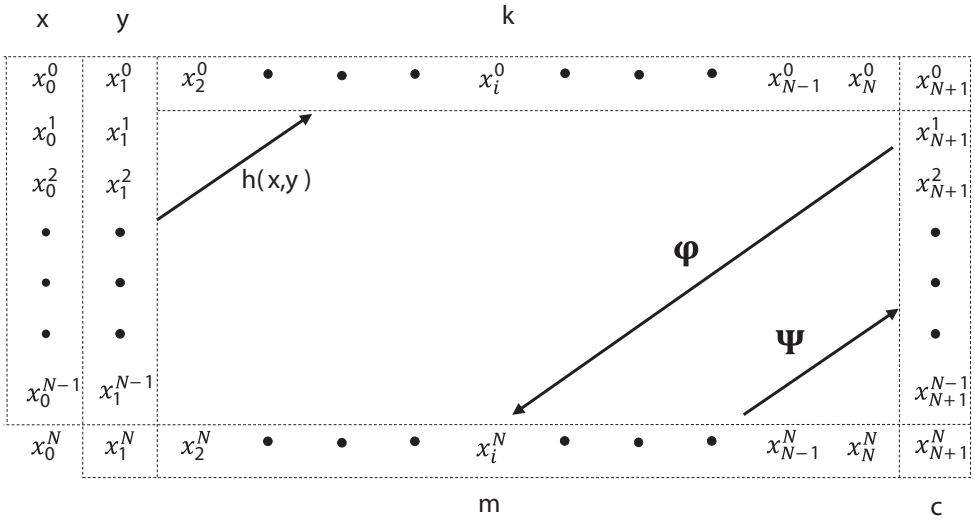


Figura 2.3: Unidad encriptadora básica.

Obtención del bit m_7 de la Figura 2.5:

$$
\begin{aligned}
m_7 &= a_{201} \oplus a_{203} \\
&= a_{185} \oplus a_{187} \oplus a_{187} \oplus a_{189} \\
&= a_{169} \oplus a_{171} \oplus a_{173} \oplus a_{175} \\
&= a_{153} \oplus a_{155} \oplus a_{155} \oplus a_{157} \oplus a_{157} \oplus a_{159} \oplus a_{159} \oplus a_{161} \\
&= a_{137} \oplus a_{139} \oplus a_{145} \oplus a_{147} \\
&= y_{10} \oplus a_{123} \oplus a_{123} \oplus a_{125} \oplus a_{129} \oplus a_{131} \oplus a_{131} \oplus a_{133} \\
&= x_9 \oplus a_{107} \oplus a_{109} \oplus a_{111} \oplus a_{113} \oplus a_{115} \oplus a_{117} \oplus a_{119} \\
&= x_9 \oplus y_8 \oplus a_{93} \oplus a_{93} \oplus a_{95} \oplus a_{95} \oplus a_{97} \oplus a_{97} \oplus a_{99} \oplus a_{99} \oplus a_{101} \oplus a_{101} \oplus a_{103} \oplus a_{103} \oplus a_{105} \\
&= x_9 \oplus x_7 \oplus a_{77} \oplus a_{89} \oplus c_7 \\
&= x_9 \oplus x_7 \oplus y_6 \oplus a_{63} \oplus a_{73} \oplus a_{75} \oplus c_7 \\
&= x_9 \oplus x_7 \oplus x_5 \oplus a_{47} \oplus a_{47} \oplus a_{49} \oplus a_{57} \oplus a_{59} \oplus a_{59} \oplus c_7 \oplus c_5 \\
&= x_9 \oplus x_7 \oplus x_5 \oplus c_7 \oplus c_5 \oplus a_{33} \oplus a_{35} \oplus a_{41} \oplus a_{43} \\
&= x_9 \oplus x_7 \oplus x_5 \oplus c_7 \oplus c_5 \oplus a_{17} \oplus a_{19} \oplus a_{19} \oplus a_{21} \oplus a_{25} \oplus a_{27} \oplus a_{27} \oplus a_{29} \\
&= x_9 \oplus x_7 \oplus x_5 \oplus c_7 \oplus c_5 \oplus y_2 \oplus a_3 \oplus a_5 \oplus a_7 \oplus a_9 \oplus a_{11} \oplus a_{13} \oplus a_{15} \\
&= x_9 \oplus x_7 \oplus x_5 \oplus c_7 \oplus c_5 \oplus x_1 \oplus t_1 \oplus t_1 \oplus t_3 \oplus t_3 \oplus t_5 \oplus t_5 \oplus t_7 \oplus t_7 \oplus t_9 \oplus t_9 \oplus t_{11} \oplus t_{11} \\
&\quad \oplus t_{13} \oplus t_{13} \oplus c_1 \\
&= x_1 \oplus x_5 \oplus x_7 \oplus x_9 \oplus c_1 \oplus c_5 \oplus c_7
\end{aligned}
\tag{2.2}
$$

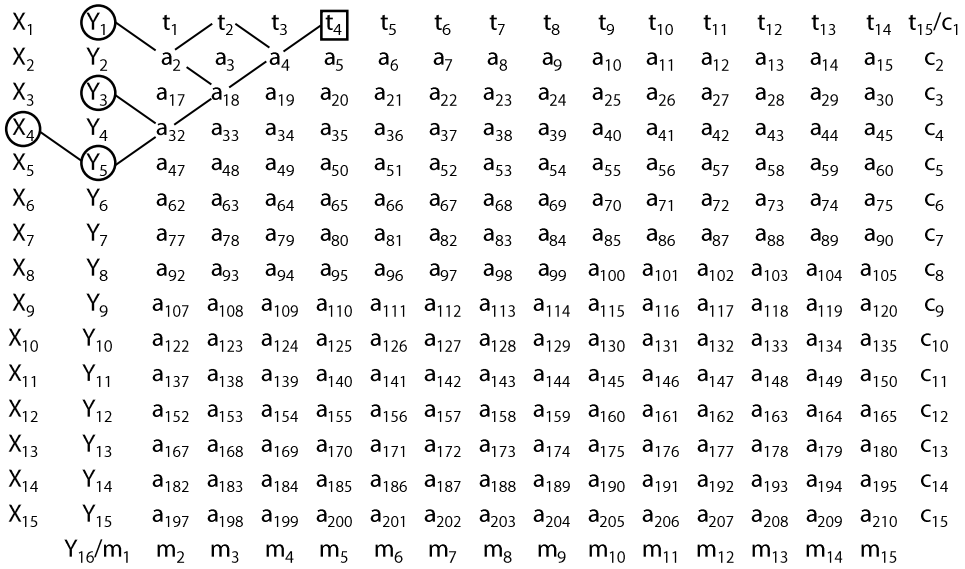


Figura 2.4: Reducción del bit t_4 de la palabra $\mathbf{t}$.

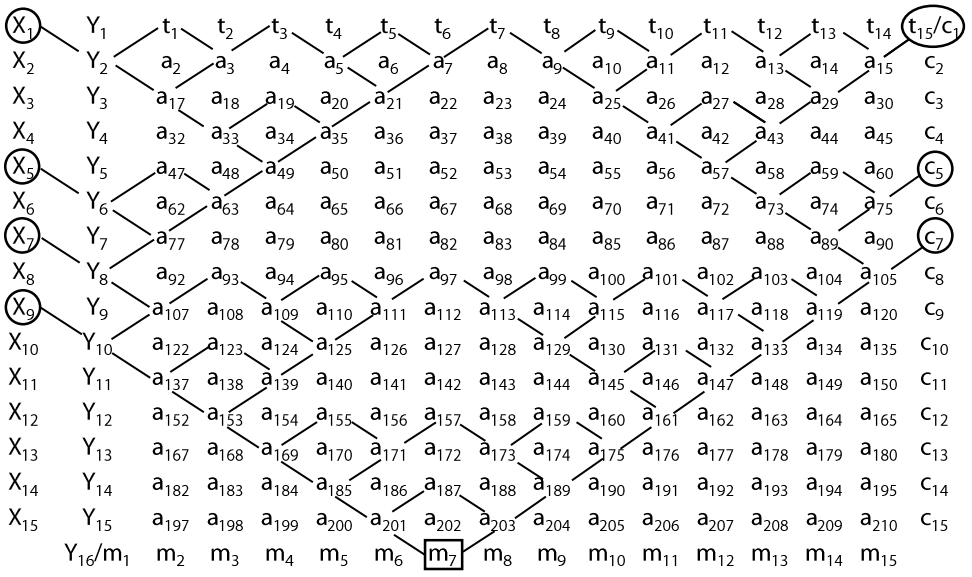


Figura 2.5: Reducción del bit m_7 de la palabra $\mathbf{m}$.

2.2. Sistema ESCA

El sistema de Encriptación por Sincronización de Autómatas Celulares (ESCA) es un algoritmo criptográfico simétrico, capaz de cifrar bloques de 8 bits. El sistema esta compuesto por 7 módulos: un generador de números pseudo-aleatorios que genera las llaves (números pseudo-aleatorios o llaves), las permutaciones "Ψ" y "Φ", que calculan el texto cifrado o descifrado respectivamente, las cajas de sustitución S-box y S-box^{-1}, las cuales son transformaciones lineales y por último, los módulos de Pre-procesado y Pre-procesado^{-1}, cuyo propósito es hacer al sistema más seguro ante un ataque. A continuación se presenta una breve descripción de los bloques ya mencionados que conforman el sistema implementado:

2.2.1. Generador de números pseudo-aleatorios

El generador de números pseudo-aleatorios (PRNG), es el bloque encargado de generar las llaves para los módulos de cifrado y descifrado, cuyas propiedades se asemejan a las propiedades de los números aleatorios. Esta secuencia esta determinada por un valor inicial o semilla. En [14] se presenta el diseño de un PRNG basado en autómatas celulares y en [16] su implementación matricial, se deja a cargo del lector interesado en verificar el diseño del PRNG.

En la Figura 2.6 se muestra el diagrama del PRNG, el cual requiere dos semillas como entradas, $\mathbf{x}$ y $\mathbf{y}$ de N y $(N+1)$ bits respectivamente, donde N es de tamaño de 2^{k-1}. Estas semillas son empleadas como condición inicial para la generación de los números pseudoaleatorios, los cuales son denotados como las llaves t de tamaño de N bits, que corresponden a la salida del bloque. Para la generación del siguiente número pseudoaleatorio, la salida $\mathbf{t}$ es retroalimentada a la semilla $\mathbf{x}$ y la semilla $\mathbf{x}$ anterior toma el lugar de la nueva semilla $\mathbf{y}$, para completar la longitud de $(N+1)$ que conforman la semilla $\mathbf{y}$, se concatena el bit menos significativo de la semilla $\mathbf{y}$ anterior, en la posición del bit más significativo de la semilla $\mathbf{y}$ actual. Este proceso permanece iterando para la generación de la secuencia de números pseudoaleatorios. Para realizar el proceso de descifrado, es necesario conocer la semilla empleada para generar las mismas llaves que se utilizaron en el proceso de cifrado debido a que el sistema es un algoritmo simétrico.

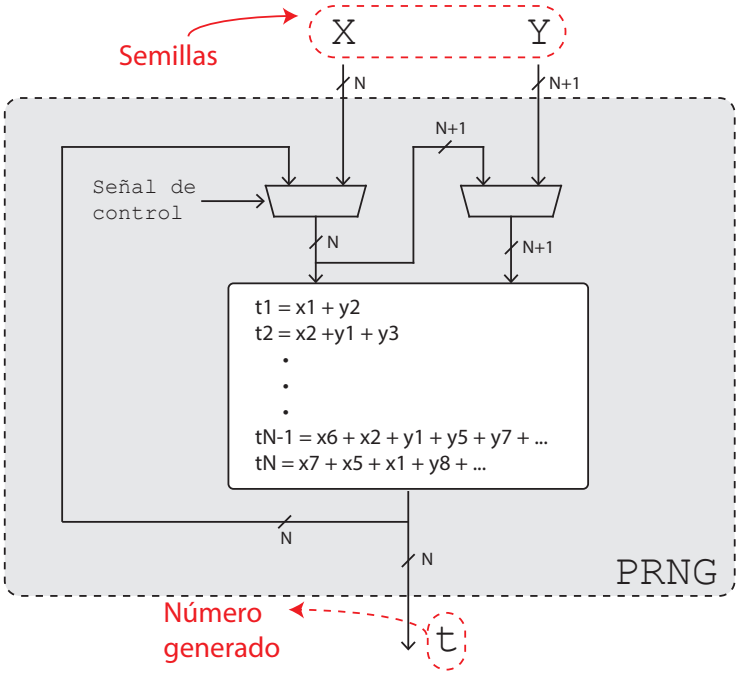


Figura 2.6: Diagrama del bloque generador de números pseudoaleatorios.

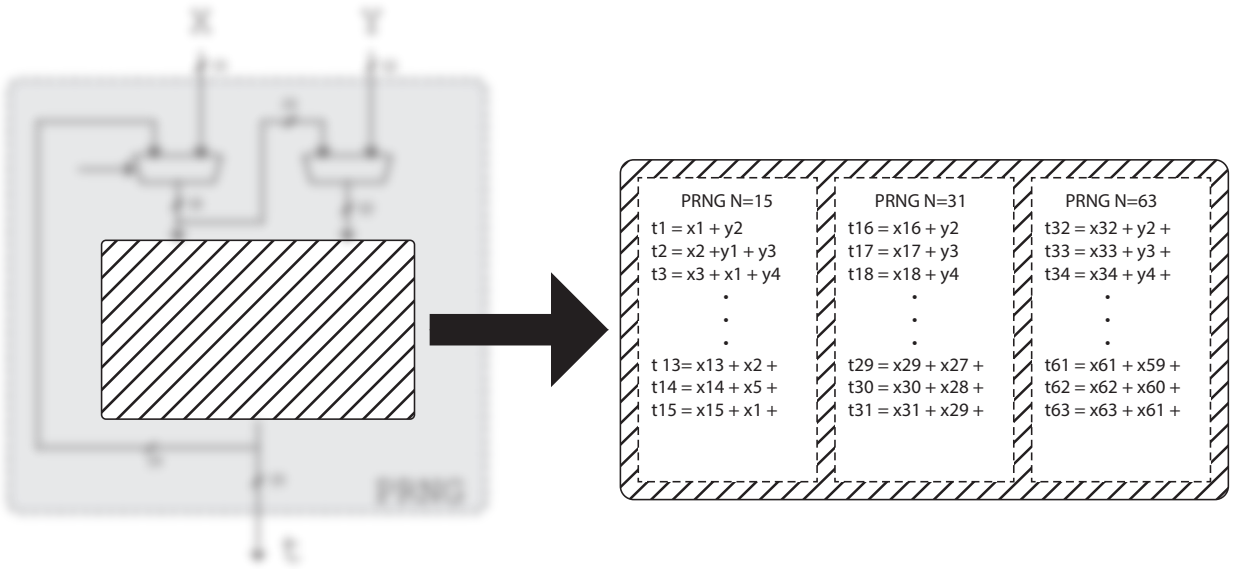


Figura 2.7: Generador de números pseudo-aleatorios implementado para 3 tamaños distintos de llave.

Para la elaboración del módulo PRNG, se tomó en cuenta que la longitudes de llaves que aseguran un rango de seguridad adecuado, estén en el rango de 128 a 256. En particular, el

tamaño de 128 generalmente se considera suficiente para la mayoría de las aplicaciones [22]. En consecuencia, se tomo $N = 127$ como la longitud mínima para el tamaño de las llaves generadas por el PRNG. A pesar de la escalabilidad del PRNG para crecer el tamaño de los números pseudo-aletorios, sus posibles aplicaciones podrían tener problemas en cuanto a la eficiencia en la velocidad o el tamaño de la implementación. Sin embargo, es posible hacer uso de la arquitectura del PRNG para adaptar la estructura del módulo y que éste sea capaz de generar números mas pequeños.

Dicha implementación puede verse como 3 módulos distintos (en uno solo) cuya longitud de llave corresponde a $N = 127$, $N = 63$ y $N = 31$, donde cada opción genera una llave de la longitud correspondiente al tamaño de N o $(2^k/2)-1$, como se muestra en la Figura 2.7. Esta implementación es posible debido a que el incremento de la longitud en la llave, no modifica la información de bits menos significativos, dicho de otro modo, aumentar la longitud de llave, únicamente implica la concatenación del resto del algoritmo. Por lo que es necesaria una señal de control, la cual indique qué tamaño de llave se desea usar y la funcionalidad del módulo. Esto puede verse fácilmente en las expresiones booleanas para el módulo de $N = 31$, de las cuales, las primeras 15 expresiones corresponden a las del módulo de $N = 15$, las cuales son mostradas a continuación.

Expresiones del algoritmo de un solo tiempo para $N = 31$:

$t_1 = x_1 \oplus y_2$

$t_2 = x_2 \oplus y_1 \oplus y_3$

$t_3 = x_1 \oplus x_3 \oplus y_4$

$t_4 = x_4 \oplus y_1 \oplus y_3 \oplus y_5$

$t_5 = x_5 \oplus x_3 \oplus x_1 \oplus y_2 \oplus y_6$

$t_6 = x_6 \oplus x_2 \oplus y_1 \oplus y_5 \oplus y_7$

$t_7 = x_7 \oplus x_5 \oplus x_1 \oplus y_8$

$t_8 = x_8 \oplus y_1 \oplus y_5 \oplus y_7 \oplus y_9$

$t_9 = x_9 \oplus x_7 \oplus x_5 \oplus x_1 \oplus y_2 \oplus y_6 \oplus y_{10}$

$t_{10} = x_{10} \oplus x_6 \oplus x_2 \oplus y_1 \oplus y_3 \oplus y_5 \oplus y_9 \oplus y_{11}$

$t_{11} = x_{11} \oplus x_9 \oplus x_5 \oplus x_3 \oplus x_1 \oplus y_4 \oplus y_{12}$

$t_{12} = x_{12} \oplus x_4 \oplus y_1 \oplus y_3 \oplus y_9 \oplus y_{11} \oplus y_{13}$

$t_{13} = x_{13} \oplus x_{11} \oplus x_9 \oplus x_3 \oplus x_1 \oplus y_2 \oplus y_{10} \oplus y_{14}$

$t_{14} = x_{14} \oplus x_{10} \oplus x_2 \oplus y_1 \oplus y_9 \oplus y_{13} \oplus y_{15}$

$t_{15} = x_{15} \oplus x_{13} \oplus x_9 \oplus x_1 \oplus y_{16}$

Expresiones del algoritmo de un solo tiempo para $N = 15$:

$$t_1 = x_1 \oplus y_2$$

$$t_2 = x_2 \oplus y_1 \oplus y_3$$

$$t_3 = x_1 \oplus x_3 \oplus y_4$$

$$t_4 = x_4 \oplus y_1 \oplus y_3 \oplus y_5$$

$$t_5 = x_5 \oplus x_3 \oplus x_1 \oplus y_2 \oplus y_6$$

$$t_6 = x_6 \oplus x_2 \oplus y_1 \oplus y_5 \oplus y_7$$

$$t_7 = x_7 \oplus x_5 \oplus x_1 \oplus y_8$$

Para el caso de $N = 31$ se emplearon las mismas expresiones de $N = 15$ y la adición del resto del algoritmo. Las expresiones booleanas para el resto de módulos de diferente tamaño se muestran en A.

2.2.2. Cifrado y Descifrado

Estos bloques corresponden a las dos familias de permutaciones, Ψ y Φ, para cifrar y descifrar, respectivamente. En la Figura 2.8 se muestra el diagrama del bloque de cifrado. Al igual que en el bloque de PRNG, el cifrador y descifrador se pueden implementar de manera práctica con funciones booleanas. Dicho bloque tiene como entradas, el texto a cifrar denotado como **s** de 8 bits y la llave **t**, que puede ser de tamaño de $N = 15, 31$ o 63 bits. Como salida del bloque se obtiene el dato cifrado denotado como **c** de 8 bits.

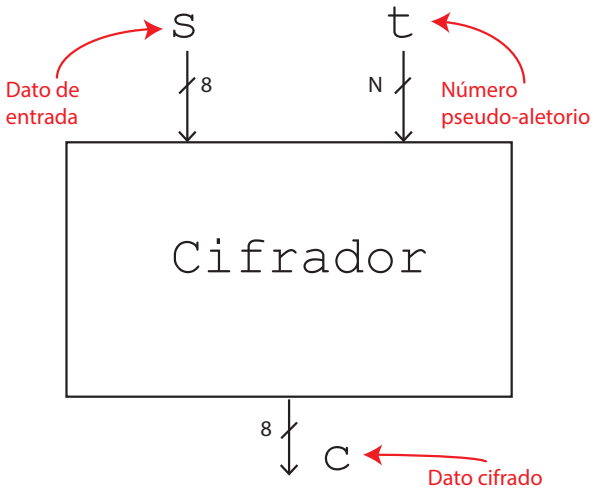


Figura 2.8: Diagrama del bloque de cifrado.

De manera práctica, es posible implementar estos bloques con el algoritmo de un solo tiempo, pero a diferencia del PRNG no es posible reutilizar los bloques que requieren una semilla de menor tamaño para crear un solo módulo el cual sea capaz de cumplir la función de tres distintos, por lo que es necesario emplear 3 módulos diferentes de cifrado y descifrado que requieren una longitud de llave distinta. A continuación se muestran las expresiones booleanas para los módulos de cifrado y descifrado para una llave de tamaño de $N = 15$.

Cifrador:

$m_1 = x_{15} \oplus x_{13} \oplus x_9 \oplus x_1 \oplus c_1$

$m_2 = x_{14} \oplus x_{10} \oplus x_2 \oplus c_2$

$m_3 = x_{13} \oplus x_{11} \oplus x_9 \oplus x_3 \oplus x_1 \oplus c_1 \oplus c_3$

$m_4 = x_{12} \oplus x_4 \oplus c_4$

$m_5 = x_{11} \oplus x_9 \oplus x_5 \oplus x_3 \oplus x_1 \oplus c_1 \oplus c_3 \oplus c_5$

$m_6 = x_{10} \oplus x_6 \oplus x_2 \oplus c_2 \oplus c_6$

$m_7 = x_9 \oplus x_7 \oplus x_5 \oplus x_1 \oplus c_1 \oplus c_5 \oplus c_7$

$m_8 = x_8 \oplus c_8$

Descifrador:

$c_1 = x_{15} \oplus x_{13} \oplus x_9 \oplus m_1 \oplus m_1$

$c_2 = x_{14} \oplus x_{13} \oplus x_2 \oplus m_2$

$c_3 = x_{14} \oplus x_{13} \oplus x_3 \oplus m_1 \oplus m_3$

$c_4 = x_{12} \oplus x_4 \oplus m_4$

$c_5 = x_{13} \oplus x_5 \oplus m_3 \oplus m_5$

$c_6 = x_{14} \oplus x_6 \oplus m_2 \oplus m_6$

$c_7 = x_{15} \oplus x_7 \oplus m_1 \oplus m_3 \oplus m_5 \oplus m_7$

$c_8 = x_8 \oplus m_8$

Las expresiones booleanas del cifrador y descifrador con distinto tamaño de llave, se muestran en los Apéndices A.1 a A.6.

2.2.3. Pre-procesado

El bloque de pre-procesado fue integrado al sistema de cifrado junto con la aproximación matricial del mismo en [15], con la finalidad de transformar el texto original a una forma ilegible antes de que sea transformado por la S-box y así hacer mas seguro el sistema. Este bloque requiere como entrada el texto a cifrar o texto plano denotado como **m** y una semilla denotada como **z**, de 8 y 9 bits respectivamente. Como salida se obtiene el texto pre-procesado denotado como $\hat{\mathbf{m}}$ de 8 bits.

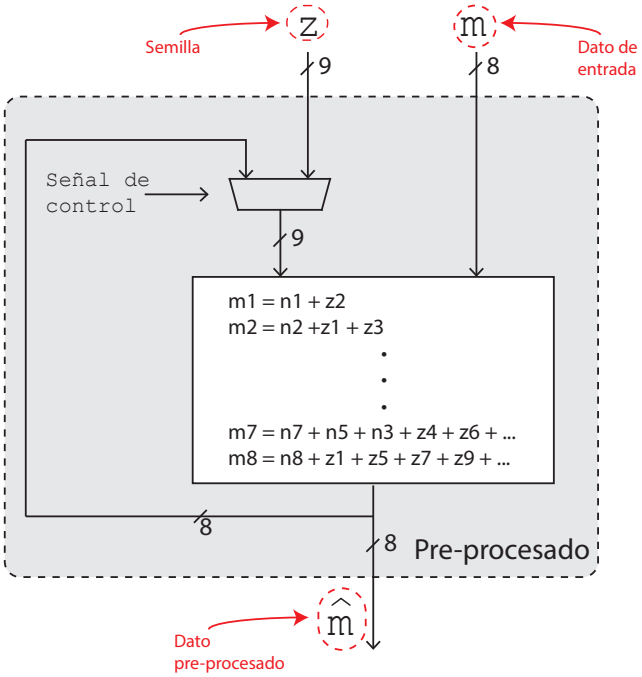


Figura 2.9: Diagrama del pre-procesamiento de datos.

Similar al módulo de PRNG, la salida del pre-procesado es retroalimentada a la semilla, es decir, después de la primera iteración los 8 bits preprocesados toman el lugar de los 8 bits menos significativos de la semilla y el bit mas significativo toma el lugar del bit menos significativo de la semilla anterior. Al igual que los módulos anteriores, el pre-procesado puede ser implementado con 8 expresiones booleanas que se muestran a continuación.

$$\hat{m}_1 = m_1 \oplus z_2$$

$$\hat{m}_2 = m_2 \oplus z_1 \oplus z_3$$

$$\hat{m}_3 = m_1 \oplus m_3 \oplus z_2 \oplus z_4$$

$$\hat{m}_4 = m_4 \oplus z_1 \oplus z_3 \oplus z_5$$

$\hat{m}_5 = m_5 \oplus m_3 \oplus z_2 \oplus z_4 \oplus z_6$

$\hat{m}_6 = m_6 \oplus m_2 \oplus z_3 \oplus z_5 \oplus z_7$

$\hat{m}_7 = m_7 \oplus m_5 \oplus m_3 \oplus m_1 \oplus z_4 \oplus z_6 \oplus z_8$

$\hat{m}_8 = m_8 \oplus z_1 \oplus z_5 \oplus z_7 \oplus z_9$

De manera práctica, el proceso inverso posee la misma arquitectura que el pre-procesado, con la diferencia de que la retroalimentación proviene del dato de entrada y que las expresiones booleanas son distintas, dichas funciones se muestran en los Apéndices A.8 y A.9.

2.2.4. S-box

El bloque de S-box confiere la propiedad no lineal al sistema de cifrado, donde cada byte de entrada es remplazado por el resultado de aplicar la función de la S-box a ese byte. Los valores de entrada son $\hat{\mathbf{m}}$ de 8 bits que denota el dato de entrada, $\mathbf{c}$ de 8 bits que denota una constante para su posterior adición y el dato de salida se denota como $\mathbf{s}$ de 8 bits, como se muestra en la Figura 2.10.

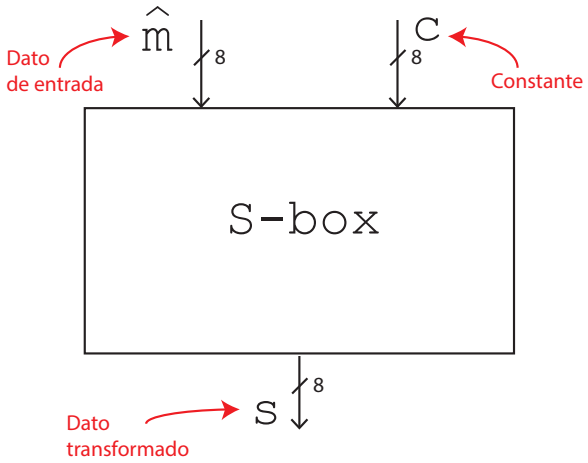


Figura 2.10: Diagrama de la caja de sustitución.

Esta función no lineal involucra encontrar el inverso de un número de 8 bits, considerándolo como un elemento de los campos de Galois $GF(2^8)$. En [17] se presenta la generación de la S-box implementada en el sistema final y esta dada por la siguiente expresión:

$$f(x) = \begin{cases} (K \times x^t)^{-1} & \text{si } x \neq 0 \\ 0 & \text{si } x = 0, \end{cases} \tag{2.3}$$

donde x^t representa el dato de entrada como un vector transpuesto y k representa una matriz cuadrada. En [17] se presenta la generación de la matriz k, la cual esta basada en los autómatas celulares y su evolución con la regla 90. Esta matriz tiene dimensiones de 8×8 y esta compuesta por dos matrices, K_L (Ec. 2.4) y K_R (Ec. 2.5), las cuales constituyen la parte izquierda y derecha de la matriz k, respectivamente. Cada una de ellas tiene dimensiones 8×4. Para generar la matriz k (Ec. 2.6), se comienza con la implementación de k_L, para esto se toma el vector inicial $a = [0, 0, 0, 1, 0, 0, 0, 0]$ para la primer fila y se hace evolucionar el autómata con la regla 90 y con ceros como condiciones de frontera, para generar 3 filas mas. Finalmente se rota la matriz generada 90 grados en dirección de las manecillas del reloj para obtener k_L:

$$k_L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{2.4}$$

Para generar K_R, se multiplica la matriz K_L por una matriz de permutación fija P:

$$k_R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \tag{2.5}$$

La matriz K resultante es la siguiente:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{2.6}$$

A la ecuación 2.6 se añadió la suma de una constante denominada como **C**, esta representa un vector cuya finalidad es que al realizar la suma módulo 2 con el resultado de $(K \times x^t)^{-1}$, de manera que la caja de sustitución no tenga puntos fijos o transformaciones complementarias [23], dando como resultado $(K \times x^t)^{-1} + C$. Las constantes válidas para los fines descritos anteriormente son las siguientes:

1	3	14	1F	23	24	25	40
44	47	55	57	6D	76	7E	81

Tabla 2.1: Constantes válidas para la suma en la S-box y su inversa correspondiente.

Como se menciona en [17], la matriz K es generada a partir de un vector inicial, en este caso, se tomó el vector $a = [0, 1, 0, 0, 1, 1, 0, 0]$ debido a que la S-box y S-box^{-1} generadas a partir de este vector, tienen la menor cantidad de puntos fijos y complementarios. De manera similar la S-box inversa puede ser implementada como una tabla de consulta, pero a diferencia de la S-box, la operación de suma mod. 2 con la constante, debe realizarse primero, antes de hacer la sustitución con el valor correspondiente de la tabla. Las tablas de consulta correspondientes a la S-box y S-box inversa se muestra en los Apéndices A.10 y A.11.

2.3. Resumen

El sistema ESCA es un algoritmo criptográfico simétrico reconfigurable y adaptable el cual opera con datos de 8 bits. El tamaño de la llave para el sistema es igual a la suma de longitud de las semillas de los módulos mas 8 bits que corresponden al byte de configuración. El tamaño de las semillas por módulo se describe a continuación:

- PRNG/Cifrado: Como se mencionó en 2.2.2, hay 3 modos de operación para el generador de números pseudoaletorios y los bloques de cifrado/descifrado. Cada modo indica el tamaño de la llave que se usará y configura el módulo de PRNG para su correcto funcionamiento. El modo 1 corresponde a $N = 31$, el modo 2 a $N = 63$ y el modo 3 a $N = 127$.
- El pre-procesamiento y su respectivo inverso emplean una semilla de 9 bits.
- Los módulos de S-box y Sbox^{-1} emplean 4 bits para decodificar la constante.

Dicho lo anterior, el tamaño de la llave puede variar desde 4 hasta 140 bits como se muestra en la Figura 2.11. El sistema posee 14 distintas configuraciones sin contar la posibilidad de no realizar ninguna transformación a los datos.

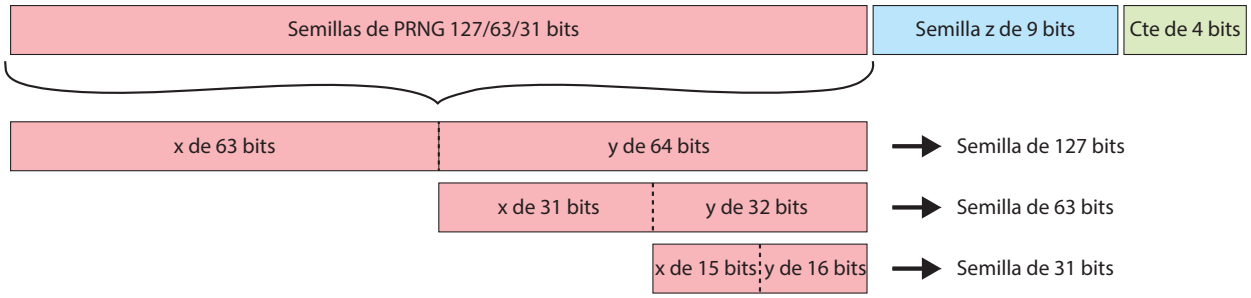


Figura 2.11: Distribución de semillas en un registro único.

La información fluye a través de bloques individuales que conforman la arquitectura del sistema. El proceso de descifrado del sistema ESCA es básicamente el inverso de cada módulo a excepción del generador de llaves (Descifrado, S-box inversa y Pre-procesado inverso), en orden inverso. El orden de las 3 transformaciones del sistema ESCA es el siguiente:

1. Pre-procesado.

2. S-box.

3. Cifrado y PRNG de forma paralela.

Como se muestra en la figura 2.12, la salida de cada modulo esta conectada a un multiplexor, junto con la señal de entrada del mismo modulo. Con esto, es posible enviar la información sin que este bloque la transforme. Esta estructura permite seleccionar que módulos se emplearon en el proceso de cifrado, brindando flexibilidad al sistema.

El proceso de descifrado (Figura 2.12) del sistema ESCA se compone de la permutación Φ, pre-procesado inverso, S-box inversa y el generador de números pseudo-aleatorios. El flujo de datos comienza en la permutación Φ (con llaves generados por el PRNG), seguido de la S-box inversa y termina con el pre-procesado inverso. Al igual que en el proceso de cifrado, hay multiplexores a la salida de los bloques para realizar un *by-pass* de la información. Debido a que el sistema ESCA es un sistema de cifrado simétrico, es necesario introducir la misma llave con la cual se cifro, para obtener el texto original, ademas de la misma configuración.

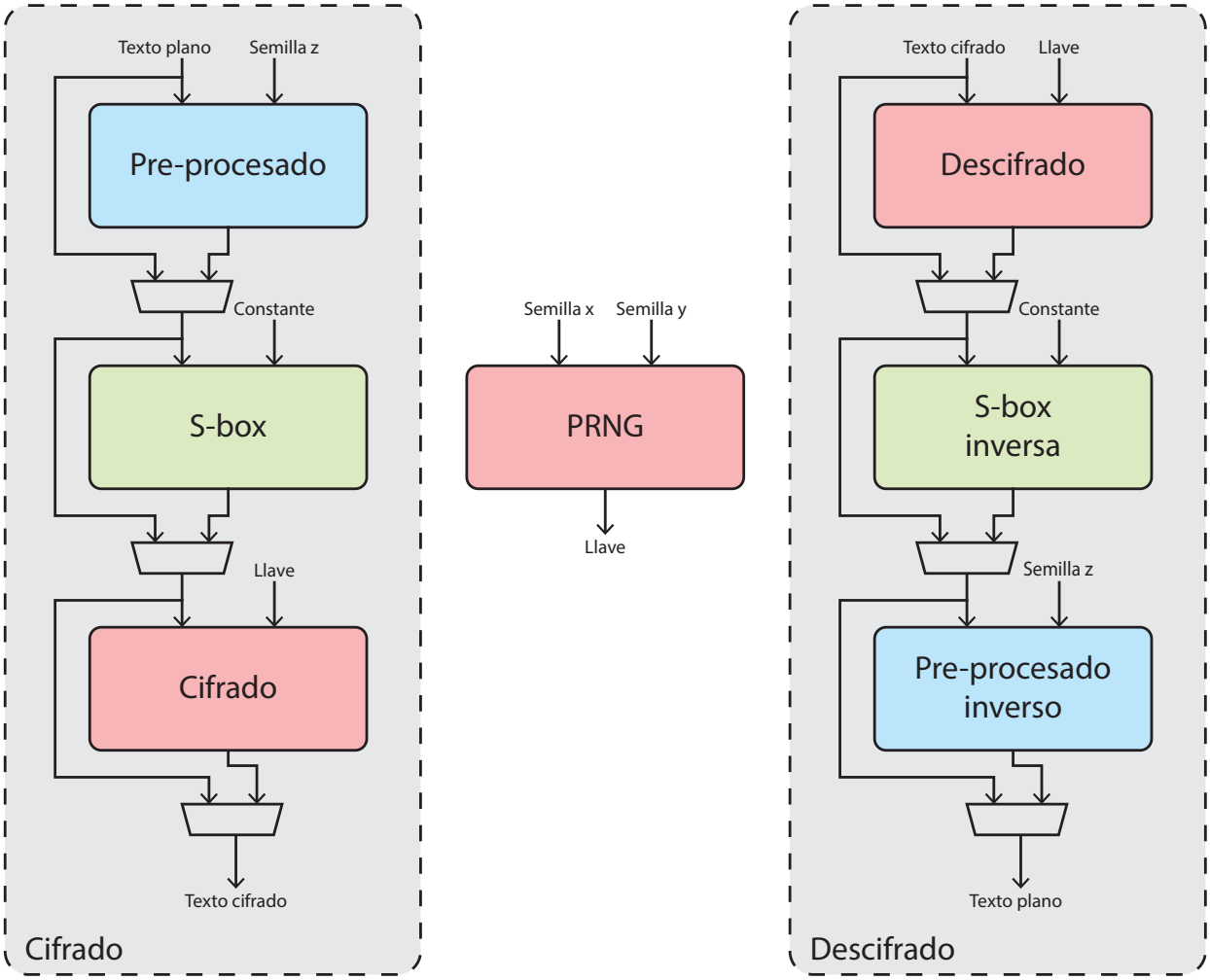


Figura 2.12: Operaciones de cifrado y descifrado del sistema ESCA.

Capítulo 3

Tecnología CMOS

En este capítulo se examinará el funcionamiento de los transistores MOSFETs comenzando desde una unión P-N hasta el modelo Shockley empleado para el estudio de los nFETs y pFETs. Enseguida se introducirá a la lógica digital donde se inspeccionará el tipo de implementaciones digitales que se pueden realizar con la tecnología CMOS. Por ultimo se implementará un sumador completo de 8 bits, para comprender el flujo de diseño para circuitos en tecnología VLSI. La información de la tecnología CMOS que se presenta en este capítulo está basada en el texto *CMOS VLSI design: a circuits and systems perspective* [2].

3.1. Transistores MOS

Como se mencionó anteriormente, hay dos tipos de transistores MOS: nMOS y pMOS, ambos con 4 terminales: fuente (source), drenador (drain), puerta (gate) y cuerpo (bulk). Estos transistores son controlados por un campo eléctrico que es formado gracias a su estructura MOS, formada de capas de conductores y aislantes. Éstos son dispositivos con portadores mayoritarios en los cuales la corriente que fluye entre la fuente y el drenador, es controlada por un voltaje aplicado a la puerta. En un transistor nMOS los portadores mayoritarios son los electrones y en el transistor pMOS los portadores mayoritarios son los huecos. Para una mejor comprensión del funcionamiento de los transistores tipos MOSFETs, es importante revisar primero la funcionalidad de una estructura MOS simple, con puerta y cuerpo, y sin fuente ni drenador. La capa superior de la estructura es un buen conductor y corresponde a la puerta. Anteriormente, era común el uso de metal para esta capa superior, pero tiempo después se optó por usar silicio policristalino. La capa intermedia, es una película delgada de dióxido de silicio (SiO_2), llamada puerta de óxido. La capa inferior es el cuerpo de silicio dopado.

En la Figura 3.1 se muestra una estructura MOS, con un cuerpo dopado de material tipo-

p, en el cual los portadores mayoritarios son huecos. El cuerpo está aterrizado y un voltaje es aplicado a la puerta. El óxido debajo de la puerta es un buen aislante por lo que la corriente que fluye entre la puerta y el cuerpo es prácticamente nula.

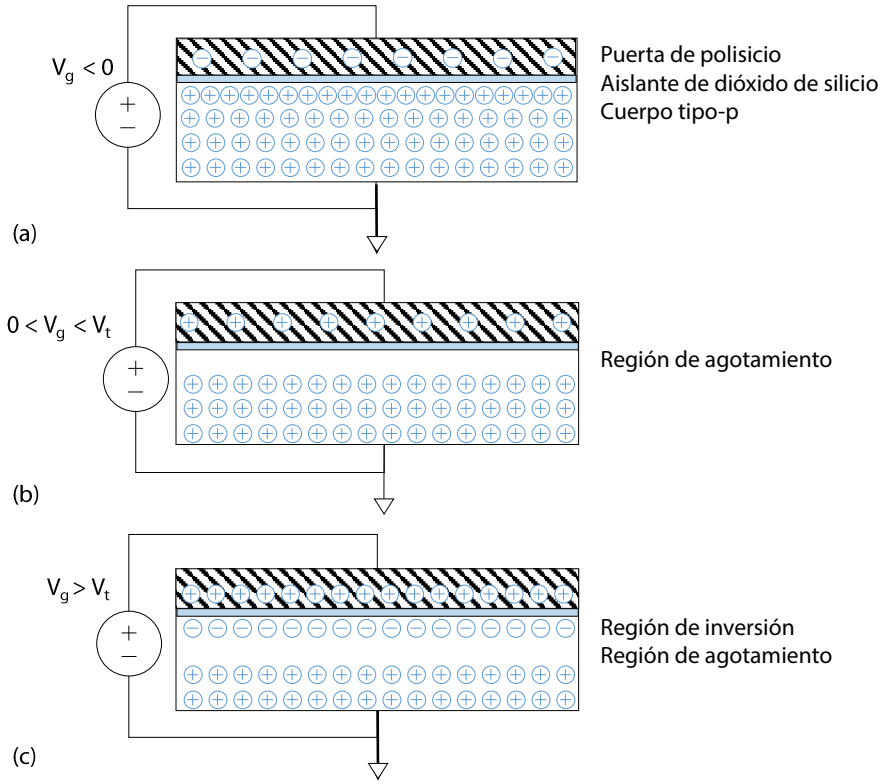


Figura 3.1: Sección transversal de una estructura MOS.

En la Figura 3.1(a), se aplica un voltaje negativo a la puerta, por lo que hay carga negativa en la misma y los huecos (cargados positivamente), son atraídos a la región inferior de la puerta de óxido. A esto se le conoce como modo de acumulación. En la Figura 3.1(b), se aplica un pequeño voltaje a la puerta y con esto algunas cargas positivas que se encuentran en la puerta, repelen huecos que se encuentren cerca de la zona inferior de la puerta de óxido, a esto se le conoce como modo de agotamiento. En la Figura 3.1(c), se aplica un voltaje más alto a la puerta, siendo este mayor que el voltaje de umbral V_t (el voltaje de umbral depende en el número de dopantes). Los huecos son repelidos más lejos y algunos electrones del cuerpo son atraídos a la zona que se encuentra justo debajo de la puerta de óxido, formando una delgada capa conductora de electrones en el cuerpo tipo-p, llamada *capa de inversión*. En el caso donde se cuente con un cuerpo tipo-n, para el modo de acumulación se aplica un voltaje positivo en la puerta, introduciendo huecos y causando atracción de electrones por debajo de la puerta de óxido. El modo de agotamiento ocurre cuando se aplica un pequeño voltaje negativo y esto ocasiona que algunas cargas negativas se encuentren en la puerta (cerca de la interfaz) y los electrones del cuerpo son repelidos de la región cercana de la interfaz. Por

último, la capa de inversión se genera cuando se aplica un voltaje más negativo, llenando de electrones la puerta y atrayendo huecos cerca de la interfaz entre el cuerpo y la puerta de óxido.

La estructura de un transistor nMOS consiste en una estructura tipo MOS entre dos regiones tipo-n llamadas fuente y drenador. En la Figura 3.2(a) el voltaje puerta-fuente (V_{gs}) es menor al voltaje de umbral ($V_{gs} < V_t$), la fuente y el drenador tienen electrones libres (debido a su dopaje) y el cuerpo tiene huecos libres, pero no electrones libres. Suponiendo que la fuente está aterrizada, las uniones entre el cuerpo y la fuente o el drenador están inversamente polarizadas, por esto, prácticamente la corriente que fluye entre esas regiones es cero. En estas condiciones se dice que el transistor está "apagado" y que está en modo de corte.

En la Figura 3.2(b), el voltaje en la puerta es más grande que el voltaje de umbral ($V_{gs} > V_t$). Con esto, se genera una región de inversión con electrones (ahora, portadores mayoritarios), llamada canal, el cual conecta la fuente con el drenador, creando un camino conductor y "prendiendo" el transistor. El número de portadores y la conductividad, incrementan con el voltaje de la puerta. La diferencia de potencial entre el drenador y la fuente es $V_{ds} = V_{gs} - V_{gd}$. Si $V_{ds} = 0$, no hay campo eléctrico para generar corriente del drenador a la fuente. En la Figura 3.2(c), se aplica un pequeño voltaje V_{ds} al drenador, de tal manera que se generará un flujo de corriente I_{ds} a través del canal que conecta el drenador y la fuente. Este modo de operación se denomina de varias formas: lineal, resistivo, triodo, no saturado o insaturado.

Cuando el voltaje entre el drenador y la fuente se incrementa lo suficiente ($V_{ds} > V_{gs} - V_t$), de tal manera que $V_{gd} < V_t$, cerca del drenador el canal deja de ser una zona invertida y tiene una forma de estrangulamiento, la cual se conoce como *pinched off* (Figura 3.2(d)). Sin embargo, la conducción todavía se produce por la corriente de electrones bajo la influencia del voltaje del drenador positivo. Cuando los electrones llegan al final del canal, se inyectan en la región de agotamiento cerca del drenador y se aceleran hacia el drenador. Por encima de este voltaje del drenador, la corriente I_{ds} es controlada solo por el voltaje de la puerta y deja de ser influenciada por el drenador. Este modo de operación se le conoce como "modo de saturación".

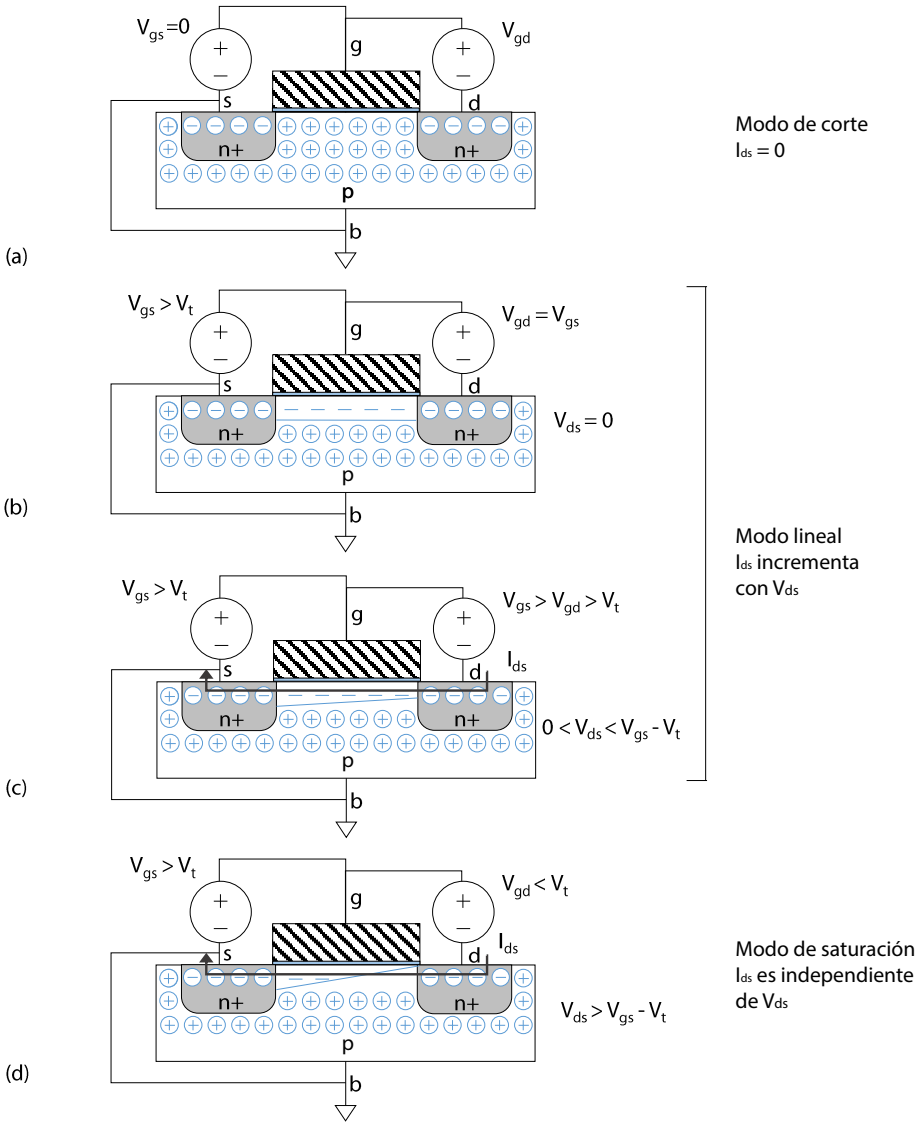


Figura 3.2: Regiones de operación de un transistor nMOS.

Para el caso de un transistor pMOS, ocurre lo contrario. El cuerpo está dopado con material tipo-n, las regiones que corresponden a la fuente y drenador, están dopadas con material tipo-p, mientras que la estructura de la puerta y puerta de óxido permanecen igual que las de la estructura de un transistor nMOS. El cuerpo, está conectado al voltaje alto de operación, por lo que las regiones del drenador y la fuente están polarizadas inversamente. Cuando el voltaje aplicado en la puerta es alto, no se forma el canal de conducción y por ende no es posible generar una corriente entre la fuente y el drenador. Cuando este voltaje decrece cerca del voltaje de umbral, los huecos son atraídos para formar un canal tipo-p cerca de la interfaz de la puerta de óxido y el cuerpo, permitiendo el flujo de corriente entre el drenador y la fuente.

En resumen, hay 3 modos de operación. Para el caso de un nMOS, si $V_{gs} < V_t$, el transistor está en modo de corte, es decir, el transistor está apagado. Si $V_{gs} > V_t$ el transistor está encendido. Si V_{ds} es pequeño, el transistor actúa como una resistencia lineal en la cual, el flujo de corriente es regulado por V_{ds}. Si $V_{gs} > V_t$ y V_{ds} es más grande, el transistor actúa como una fuente de corriente independiente de V_{ds}.

3.1.1. Modelo Shockley

El modelo Shockley es utilizado en el análisis de los circuitos con dispositivos MOSFETs, en el cual se considera que la longitud del canal es suficientemente larga para que el campo eléctrico entre el drenador y la fuente sea relativamente pequeño y que para un transistor nMOS, la corriente a través del canal sea cero cuando el transistor está apagado. Cuando está encendido ($V_{gs} > V_t$), en la interfaz de la puerta de óxido con el cuerpo, se atraen portadores para formar el canal. La corriente de electrones de la fuente al drenador es proporcional al campo eléctrico entre ambas regiones. Por lo tanto, la carga en el canal es la siguiente:

$$Q_{channel} = C_g(V_{gc} - V_t), \tag{3.1}$$

donde C_g es la capacitancia que hay entre la puerta y el canal (Figura 3.3), y $(V_{gc} - V_t)$ es la cantidad de voltaje que atrae carga al canal más allá del mínimo requerido para invertir de p a n. La referencia del voltaje de la puerta, es el canal el cual no está aterrizado. Si la fuente tiene un voltaje V_s y el drenador V_d, el promedio es $V_c = (V_s + V_d)/2$. Por lo tanto, la diferencia media de potenciales V_{gc} entre la puerta y el canal es $V_g - V_c = V_{gc} - V_{ds}/2$ [2].

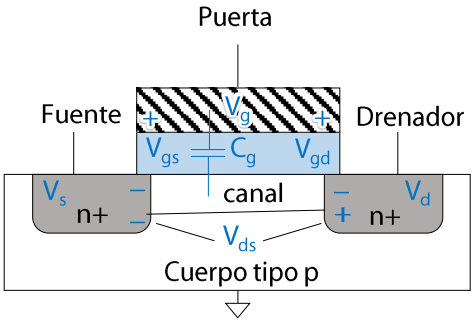


Figura 3.3: Promedio de voltaje entre la puerta y el canal.

Es posible modelar la puerta y el cuerpo como un capacitor de placas paralelas, cuya capacitancia es proporcional al área sobre el espesor de la puerta de óxido. Si la puerta tiene un largo L y ancho W, y el óxido tiene un espesor t_{ox}, como se muestra en la Figura 3.4, la capacitancia es

$$C_g = k_{ox}\varepsilon_0 \frac{WL}{t_{ox}} = \varepsilon_{ox}\frac{WL}{t_{ox}} = C_{ox}WL \tag{3.2}$$

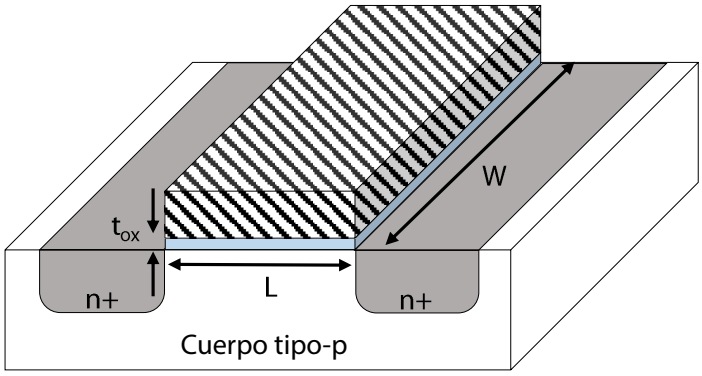


Figura 3.4: Dimensiones de un transistor.

donde ε_0 es la permitividad en el espacio libre y la permitividad del SiO_2 es $k_{ox} = 3.9$ veces más grande. En ocasiones, la relación ε_{ox}/t_{ox} es llamada C_{ox}, la capacitancia por unidad de área de la puerta de óxido.

Cada portador en el canal es acelerado a una velocidad promedio v, proporcional al campo eléctrico entre el drenador y la fuente, como $v = \mu E$. Donde μ es la constate de proporcionalidad llamada movilidad. El campo eléctrico E es la diferencia de voltaje entre drenador con la fuente, dividida por el largo, como se muestra en la siguiente ecuación:

$$E = \frac{V_{ds}}{L} \tag{3.3}$$

El tiempo requerido para que los portadores crucen el canal es la razón entre el largo del canal y la velocidad de los portadores L/v. Por lo tanto, la corriente entre la fuente y el drenador es la carga total divida por el tiempo requerido para que los portadores crucen, como se muestra enseguida:

$$\begin{aligned} I_{ds} &= \frac{Q_{channel}}{L/v} \\ &= \mu C_{ox}\frac{W}{L}(V_{gs} - V_t - V_{ds}/2)V_{ds} \\ &= \beta(V_{GT} - V_{ds}/2)V_{ds} \end{aligned} \tag{3.4}$$

donde $\beta = \mu C_{ox}\frac{W}{L}$ y $V_{GT} = V_{gs} - V_t$. Los parámetros que dependen de la geometría del transistor y de la tecnología, usualmente son unidos en la constante β. La ecuación 3.4 describe la región lineal de operación, para $V_{gs} > V_t$, pero con V_{ds} relativamente pequeño.

Este modo es llamado lineal o resistivo debido a que cuando $V_{ds} << V_{GT}$, I_{ds} incrementa casi linealmente con V_{ds}, como una resistencia lineal.

Si $V_{ds} > V_{dsat} \equiv V_{GT}$, el canal dejará de estar invertido cerca del drenador, con una forma que asemeja a un pellizco. A partir de este punto, aumentar el voltaje del drenador no tiene más efecto sobre la corriente. Sustituyendo $V_{ds} = V_{dsat}$ en la ecuación 3.4 para este punto de máxima corriente, se encuentra una expresión para la corriente de saturación que es independiente de V_{ds}:

$$I_{ds} = \frac{\beta}{2} V_{GT}^2 \tag{3.5}$$

Está expresión es valida para $V_{gs} > V_t$ y $V_{ds} > V_{dsat}$. En la ecuación 3.6 se muestra un resumen del comportamiento de la corriente en las 3 regiones.

$$I_{ds} = \begin{cases} 0 & V_{gs} < V_t & \text{Cutoff} \\ \beta(V_{GT} - V_{ds}/2)V_{ds} & V_{ds} < V_{dsat} & \text{Linear} \\ \frac{\beta}{2} V_{GT}^2 & V_{dsat} & \text{Saturation} \end{cases} \tag{3.6}$$

En la Figura 3.5 se muestran las características I-V para un transistor. De acuerdo al modelo Shockley, la corriente es cero cuando los voltajes en la puerta son menores que V_t. Para voltajes mayores, la corriente incrementa linealmente con V_{ds} (para pequeños voltajes V_{ds}). A medida que V_{ds} alcanza el punto de saturación $V_{dsat} = V_{GT}$, el flujo de corriente desciende y finalmente se vuelve independiente de V_{ds} cuando el transistor está saturado. Los transistores pMOS se comportan de la misma manera, pero con los signos de todos los voltajes y corrientes invertidos. La característica I-V están en el tercer cuadrante, como se muestra en la Figura 3.5(b). En un transistor pMOS, la corriente fluye desde la fuente para drenarse. La movilidad de los agujeros en el silicio es típicamente menor que la de los electrones, esto significa que los transistores pMOS proporcionan menos corriente que los transistores nMOS en un tamaño similar y por lo tanto, son más lentos.

3.1.2. Modelo simple para la capacitancia en la estructura MOS

Debido a la estructura tipo sándwich de la estructura MOS, la puerta y el cuerpo forman un capacitor de placas paralelas teniendo a la puerta de óxido como un dieléctrico. Dicho capacitor genera el campo eléctrico que atrae a los portadores cerca de la interfaz de la puerta de óxido y el cuerpo, por lo tanto, se requiere una alta capacitancia para obtener una corriente I_{ds} alta. Dicha capacitancia formada en la estructura MOS, está dada por

$$C_g = C_{ox} W L \tag{3.7}$$

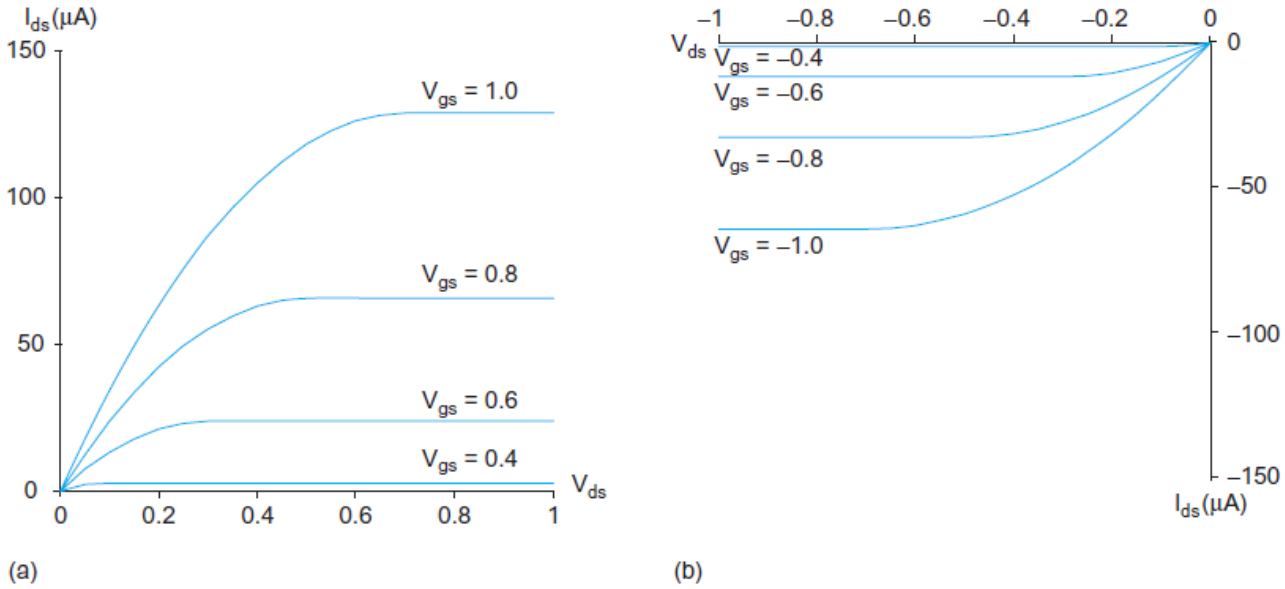


Figura 3.5: Curva característica I-V de un transistor (a) nMOS y (b) pMOS (tomada de [2]).

El plato inferior del capacitor es el canal, el cual no es una de las terminales del transistor. Cuando el transistor está prendido, el canal se extiende de la fuente al drenador, y cuando está saturado, el canal se hace más pequeño cerca del drenador, por esto, se aproxima la capacitancia de la puerta respecto a la fuente y se conoce como C_{gs}. La mayoría de los transistores usados en circuitos digitales, usan el largo mínimo del proceso de manufactura para optimizar las implementaciones respecto a consumo de energía, área o velocidad. Tomando L como una constante es posible definir

$$C_g = C_{permicron} \times W \tag{3.8}$$

donde

$$C_{permicron} = C_{ox} L = \frac{\varepsilon_{ox}}{t_{ox}} L \tag{3.9}$$

Para el proceso de 600 nm, $C_{permicron}$ es aproximadamente de $2\,\mathrm{fF}/\mu\mathrm{m}$. $C_{permicron}$ se emplea para simplificar el uso de las constantes que se muestra en la ecuación 3.9.

Además de la capacitancia que genera la puerta, el drenador y la fuente también generan una. Estas capacitancias no son fundamentales para la operación de los dispositivos, pero repercuten en el desempeño del mismo, añadiendo capacitancias parasíticas. Estas capacitancias de la fuente y el drenador, son generadas por la unión p-n que conforman, las cuales se conocen como capacitancias de difusión C_{sb} y C_{db}. La región de agotamiento sin portadores

libres a lo largo de la región de difusión, actúa como un aislante entre la la unión p-n. La capacitancia de estas uniones depende del área y el perímetro de la difusión de la fuente y drenador, la profundidad de la difusión, los niveles de dopaje y el voltaje. Como la difusión tiene alta capacitancia y alta resistencia, generalmente se hace lo más pequeña posible en el diseño. En la Figura 3.6 se muestran tres ejemplos de regiones de difusión en dos transistores conectados en serie.

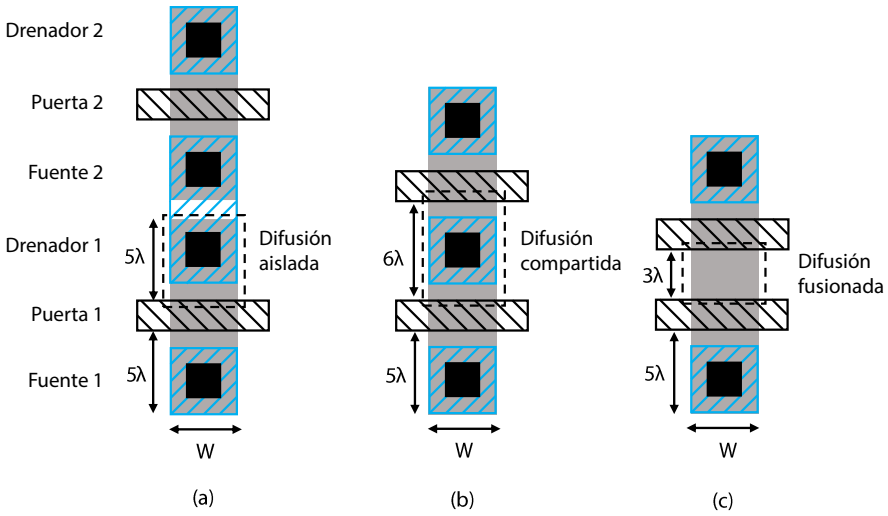


Figura 3.6: Regiones de difusión y contactos entre la fuente y el drenador para el caso en el que están (a) aisladas, (b) compartidas y (c) cuando no hay contacto y las regiones de difusión son fusionadas.

En la Figura 3.6(a) cada fuente y drenador tienen su propia región de difusión con un contacto independiente. En la Figura 3.6(b) el drenador 1 y la fuente 2 tienen las regiones de difusión fusionadas, con un solo contacto. En la Figura 3.6(c), la fuente y el drenador están fusionadas sin ningún contacto. El promedio de la capacitancia de los 3 diseños anteriores son distintas, la capacitancia de difusión de la fuente o drenador no contactado es algo menor porque el área es más pequeña y compartir región de difusión reduce alrededor de la mitad de la capacitancia promedio final. En el proceso de 600 nm, la capacitancia aislada es $C_d = 1.14\,\mathrm{fF}/\mu\mathrm{m}$, la capacitancia compartida es $C_d = 1.41\,\mathrm{fF}/\mu\mathrm{m}$ y la capacitancia fusionada es $C_d = 1.19\,\mathrm{fF}/\mu\mathrm{m}$.

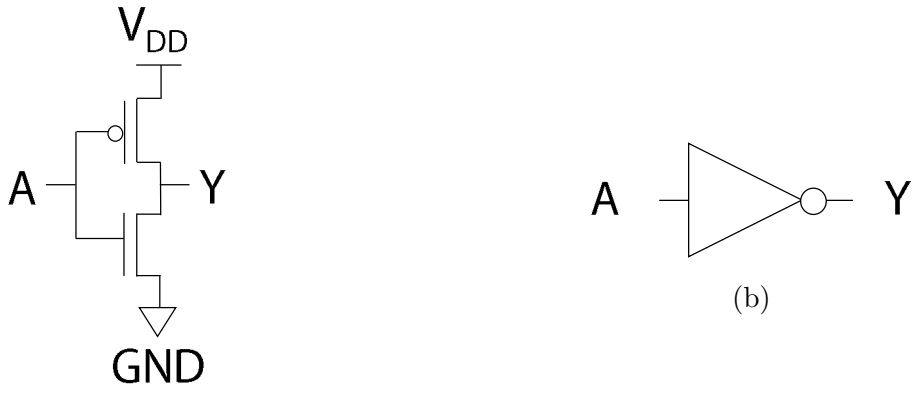
En la Tabla 3.1 se muestran las características de dispositivos de distinto tamaño y fabricante.

Vendedor		Orbit	HP	AMI	AMI	TSMC	TSMC	TSMC	IBM	IBM	IBM
Fabricante		MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	TSMC	IBM	IBM	IBM
Feature Size	nm	2000	800	600	600	350	250	180	130	90	65
V_{DD}	V	5	5	5	3.3	3.3	2.5	1.8	1.2	1.0	1.0
Gates											
C_g (*delay*)	$fF/\mu m$	1.77	1.67	1.55	1.48	1.9	2.3	1.67	1.04	0.97	0.8
C_g (potencia)	$fF/\mu m$	2.24	1.7	1.83	1.76	2.2	2.92	2.06	1.34	1.23	1.07
FO4 Inv. *Delay*	ps	856	297	230	312	210	153	75.6	45.9	37.2	17.2
nMOS											
C_d (aislado)	$fF/\mu m$	1.19	1.11	1.14	1.21	1.63	1.88	1.12	0.94	0.89	0.76
C_d (compartido)	$fF/\mu m$	1.62	1.43	1.41	1.5	2.04	2.6	1.62	1.56	1.6	1.28
C_d (fusionado)	$fF/\mu m$	1.48	1.36	1.19	1.24	1.6	2.16	1.41	1.4	1.51	1.2
R_n (individual)	$k\Omega \cdot \mu m$	30.3	10.1	9.19	11.9	5.73	4.02	2.69	2.54	2.35	1.34
R_n (serie)	$k\Omega \cdot \mu m$	22.1	6.95	6.28	8.59	4.01	3.1	2	1.93	1.81	1.13
V_{tn} (I const.)	V	0.65	0.65	0.7	0.7	0.59	0.48	0.41	0.32	0.32	0.31
V_{tn} (Lineal)	V	0.65	0.75	0.76	0.76	0.67	0.57	0.53	0.43	0.43	0.43
I_{dsat}	$\mu A/\mu m$	152	380	387	216	450	551	566	478	497	755
I_{off}	$pA/\mu m$	2.26	9.36	2.21	1.45	6.57	56.3	93.9	1720	4000	33400
I_{gate}	$pA/\mu m$	n/a	n/a	n/a	n/a	n/a	n/a	n/a	1.22	3620	8520
pMOS											
C_d (aislado)	$fF/\mu m$	1.42	1.17	1.31	1.42	1.89	2.07	1.24	0.94	0.74	0.73
C_d (compartido)	$fF/\mu m$	1.92	1.62	1.73	1.86	2.37	2.89	1.79	1.56	1.25	1.25
C_d (fusionado)	$fF/\mu m$	1.52	1.23	1.35	1.43	1.83	2.4	1.56	1.41	1.16	1.18
R_n (individual)	$k\Omega \cdot \mu m$	67.1	26.7	19.9	29.6	16.1	8.93	6.51	6.39	5.47	2.87
R_n (serie)	$k\Omega \cdot \mu m$	53.9	21.4	15.4	23.6	13.3	6.91	5.91	5.48	4.92	2.42
V_{tn} (I const.)	V	0.72	0.91	0.9	0.9	0.83	0.46	0.43	0.33	0.35	0.33
V_{tn} (Lineal)	V	0.71	0.94	0.93	0.93	0.88	0.52	0.51	0.42	0.43	0.42
I_{dsat}	$\mu A/\mu m$	70.5	154	215.3	99	181	245	228	177	187	360
I_{off}	$pA/\mu m$	2.18	1.57	2.08	1.38	2.06	30.1	25.2	1330	2780	19500
I_{gate}	$pA/\mu m$	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0.06	1210	2770

Tabla 3.1: Características de diversos dispositivos en distintos procesos (tomada de [2]).

3.2. Lógica CMOS

La tecnología CMOS se refiere a un proceso de fabricación que utiliza pares complementarios y simétricos de transistores nMOS y pMOS para realizar funciones lógicas. El ejemplo más básico de una implementación en CMOS, es un inversor o compuerta NOT, la cual emplea un transistor pMOS y un transistor nMOS conectados en serie. La fuente del transistor pMOS se conecta a V_{DD}, y GND a la fuente del nMOS. El nodo entre los dos drenadores conectados, forman la salida de la compuerta mientras que la entrada corresponde a las puertas de ambos transistores (ver Figura 3.7). Cuando a la entrada del inversor se coloca un voltaje alto, el transistor pMOS es apagado y el transistor nMOS es encendido, dando lugar en la salida a valor bajo de voltaje (GND), y se dice que la salida es "jalada" a 0 (*pull-down*). Cuando en la entrada se coloca un voltaje bajo, el transistor nMOS es apagado y el transistor pMOS es prendido provocando en la salida un voltaje alto (V_{DD}) y se dice que es jalada a 1 (*pull-up*).



(a) (b)

Figura 3.7: (a)Diagrama esquemático de un inversor en tecnología CMOS. (b) Símbolo del inversor.

Este tipo de celdas también conocidas como compuertas CMOS lógicas estáticas, se conforman de dos redes. Una red *pull-up*, compuesta únicamente por transistores pMOS que conecta la salida a 1, y una red *pull-down* compuesta únicamente por transistores nMOS que conecta la salida a 0 (ver Figura 3.8). El diseño de estas compuertas estáticas, permite que cuando una red está apagada, la otra pueda estar prendida para generar un nivel lógico a la salida. Tomando como ejemplo el inversor, la red *pull-up* está compuesta por el transistor pMOS y la red *pull-down* está compuesta por el transistor nMOS.

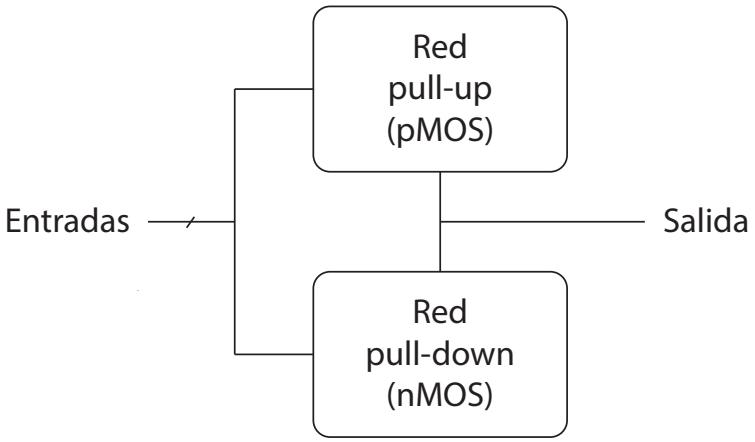


Figura 3.8: Esquema general de una compuerta usando una red *pull-up* y una red *pull-down*.

Para hacer compuertas con mayor cantidad de transistores, como una compuerta NAND o NOR, de dos entradas, se debe tomar en cuenta que la salida de dos transistores en serie está encendida si ambos transistores se encuentran encendidos y que la salida de dos o más transistores en paralelo está encendida si por lo menos un transistor está encendido. En la Figura 3.9 se muestra la implementación CMOS de la compuerta NAND.

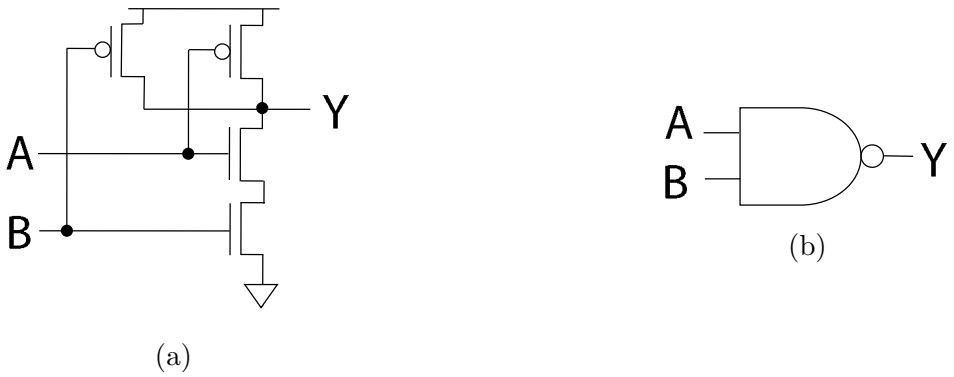


Figura 3.9: (a) Diagrama esquemático de compuerta NAND de 2 entradas y su (b) símbolo.

Esta compuerta consiste en dos transistores pMOS en paralelo que conforman la red *pull-up* y dos transistores nMOS en serie que conforman la red *pull-down*. Si las dos entradas, A o B, son 1, ambos transistores pMOS estarán apagados y los transistores nMOS en serie, estarán prendidos y la salida Y sera igual a 0. Si cualquier entrada es 0, la red *pull-down* estará apagada y por lo menos uno de la transistores pMOS en paralelo, estará prendido, dando lugar a un 1 en la salida.

En la Figura 3.10 se muestra la implementación CMOS de una compuerta NOR de dos entradas. La cual está compuesta por dos transistores nMOS y dos transistores pMOS. La red

pull-up se conforma por los dos transistores pMOS en serie y la red *pull-down* se conforma por los transistores nMOS en paralelo. Cuando cualquier entrada se encuentra en 1, la red *pull-up* se apaga y la red *pull-down* es encienda, dando lugar a un 0 en la salida. Cuando ambas entradas son 0, la red *pull-up* se enciende y la otra se apaga, generando un 1 a la salida.

(a) (b)

Figura 3.10: (a) Diagrama esquemático de compuerta NOR de 2 entradas y su (b) símbolo.

Los posibles estados que puede tomar la salida de una compuerta CMOS, pueden ser 4, dependiendo del estado de las redes. Como se mencionó anteriormente, en las compuertas NAND y NOR, la salida puede tomar 1 o 0 lógicos, prendiendo una red y apagando la otra. Cuando ambas redes están apagadas, se genera una condición de alta impedancia (o flotante), denotada con Z. Cuando ambas redes están encendidas se genera una condición de contención (o *crowbarred*), denotada con X. La salida de está condición está indeterminada.

3.3. Flujo de Diseño VLSI

El flujo de diseño en VLSI es la secuencia de procesos o pasos involucrados en la creación de un circuito integrado. La mayoría de los flujos de diseño siguen el flujo que se muestra en la Figura 3.11, en el cual, la construcción de bloques para diseños digitales como compuertas combinacionales lógicas (NOT, AND, OR, NAND, AND-OR-INVERT, OR-AND-INVERT y otras compuertas compuestas), celdas secuenciales (flip-flops y latches) y otras celdas especiales son diseñadas especialmente (como los diseños analógicos), verificadas, caracterizadas y usadas en diseños de circuitos más grandes. Estos diseños pre-definidos y pre-verificados conforman una librería de celdas estándar.

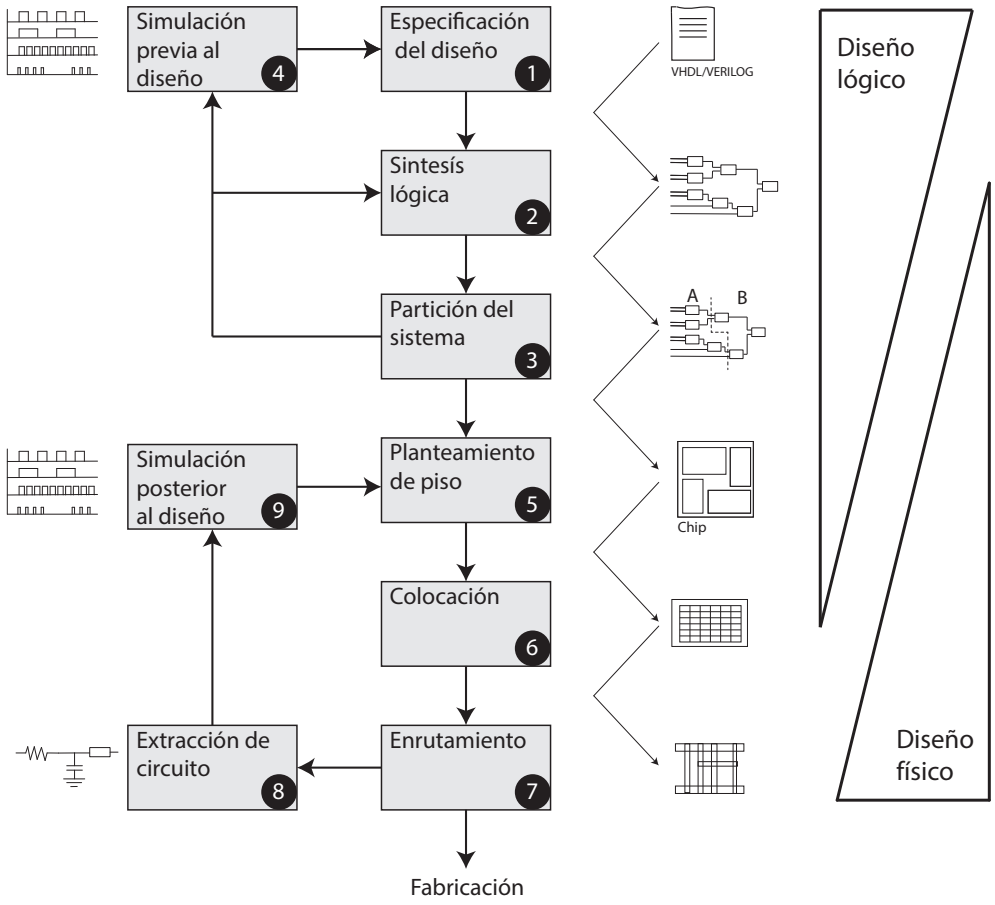


Figura 3.11: Flujo de diseño de ASICs.

A continuación se describen los pasos necesarios para llevar a cabo el proceso de diseño típico de un circuito integrado, llevado de la mano con la implementación de un sumador completo de 8 bits como el que se muestra en la Figura 3.12.

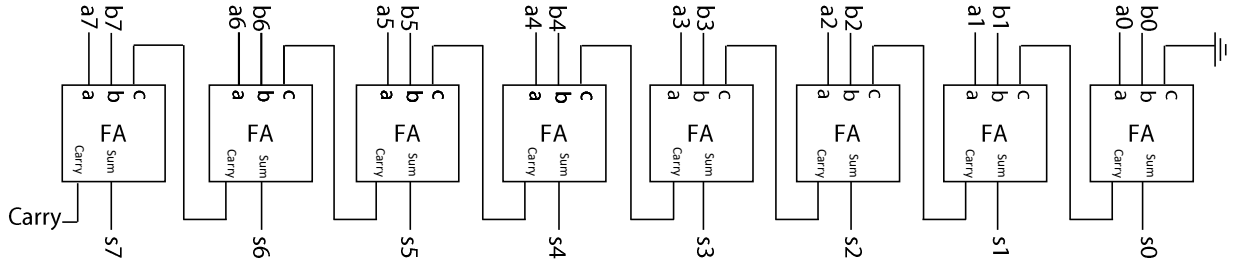


Figura 3.12: Esquema de un sumador completo de 8 bits.

1. **Especificación del sistema**: El primer paso para el proceso de diseño es establecer

las especificaciones del sistema. Esta especificación puede ser una representación de alto nivel del sistema, considerando factores como el rendimiento, funcionalidad y dimensiones físicas (o tecnología de fabricación). El resultado son especificaciones para el tamaño, velocidad, potencia y funcionalidad del sistema. La especificación generalmente habla de los requisitos funcionales detallados del chip, la velocidad mínima de operación, la potencia máxima y el área para el circuito y otros requisitos de robustez y confiabilidad.

Para ejemplificar lo anterior, definiremos un bloque que sea capaz de sumar 2 números de 8 bits y con la posibilidad de agregar un bit de *carry* a la suma (sumador completo de 8 bits). A la salida debemos obtener un número de 8 bits resultante de la suma de las entradas y 1 bit de *carry* de salida. La elaboración de un sumador completo de 8 bits, se puede implementar a través de 8 instancias de sumadores completos de 2 bits, como se muestra en el siguiente código de Verilog:

```
module full_adder( A, B, Ci, S, Co );

input wire A, B, Ci;
output wire S, Co;
//Sum
assign S = A ^ B ^ Ci;
//Carry
assign Co = (A&B) | (B&Ci) | (A&Ci);

endmodule
```

Realizando instancias de este módulo 8 veces, podemos formar un sumador completo de 8 bits, como se muestra en el siguiente código Verilog:

```
module FA8 (A,B,Ci,S,Co);

input wire [7:0] A, B;
input wire Ci;
output wire [7:0] S;
output wire Co;

wire [6:0] Cix;

full_adder I1 (.A(A[0]),.B(B[0]), .Ci(Ci), .S(S[0]), .Co(Cix[0]));
full_adder I2 (.A(A[1]),.B(B[1]), .Ci(Cix[0]), .S(S[1]), .Co(Cix[1]));
full_adder I3 (.A(A[2]),.B(B[2]), .Ci(Cix[1]), .S(S[2]), .Co(Cix[2]));
full_adder I4 (.A(A[3]),.B(B[3]), .Ci(Cix[2]), .S(S[3]), .Co(Cix[3]));
full_adder I5 (.A(A[4]),.B(B[4]), .Ci(Cix[3]), .S(S[4]), .Co(Cix[4]));
full_adder I6 (.A(A[5]),.B(B[5]), .Ci(Cix[4]), .S(S[5]), .Co(Cix[5]));
full_adder I7 (.A(A[6]),.B(B[6]), .Ci(Cix[5]), .S(S[6]), .Co(Cix[6]));
full_adder I8 (.A(A[7]),.B(B[7]), .Ci(Cix[6]), .S(S[7]), .Co(Co));

endmodule
```

2. **Síntesis lógica**: En este paso, el ancho de palabras, la asignación de registros, las operaciones aritméticas y las operaciones lógicas que representan el diseño funcional,

se crean en base a las especificaciones del sistema, en una representación de compuertas lógicas, generalmente a nivel de transferencia de registro (RTL) y se expresa en un lenguaje de descripción de hardware (HDL, *Hardware Description Language*) como Verilog o VHDL. Esta descripción puede ser usada en simulación y verificación, y se conforma de expresiones booleanas e información de tiempo. Las expresiones booleanas se minimizan para lograr el diseño lógico más pequeño que se ajuste al diseño funcional. En algunos casos especiales, el diseño lógico puede automatizarse utilizando herramientas de síntesis de alto nivel. Estas herramientas producen una descripción RTL a partir de una descripción del comportamiento del diseño. El código de diseño RTL una vez verificado, se entrega a una herramienta CAD de sintetizador lógico que convierte el código en una lista de puertos, celdas estándar, e interconexiones entre ellos, generando un nueva descripción en verilog pero con la información de la librería estándar.

Continuando con el ejemplo de un sumador, el proceso de sintetizado puede llevarse a cabo por la herramienta *Design Compiler* de la empresa *Synopsys*, la cual toma el diseño Verilog en un alto nivel mostrado anteriormente y usando una librería precargada de celdas estándar y diferentes parámetros como cargas capacitivas a la salida o tiempos de procesado definidos en las especificaciones del sistema, sintetiza un nuevo netlist compuesto por compuertas de la librería estándar, dicho circuito está optimizado con información precisa de temporización de cada celda. En este trabajo se empleo la librearía estándar C5 de la empresa AMIS. Dicha librería está diseñada en un proceso de 0.5 μm CMOS y provee una sólida base de celdas comunes para la elaboración de circuitos digitales. En seguida se muestra el netlist sintetizado del módulo del sumador de 8 bits y el del sub-módulo de un sumador de 2 bits.

```
1  module FA8 ( A, B, Ci, S, Co );
2    input [7:0] A;
3    input [7:0] B;
4    output [7:0] S;
5    input Ci;
6    output Co;
7    wire   n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14,
      n15, n16,
8          n17, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27, n28,
      n29, n30,
9          n31, n32, n33, n34;
10   wire   [6:0] Cix;
11
12   full_adder I1 (.A(n20), .B(n4), .Ci(n2), .S(S[0]), .Co(Cix[0]) );
13   full_adder I2 (.A(n22), .B(n6), .Ci(Cix[0]), .S(S[1]), .Co(Cix[1]) )
      ;
14   full_adder I3 (.A(n24), .B(n8), .Ci(Cix[1]), .S(S[2]), .Co(Cix[2]) )
      ;
15   full_adder I4 (.A(n26), .B(n10), .Ci(Cix[2]), .S(S[3]), .Co(Cix[3])
      );
16   full_adder I5 (.A(n28), .B(n12), .Ci(Cix[3]), .S(S[4]), .Co(Cix[4])
      );
17   full_adder I6 (.A(n30), .B(n14), .Ci(Cix[4]), .S(S[5]), .Co(Cix[5])
```

```
    );
  full_adder I7 (.A(n32), .B(n16), .Ci(Cix[5]), .S(S[6]), .Co(Cix[6])
    );
  full_adder I8 (.A(n34), .B(n18), .Ci(Cix[6]), .S(S[7]), .Co(Co) );
  inv1_b U1 ( .A(Ci), .Q(n1) );
  inv1_b U2 ( .A(n1), .Q(n2) );
  inv1_b U3 ( .A(B[0]), .Q(n3) );
  inv1_b U4 ( .A(n3), .Q(n4) );
  inv1_b U5 ( .A(B[1]), .Q(n5) );
  inv1_b U6 ( .A(n5), .Q(n6) );
  inv1_b U7 ( .A(B[2]), .Q(n7) );
  inv1_b U8 ( .A(n7), .Q(n8) );
  inv1_b U9 ( .A(B[3]), .Q(n9) );
  inv1_b U10 ( .A(n9), .Q(n10) );
  inv1_b U11 ( .A(B[4]), .Q(n11) );
  inv1_b U12 ( .A(n11), .Q(n12) );
  inv1_b U13 ( .A(B[5]), .Q(n13) );
  inv1_b U14 ( .A(n13), .Q(n14) );
  inv1_b U15 ( .A(B[6]), .Q(n15) );
  inv1_b U16 ( .A(n15), .Q(n16) );
  inv1_b U17 ( .A(B[7]), .Q(n17) );
  inv1_b U18 ( .A(n17), .Q(n18) );
  inv1_b U19 ( .A(A[0]), .Q(n19) );
  inv1_b U20 ( .A(n19), .Q(n20) );
  inv1_b U21 ( .A(A[1]), .Q(n21) );
  inv1_b U22 ( .A(n21), .Q(n22) );
  inv1_b U23 ( .A(A[2]), .Q(n23) );
  inv1_b U24 ( .A(n23), .Q(n24) );
  inv1_b U25 ( .A(A[3]), .Q(n25) );
  inv1_b U26 ( .A(n25), .Q(n26) );
  inv1_b U27 ( .A(A[4]), .Q(n27) );
  inv1_b U28 ( .A(n27), .Q(n28) );
  inv1_b U29 ( .A(A[5]), .Q(n29) );
  inv1_b U30 ( .A(n29), .Q(n30) );
  inv1_b U31 ( .A(A[6]), .Q(n31) );
  inv1_b U32 ( .A(n31), .Q(n32) );
  inv1_b U33 ( .A(A[7]), .Q(n33) );
  inv1_b U34 ( .A(n33), .Q(n34) );
endmodule
```

```
module full_adder ( A, B, Ci, S, Co );
  input A, B, Ci;
  output S, Co;
  wire   n1, n3;

  inv1_b U1 ( .A(n1), .Q(Co) );
  eo21_b U2 ( .A(Ci), .B(n3), .Q(S) );
  an11_b U3 ( .A(B), .B(A), .C(n3), .D(Ci), .Q(n1) );
  eo21_b U4 ( .A(B), .B(A), .Q(n3) );
endmodule
```

3. **Partición del sistema**: Este proceso se lleva a cabo dividiendo el sistema en módu-

los, de tal manera que estos se unen con la menor cantidad de interconexiones. Cada módulo puede ser diseñado individualmente para acelerar el proceso de diseño del chip. Estos módulos pueden descomponerse en un nivel jerárquico hasta alcanzar un tamaño de módulos manejable. El objetivo es principalmente para separar diferentes bloques funcionales y para realizar la colocación y el enrutamiento de una manera más sencilla.

Para el caso del sumador, no es necesario realizar una división en más módulos debido a que la estructura del mismo está constituida por módulos como se muestra en la Figura 3.12. El archivo sintetizado puede ser editado y visualizado en el editor *Virtuoso*, de la suite de *Cadence*. En la Figura 3.14 se muestra el esquema del sumador de 8 bits visto desde *Virtuoso* compuesto por sumadores completos de 2 bits (ver Figura 3.13).

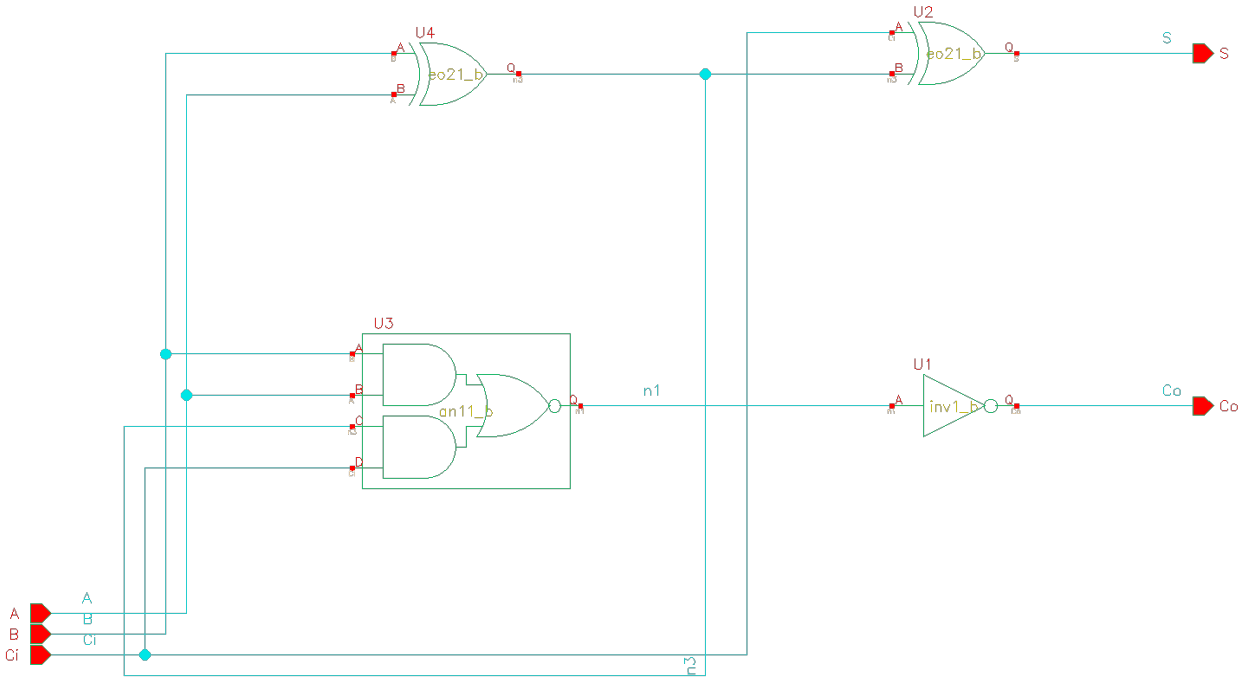


Figura 3.13: Diagrama esquemático para la implementación de un sumador completo de 2 bits generado por *Virtuoso*.

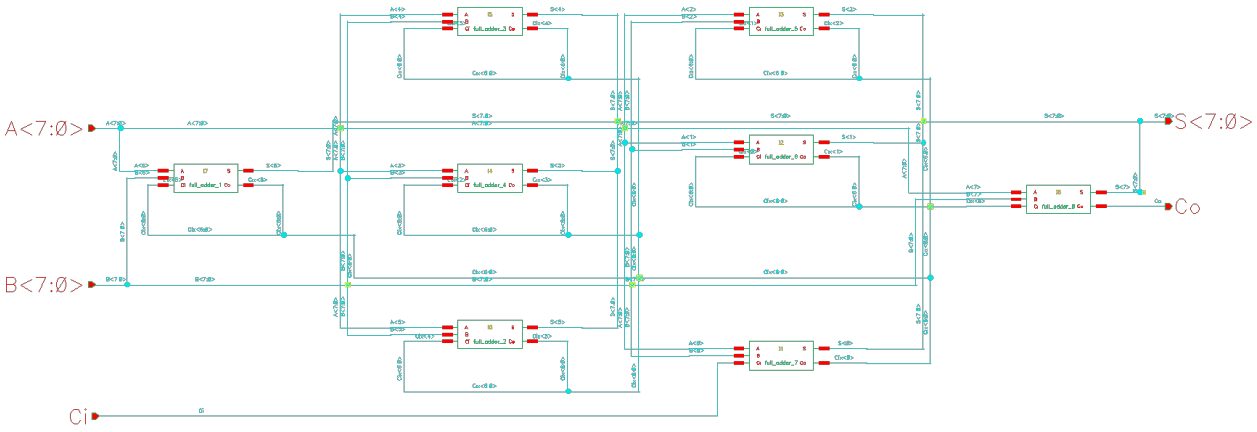


Figura 3.14: Diagrama esquemático para la implementación de un sumador completo de 8 bits generado por *Virtuoso*.

4. **Simulación previa al diseño**: En esta etapa se realiza una simulación del *netlist* sintetizado para corroborar que su funcionamiento sea el mismo que el de la descripción misma pero en alto nivel, es decir, que cumpla con las especificaciones establecidas.

   El netlist a nivel compuerta del sumador fue simulado usando el software *ModelSim* de la empresa *Mentor Graphics*. En la figura 3.15 se muestra la interfaz de *ModelSim* con la simulación del sumador de 8 bits.

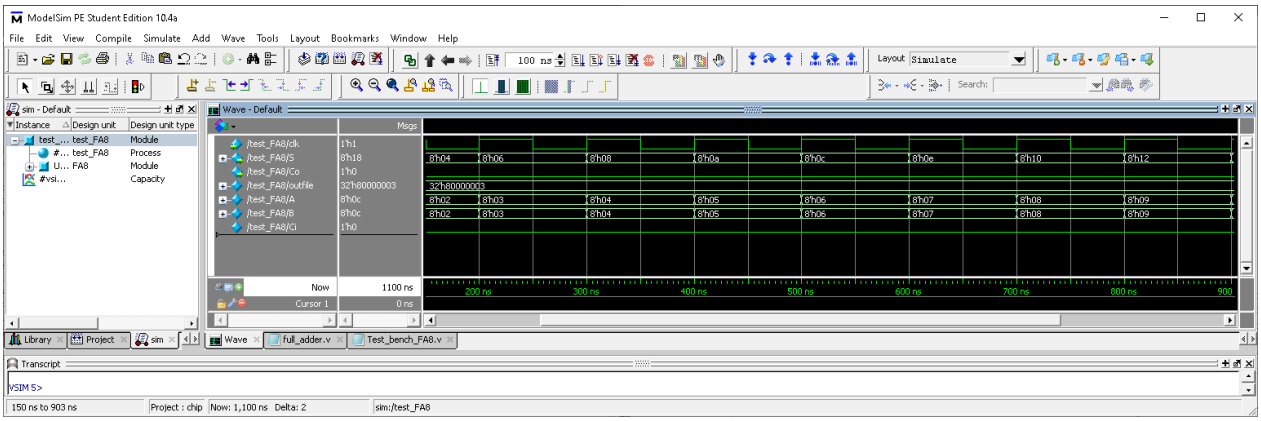


Figura 3.15: Resultados de la simulación realizadas al sumador completo empleando ModelSim.

5. **Planificación de piso:** En esta etapa del diseño, se define la dimensión y relación de

altura y ancho para cada módulo o celda, en relación al resto. Esto con el propósito de que los módulos puedan ser conectados de manera óptima entre sí, y así lograr un mayor porcentaje de ocupación de área disponible. Ciertas características del chip pueden ser ajustadas a nivel de circuito, seleccionando el tamaño de los transistores o usando diferentes estilos de lógica. Un ejemplo claro es el *delay* (retardo) del circuito, donde el planeamiento de piso (generado manual o automáticamente) toma gran importancia porque distintos factores, principalmente el tamaño de los transistores o simples cosas como el ancho de los cables, pueden determinar el *delay*. Cabe destacar que es importante aventurarse al diseño a nivel micro-arquitectura. Una práctica común es escribir código RTL, sintetizarlo (permitiendo que el sintetizador optimice la implementación) y revisar si el resultado es suficientemente rápido. Si no lo es, el diseñador modificará el código añadiendo paralelismo, *pipelines* o cambiando el algoritmo y repetir el proceso hasta conseguir lo deseado.

6. **Colocación**: Ahora, las ubicaciones reales de las instancias de las celdas estándar que se colocarán en el área central del chip, se determinan para todo el diseño. Por lo general en este proceso se colocan las celdas estándar en fila. Una ubicación deficiente requiere un área más grande y también degrada el rendimiento. Se deben tener en cuenta varios factores, como el requisito de tiempo, las longitudes netas y las conexiones de las celdas.

7. **Enrutamiento**: El enrutamiento es el proceso de crear conexiones físicas entre los puertos de las celdas estándar o dispositivos que deban estar conectados. Estas interconexiones metálicas de enrutamiento deben cumplir con los requisitos de temporización, *skew* del reloj y también requisitos físicos para una correcta fabricación.

La suite de Cadence cuenta con el software *Encounter Digital Implementation System* (EDI), el cual puede realizar los últimos 3 pasos (planificación de piso, colocación y enrutamiento) tomando como entrada el archivo que contiene la descripción HDL del circuito sintetizado para posteriormente generar el diseño físico del circuito (*layout*), colocando las celdas descritas, sus interconexiones y los pines de salida y entrada. Este diseño es descrito en formato GDS. En la Figura 3.16 se muestra el resultado de generar el layout del sumador de 8 bits usando la herramienta Encounter.

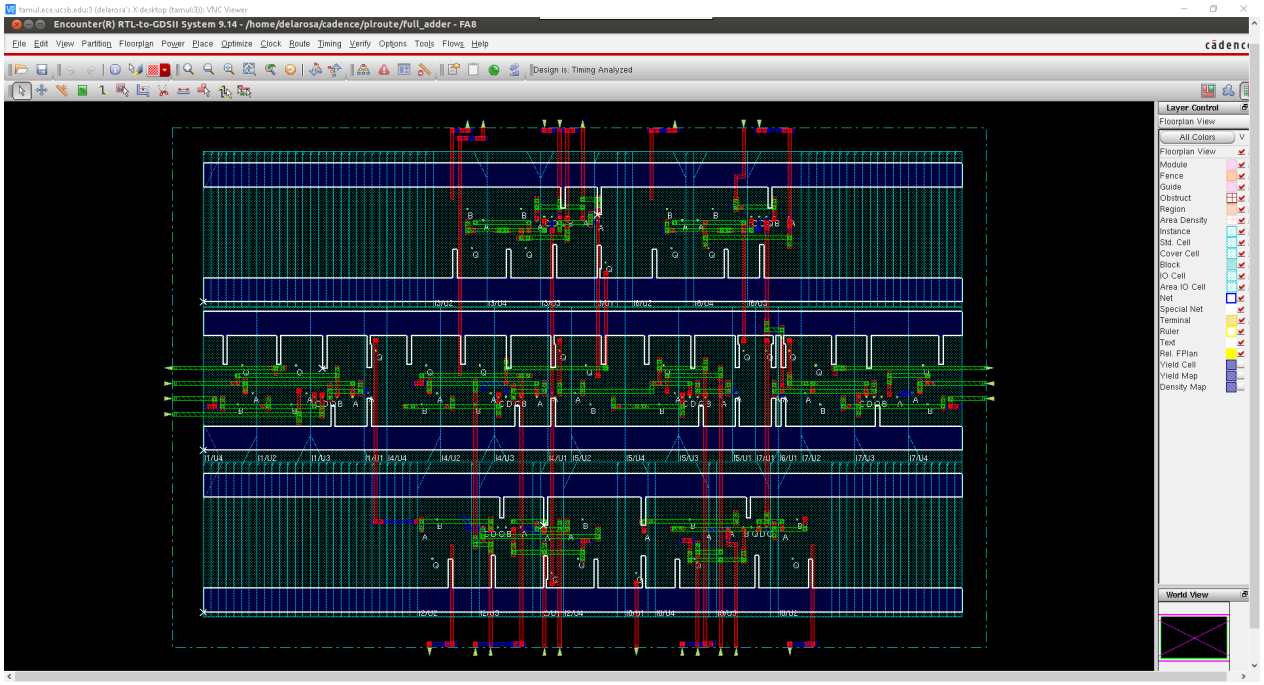


Figura 3.16: Diagrama de diseño para la implementación de un sumador completo de 8 bits generado por *Encounter*.

8. **Verificación de diseño**: Los circuitos integrados pueden llegar a ser lo suficientemente complicados, que si algo puede salir mal, seguramente lo hará [2]. La verificación del diseño es importante para resolver problemas, detectar posibles comportamientos erróneos o mejorar ciertos aspectos antes de la fabricación del chip y comúnmente representa la mitad o más del esfuerzo dedicado a un circuito. El diseño generalmente se prueba para la funcionalidad a nivel arquitectónico con un modelo en un lenguaje de programación de alto nivel como C y a nivel lógico simulando la descripción en HDL. Luego, se comprueba los circuitos para asegurarse de que sean una representación igual a la lógica y se verifica el diseño (*layout*) para garantizar que sea una representación idéntica a la de los circuitos.

   Para comprobar la correspondencia eléctrica del circuito del *layout* con el esquema del circuito, se ejecuta la comprobación LVS (*Layout versus Schematic*). A partir de estas vistas se genera un *netlist* para cada una en formato SPICE para posteriormente compararlos y corroborar que ambos diseños son equivalentes. Es importante mencionar que este proceso no indica si el diseño del esquema o *layout*, son correctos, únicamente verifica la igualdad entre ambos diseños. En la Figura 3.17 se muestra la ventana de dialogo de la herramienta para realizar la comprobación LVS integrada en *Virtuoso*.

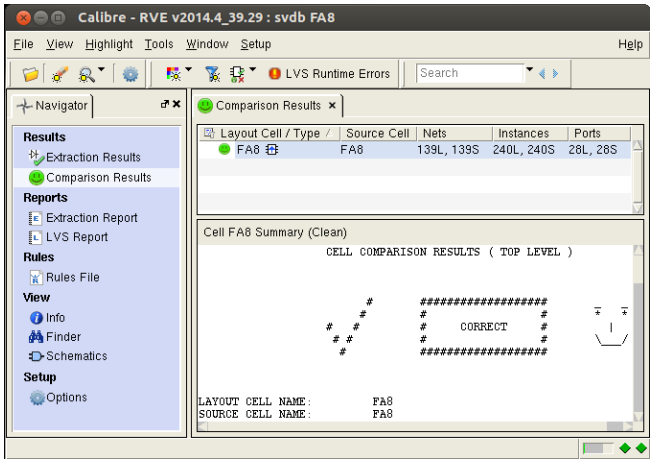


Figura 3.17: Comprobación LVS para suamdor, empleando la herramienta *Calibre* integrada en *Virtuoso*.

La comprobación DRC (*Design Rule Check*), consiste en verificar que el circuito se puede fabricar según las especificaciones del fabricante de la tecnología. Es decir, hay que comprobar aspectos como que dos capas de metal no estén demasiado juntas, ancho adecuado de cables, entre otros. Estas normas se deben considerar al momento de desarrollar el layout y se explican en la documentación de la tecnología. En la Figura 3.18 se muestra la ventana de dialogo con el resultado de la comprobación DRC del diseño del sumador, empleando la herramienta integrada en *Virtuoso*.

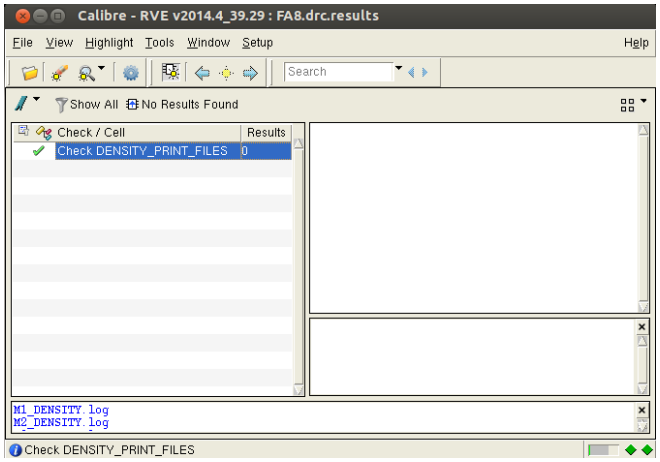


Figura 3.18: Comprobación DRC para suamdor, empleando la herramienta Calibre integrada en Virtuoso.

9. **Simulación posterior al diseño**: Una vez que se haya verificado las reglas de diseño, la equivalencia del esquema y se hayan extraído las resistencias y capacitancias parasiticas, se realiza la simulación del circuito y se verifica nuevamente el diseño. Cabe señalar que

esta simulación generalmente se realiza en varias etapas durante el flujo de diseño, pero se consolida como un solo paso aquí. En la Figura 3.19 se muestra el circuito empleado para hacer la simulación del sumador, el cual se compone de un módulo contador que se conecta a las dos entradas del sumador y el *carry* de entrada conectado a GND. Como salidas se tienen la suma y el *carry* de salida. El bus de la suma y el bit de *carry* están conectados a *buffers* de tal manera que estas señales tengan una carga capacitiva, asemejando que el módulo compone un sistema más grande.

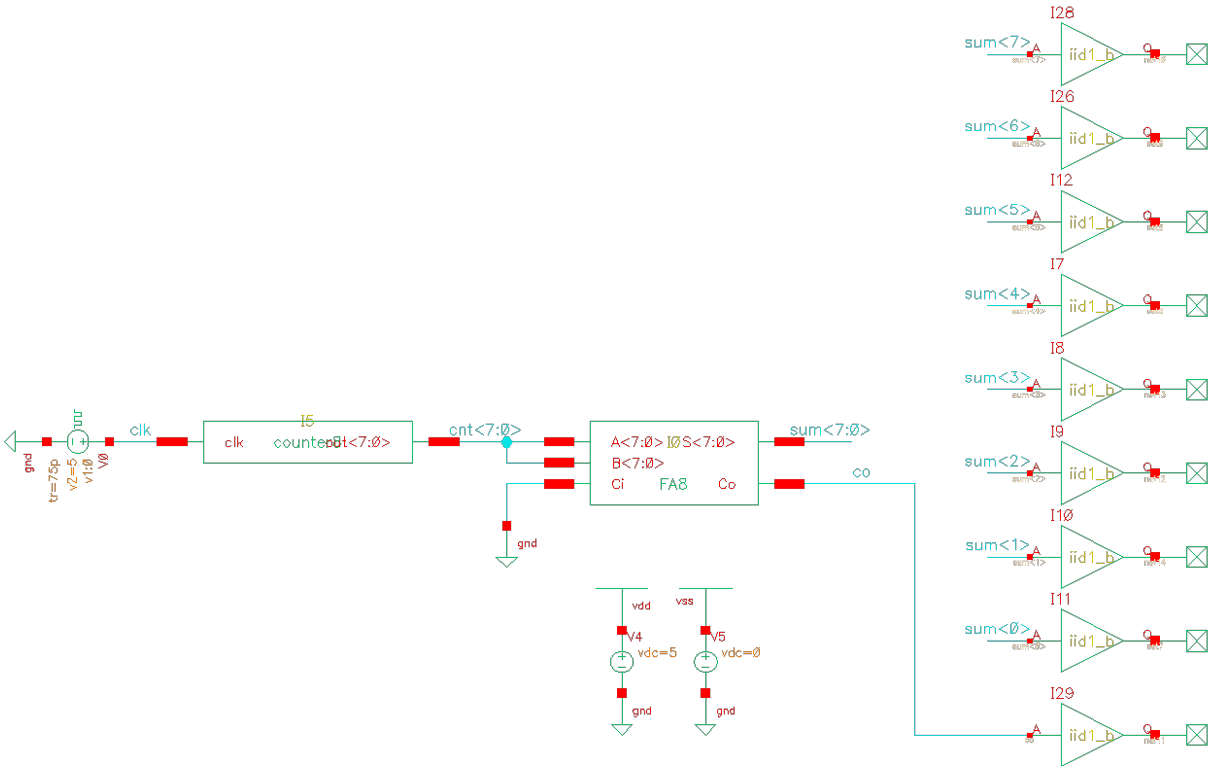


Figura 3.19: Diagrama del esquemático para realizar pruebas en el sumador completo de 8 bits, generado por *Virtuoso*.

En la Figura 3.20 se puede observar 3 buses de datos, el primero a una señal de 8 bits que proviene de un contador, la segunda, la señal del carry de salida y la última el bus de 8 bits que corresponde a la suma. Las señales corresponden a valores analógicos provenientes de la simulación del circuito.

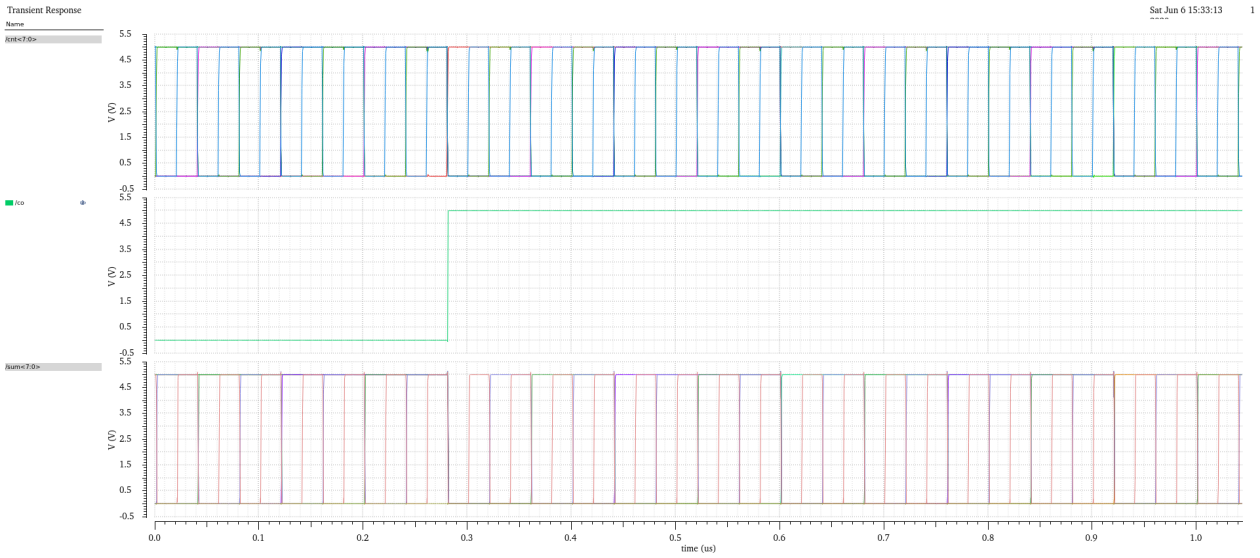


Figura 3.20: Resultados de la simulación para la verificación del correcto funcionamiento del sumador.

Cadence implementa una herramienta llamada "calculadora" en la cual es posible procesar las señales analógicas obtenidas de una simulación. Para el caso de la simulación del circuito de la Figura 3.19, procesamos las señales **cnt**$< 7:0 >$, **sum**$< 7:0 >$ y **co**, realizando un muestreo y una conversión de analógico a digital. Una vez procesada cada señal, podemos ver el bus de información, como se muestra en la Figura 3.21, donde el bus de color azul corresponde a el contador, las de color verde a la suma y la señal en rojo al *carry* de salida.

Figura 3.21: Resultados de la simulación, procesados con la herramienta de calculadora integrada en *Cadence*.

Una vez finalizado el proceso de diseño del ASIC, es posible generar el archivo GDS, el cual contendrá la información del diseño para la etapa de fabricación. El GDSII (*Graphic Design System*) es un formato que la fundición entiende y utiliza para la fabricación del circuito integrado. Los datos de diseño se convierten en máscaras fotolitográficas, una para cada capa. Las máscaras identifican espacios en la oblea, donde ciertos materiales deben depositarse, difundirse o incluso eliminarse. Los cristales de silicio se cultivan y se cortan para producir obleas. Las dimensiones extremadamente pequeñas de los dispositivos VLSI requieren que las obleas se pulan casi a la perfección. El proceso de fabricación consta de varios pasos que implican la deposición y la difusión de diversos materiales en la oblea. Durante cada paso se usa una máscara diferente y se pueden usar varias docenas de máscaras para completar el proceso de fabricación. En la Figura 3.22 se muestra el diseño final del sumador.

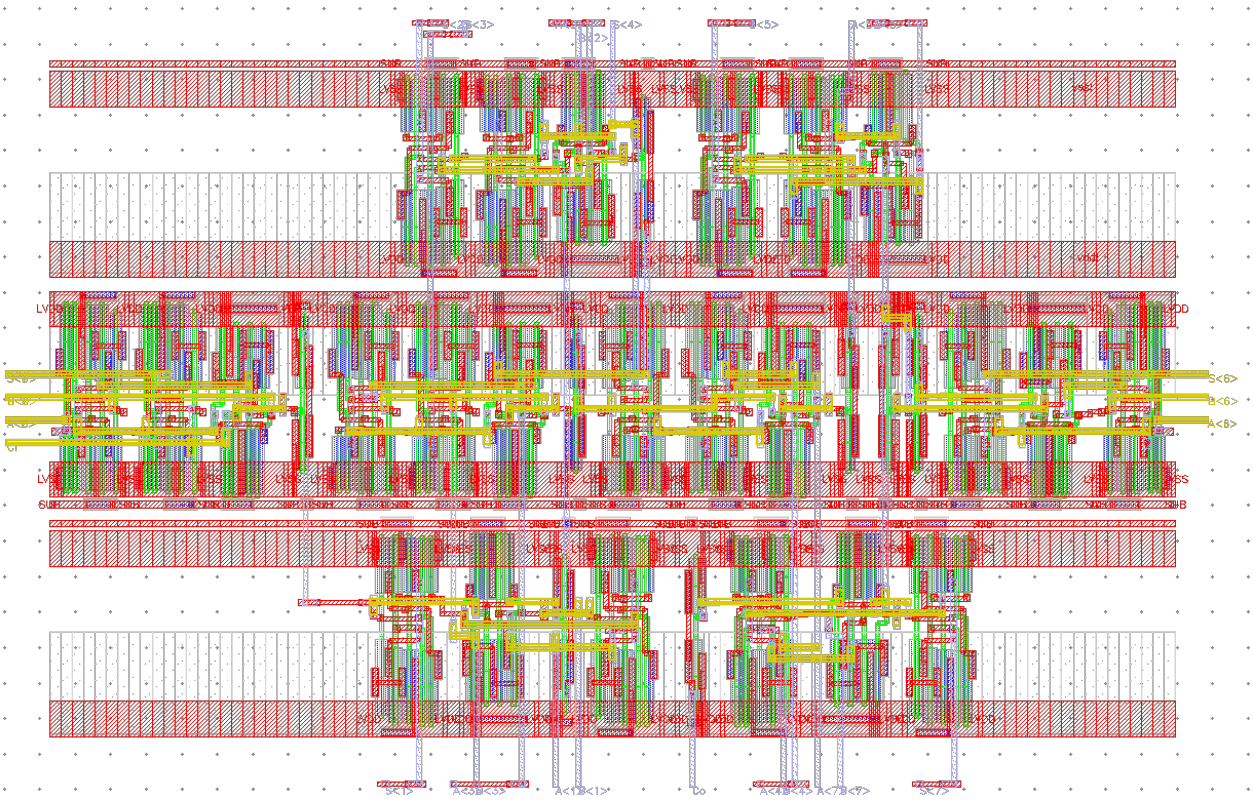


Figura 3.22: Diseño físico final para el sumador completo de 8 bits.

Finalmente, la oblea es fabricada y se corta en chips individuales. La fabricación de circuitos integrados se basa en el proceso de fabricación de obleas (generalmente obleas de silicio), que implica cientos de pasos de procesamiento e incluye varios ciclos fotolitográficos [24]. Cada chip se empaqueta y se prueba para garantizar que cumpla con todas las especificaciones de diseño y se verifica su funcionalidad.

Capítulo 4

Implementación ESCA ASIC

En [25] se implementó numéricamente una versión del sistema ESCA para el cifrado de señales de audio y vídeo en tiempo real, demostrando que dicha implementación requiere un bajo tiempo de procesado y que no hay correlación entre la señal original y la señal cifrada. En [15] se implementó en software, una versión matricial y modificada del sistema, enfocado al cifrado de imágenes. En ambos trabajos se han reportado distintas pruebas estadísticas, las cuales demuestran un alto nivel de seguridad en la información cifrada. El sistema ESCA, inicialmente fue diseñado para ser implementado en software, sin embargo, en trabajos como [26] se realizaron implementaciones en un FPGA, abriendo una brecha para investigar el desempeño del sistema de cifrado a través de su implementación en tecnología ASIC. En este trabajo se propuso la implementación en tecnología CMOS de $0.5\,\mu$m del sistema de cifrado ESCA reportado en [15], añadiendo una caja de sustitución basada en autómatas celulares.

En este capítulo se describen las implementaciones lógicas y físicas (en tecnología CMOS) de cada módulo que compone el sistema de cifrado, además de la descripción del algoritmo y parámetros necesarios para el funcionamiento de la implementación, así como las opciones que posee para ser re-configurado y los modos de operación que corresponden a los bloques de las transformaciones Φ y Ψ, y el PRNG. También, se describe el protocolo de comunicación empleado para la transferencia de datos entre el chip y el mundo exterior, y el flujo de datos a través del sistema de cifrado. Por último, se presentan otros diseños implementados en una tecnología de pseudo-CMOS (únicamente nMOS) de $5\,\mu$m. Estos diseños *full-custom* corresponden a un anillo oscilador de 19 etapas, un decodificador de 2 bits y dos sumadores completos de 2 bits implementados con compuertas NAND y NOR.

4.1. Implementación del sistema ESCA

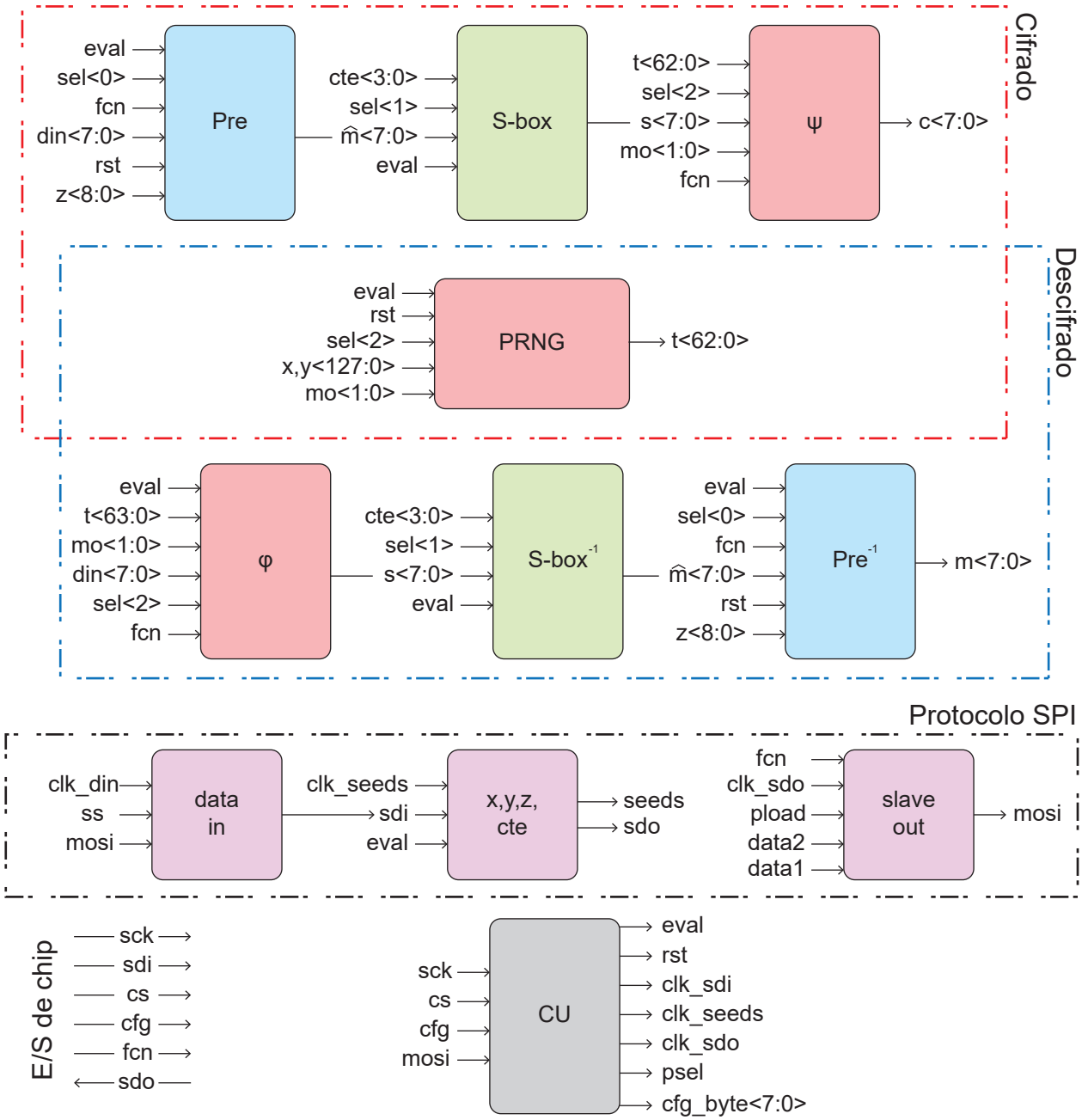


Figura 4.1: Diagrama esquemático del ESCA ASIC.

Implementar un sistema de cifrado conlleva un continuo *trade-off* debido a que un sistema complejo y avanzado implica mayor consumo de recursos como tiempo de procesado, consumo energético o chips más grandes, mientras que para el caso contrario, es decir, un sistema más

simple puede llegar a presentar problemas de seguridad como ser vulnerable ante ataques o que la información sea descifrable fácilmente. Debido a esto, en la implementación del sistema de cifrado ESCA, se consideró un adecuado balance entre ambas situaciones, dando lugar a la implementación que se muestra en la Figura 4.1.

Como se describió en el Capítulo 2, la estructura del sistema de cifrado consta de 7 módulos: un PRNG, un bloque para cifrar, un bloque para descifrar, una caja de sustitución con su correspondiente módulo inverso y el módulo de pre-procesado y su módulo inverso. Para cifrar datos se emplean los módulos de pre-procesado, S-box, Ψ y PRNG. En el caso contrario se emplean los módulos de Pre-procesado inverso, S-box inversa, Φ y PRNG. El control del flujo de datos, señales para la operación del sistema y otras, son generadas por una unidad de control. Por último, se emplearon registros de corrimiento para ingresar de manera serial la información de semillas del sistema de cifrado, bits para configurar el sistema y los datos que se procesarán, así como para extraer la información cifrada odescifrada. Estos registros utilizan el protocolo SPI (*Serial Peripheral Interface*) para la transferencia de datos.

Dicha implementación procesa bloques de 8 bits y puede utilizar uno, dos o tres módulos correspondientes a la tarea indicada (cifrado o descifrado), debido a que el sistema fue diseñado para realizar un *by-pass* en cualquier bloque. Por ejemplo, es posible cifrar información utilizando únicamente la caja de sustitución y la permutación Ψ, o procesar los datos únicamente haciendo uso del bloque de pre-procesado. En adición al *by-pass* aplicado a los bloques del sistema, se implementó la posibilidad de re-configurar los bloques de las permutaciones Ψ, Φ y PRNG, para cambiar el tamaño de las llaves necesarias en los bloques Ψ y Φ.

A continuación se describe el funcionamiento de cada módulo así como su estructura y las señales de entrada necesarias para el correcto funcionamiento.

4.1.1. PRNG

Como se explicó en el Capítulo 2, se implementó una variante del PRNG en la cual es posible cambiar el tamaño del número pseudo-aleatorio generado el cual puede ser de 15, 31 o 63 bits según sea configurado el módulo del PRNG, también se re-configuran los bloques Ψ y Φ para que operen con el tamaño de llave correcto.

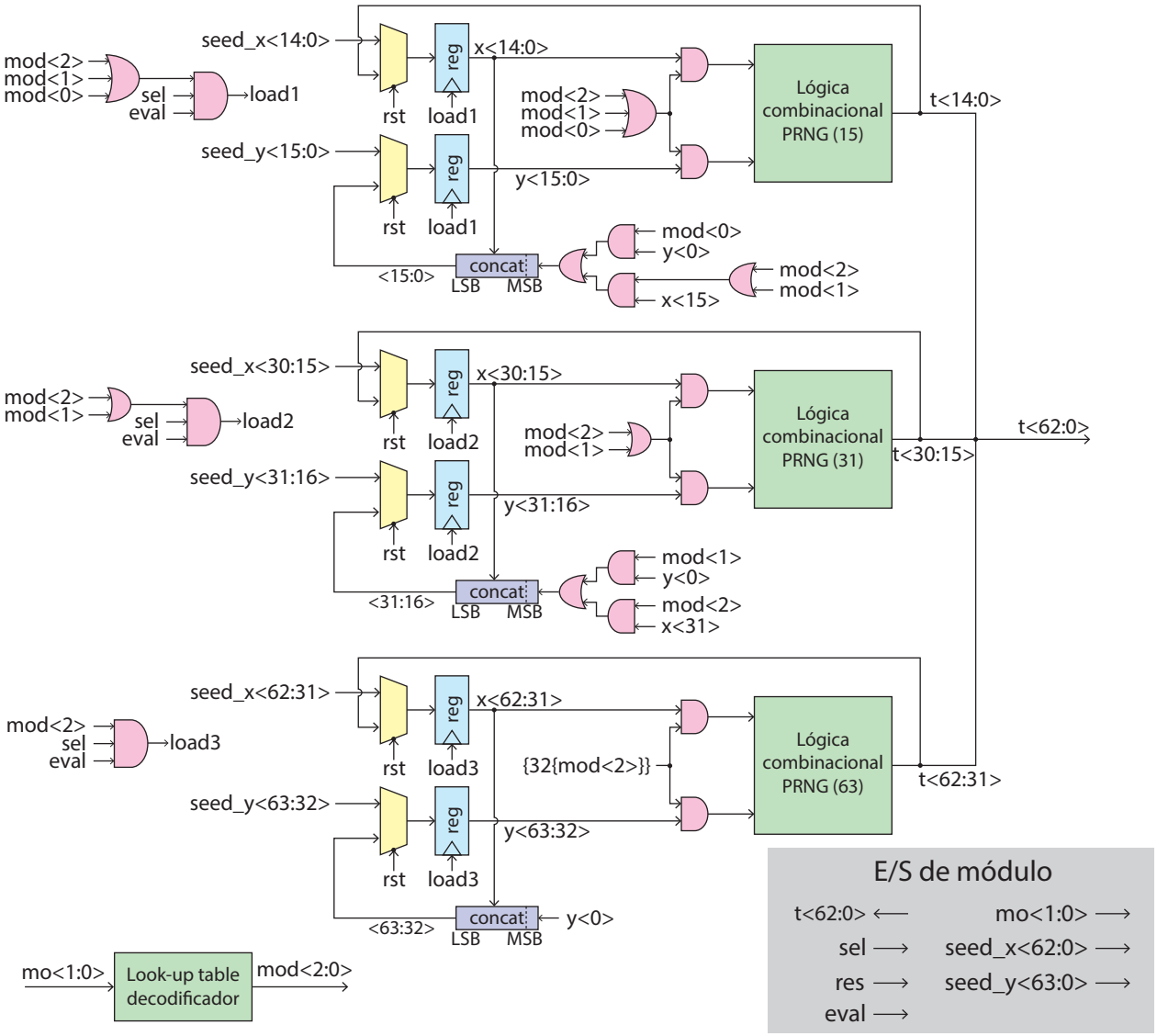


Figura 4.2: Diagrama esquemático de PRNG modular.

En la Figura 4.2 se muestra la implementación del PRNG, la cual consta de las siguientes entradas:

- **sel**: Se encarga de permitir el flujo de datos de los registros que contienen las semillas para la generación del número pseudo-aleatorio. Cuando esta señal es 0, los registros no se actualizan a la salida. Cuando es 1, los registros pueden cambiar de valor para la generación de la llave.
- **rst**: Esta señal indica si el generador generará el primer número con las condiciones iniciales (1 lógico) o los números generados serán retroalimentados (0 lógico).

- **eval**: Esta señal se emplea para cargar nuevos valores en los registros y generar un nuevo número pseudo-aleatorio en los flancos de subida.

- **mo**$< 1 : 0 >$: Estas señales corresponden al modo de operación con el cual se desea operar y se emplean en una tabla de búsqueda de 2 a 3 bits. Esta señal de 3 bits es la que define el tamaño del número pseudoaleatorio generado.

- **seed x**$< 62 : 0 >$ y **seed y**$< 63 : 0 >$: Estos buses de datos corresponden a las condiciones iniciales para generar el primer número pseudo-aleatorio.

- **t**$< 62 : 0 >$: Este bus corresponde a la salida de los 3 bloques combinacionales que conformar el número pseudo-aleatorio generado.

La estructura del PRNG puede ser vista como 3 bloques distintos, cada uno genera una parte del número pseudo-aleatorio, es decir, cada bloque representa una escala mas en el tamaño (15, 31 y 63 bits). Para generar el número de 15 bits, únicamente se emplea el bloque superior, para generar el número de 31 bits, se emplean el bloque superior y el central, finalmente, para generar el número de 63 bits su emplean los 3 bloques, concatenando su salida para formar el número final. En el caso de los números de 15 y 31 bits, el resto de bits que completan la cadena de 63, permanecen con el mismo valor (no hay cambios en el bloque combinacional), evitando así un mayor consumo de energía.

La entrada de los registros que alimentan a los bloques combinacionales, proviene de un multiplexor (como se muestra en la Figura 4.2), el cual selecciona entre valores iniciales de un registro de corrimiento (ajeno al PRNG) y la retroalimentación de la semilla x y la llave generada. Esto se realiza mediante un sencillo arreglo de compuertas, el cual selecciona si debe retroalimentar información como si el bloque fuera parte de uno más grande o este bloque fuese el último para generar la llave (ordenados de mayor a menor según el tamaño de la llave a generar), por ejemplo, si el módulo de PRNG es configurado para generar números de 31 bits, se emplea el bloque superior y el del centro. En el bloque superior la retroalimentación al registro de x es la salida del bloque combinacional (número pseudo-aleaotrio) y la retroalimentación al registro de y corresponde a los 15 bits de la salida del registro x con el bit $x < 15 >$ como el bit más significativo (si se generaran llaves de 15 bits, el bit MSB sería $y < 0 >$). En el bloque del centro la retroalimentación es similar, el registro de x se retroalimenta con la salida del bloque combinacional y el registro de y se retroalimenta con la salida del registro de x y el bit MSB es el bit $y < 0 >$ (si el tamaño de la llave fuera de 63 bits, el bit MSB sería $x < 31 >$).

En la siguiente tabla se muestran los modos obtenidos a partir del decodificador de 2 bits:

En el modo $mo < 1 : 0 >= 11$ los registros no actualizan su salida, por lo que los bloques combinacionales no generan ningún cambio a la salida, estos valores van a depender de los estados con los que se inicializan los registros cuando el dispositivo sea energizado.

mo<1>	mo<0>	mod<2>	mod<1>	mod<0>	Tamaño de llave generada
0	0	0	0	1	15 bits
0	1	0	1	0	31 bits
1	0	1	0	0	63 bits
1	1	0	0	0	0 bits

Tabla 4.1: Modos de operación de las permutaciones Ψ, Φ y PRNG.

4.1.2. Permutaciones Ψ y Φ

Las estructuras de las permutaciones Ψ y Φ dependen del tamaño de llave con la cual se procesaran los datos. En este caso, debido a la manera en que son implementados dichos bloques, no es posible reutilizar bloques que necesiten una llave de menor tamaño como en el PRNG, por lo que se empleo un módulo que a su vez se compone de 3 módulos individuales para cada tamaño de llave. En la Figura 4.3 se muestra la implementación del bloque de cifrado, la cual requiere las siguientes entradas:

- **sel**: Selecciona entre el dato cifrado y el dato de entrada para el multiplexor que corresponde a la salida del módulo.
- **fcn**: Permite el flujo de la llave hacia el bloque combinacional cuando el estado de esta señal es un 1 lógico.
- **mo**$< 1 : 0 >$: Este bus corresponde al modo de operación con el cual se desea operar. Empleando una tabla de búsqueda de 2 a 3 bits se define que bloque combinacional se usará, permitiendo el flujo de datos a través de una compuerta AND.
- **t**$< 62 : 0 >$: Este bus de datos corresponde a la llave empleada en los bloques combinacionales.
- **s**$< 7 : 0 >$: Este bus corresponden al byte que se cifrará, proveniente de la S-box.
- **c**$< 7 : 0 >$: Este bus pertenece a la salida del multiplexor final, el cual selecciona entre el dato cifrado y el dato de entrada.

La estructura del módulo, se conforma principalmente por 3 bloques combinacionales que corresponden a un bloque de cifrado para una longitud de llave distinta (63, 31 o 15 bits). Las entradas de dichos bloques provienen de compuertas NAND que permiten el flujo de los datos, dependiendo de los bits del modo de operación y la señal **fcn**. Al igual que el módulo de PRNG, cuenta con el mismo decodificador de 2 bits para extraer la señal **mode**$< 2 : 0 >$. Por último, se encuentra un multiplexor que selecciona entra el dato de entrada **s**$< 7 : 0 >$ y un *oring* de las salidas de los bloques combinacionales de la permutaciones Ψ de diferentes tamaños.

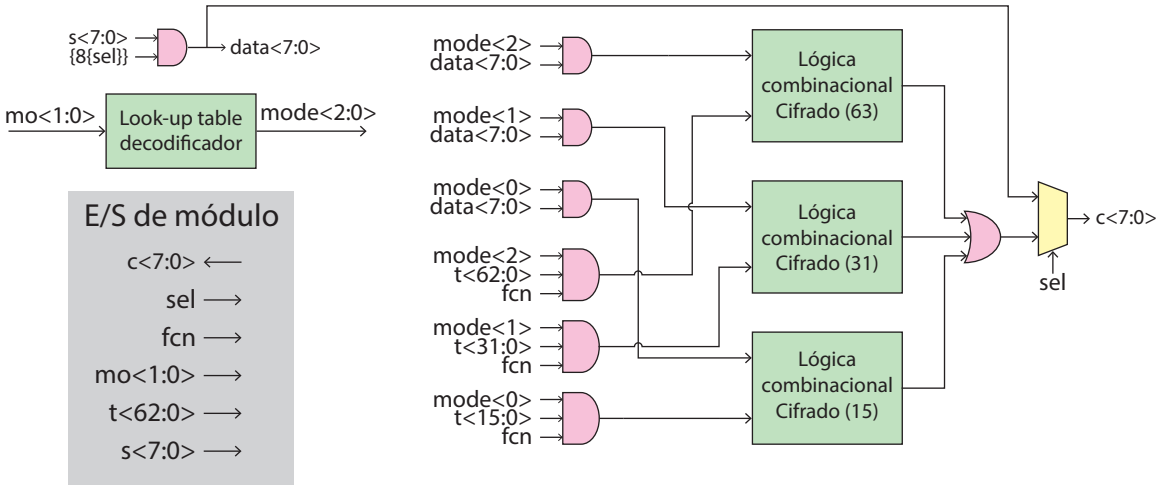


Figura 4.3: Diagrama esquemático del bloque de cifrado con las estructuras Ψ.

En la Figura 4.4 se muestra la estructura de módulo de cifrado, la cual es muy similar a la de la permutación Ψ. La diferencia entre ambos bloques radica en que el dato de entrada **din**$< 7:0 >$ esta conectado a 8 *latches*, los cuales permiten el flujo de la información cuando las señales **eval** y **fcn** son 1, es decir, el chip está configurado para descifrar y la señal **eval** indica que se evaluará un nuevo dato en los flancos de subida.

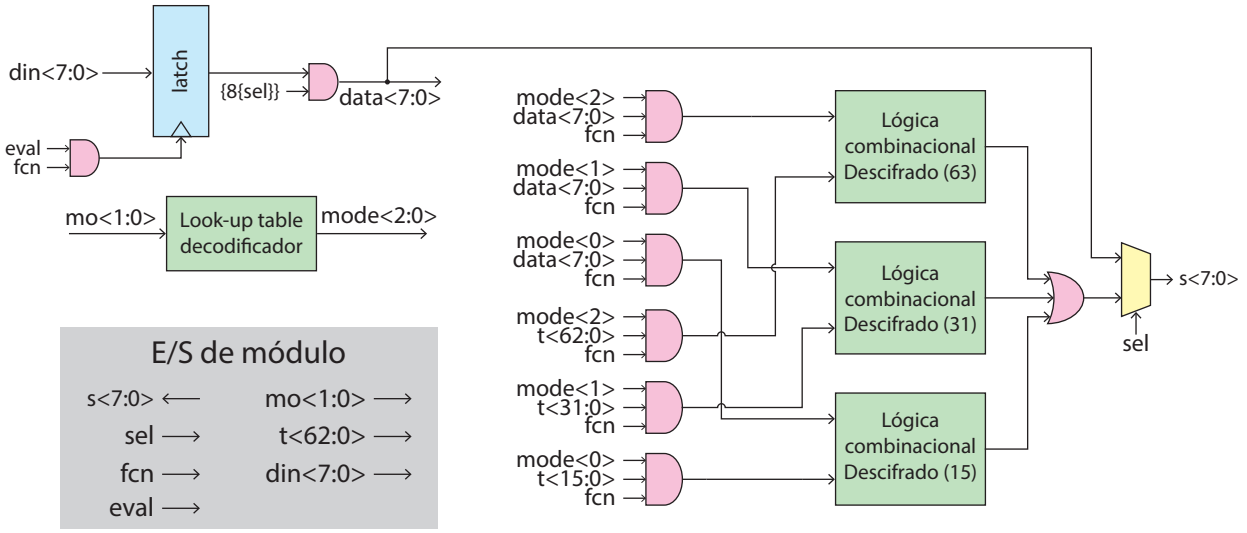


Figura 4.4: Diagrama esquemático del bloque de descifrado con las estructuras Φ.

4.1.3. Pre-procesado

La estructura del módulo de pre-procesado se muestra en la Figura 4.5 y contiene las siguientes entradas y salidas:

- **eval**: Esta señal se emplea para transmitir información a través de *latches* y registros.
- **sel**: Esta señal selecciona entre el dato cifrado y el dato de entrada para la salida de un multiplexor que corresponde a la salida del módulo.
- **fcn**: Permite el flujo de los datos de entrada y la semilla cuando **fcn** es 1.
- **rst**: Esta señal se emplea para cargar los valores iniciales o continuar con la retroalimentación, en los registros a través de un multiplexor.
- **z**$< 8 : 0 >$: Este bus corresponde a las condiciones iniciales (semillas) que emplea el bloque combinacional.
- **din**$< 7 : 0 >$: Este bus corresponde al byte de entrada.
- $\hat{m} < 7 : 0 >$: Este bus corresponde al bus de salida del multiplexor final.

La parte principal del módulo es un bloque combinacional que contiene las ecuaciones para el pre-procesado. Este bloque tiene dos entradas, el dato a procesar y una semilla. El dato de entrada proviene de un arreglo de *latches* que evita la propagación de los datos en el bloque combinacional cuando no es necesario. La semilla proviene de un arreglo de registros, que al igual que los *latches* solo es transparente (la salida se actualiza cuando hay un cambio en la entrada) cuando se evaluare un nuevo dato. La entrada de este registro proviene de un multiplexor el cual selecciona entre la condición inicial y la retroalimentación. De manera similar al PRNG, el bus para la retroalimentación se compone de la salida como los bits menos significativos y el bit más significativo toma el lugar de z$< 0 >$ anterior (salida del registro antes de evaluar).

La estructura del pre-procesado inverso es básicamente la misma que el pre-procesado, pero en este caso el arreglo de *latches* ha sido suprimido, el dato de entrada depende únicamente de las señales **sel** y **fcn**. También, para la retroalimentación se toman los bits de entrada para reemplazar los bits menos significativos. La estructura del módulo de pre-procesado inverso se muestra en la Figura 4.6.

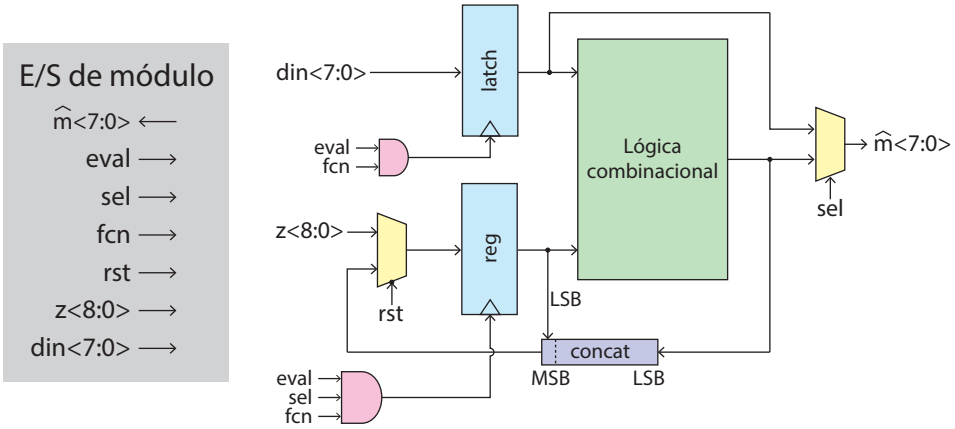


Figura 4.5: Diagrama esquemático del bloque de Pre-procesado.

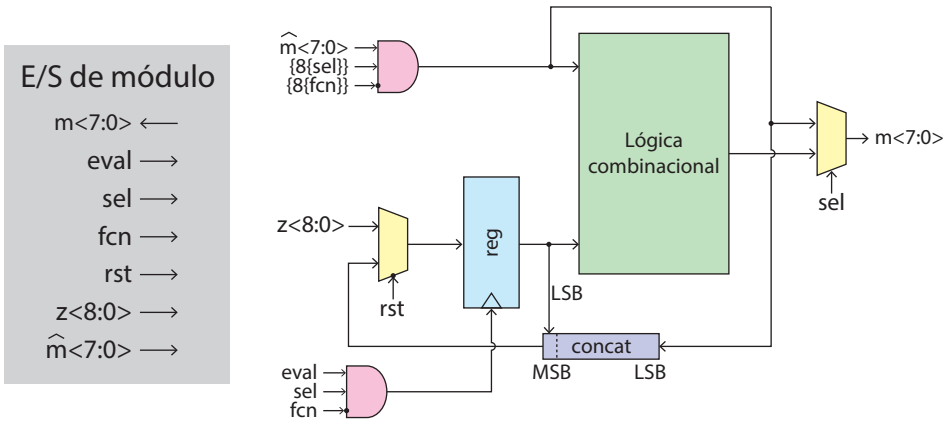


Figura 4.6: Diagrama esquemático del bloque de Pre-procesado inverso.

4.1.4. Caja de sustitución

La estructura de esta implementación se conforma principalmente por dos tablas de búsqueda (ver Figura 4.7), la S-box y la tabla de constantes. La S-box recibe como entrada el byte a ser sustituido y cada señal esta condicionada por una compuerta AND con la señal **sel**. La tabla de la constante recibe un número de 4 bits proveniente de *latches*, y la salida es un número de 8 bits con el cual se realiza la operación XOR, obteniendo como resultado la sustitución final del byte. Este valor entra a un multiplexor junto con el byte de entrada para tener la opción de realizar el *by-pass* del módulo.

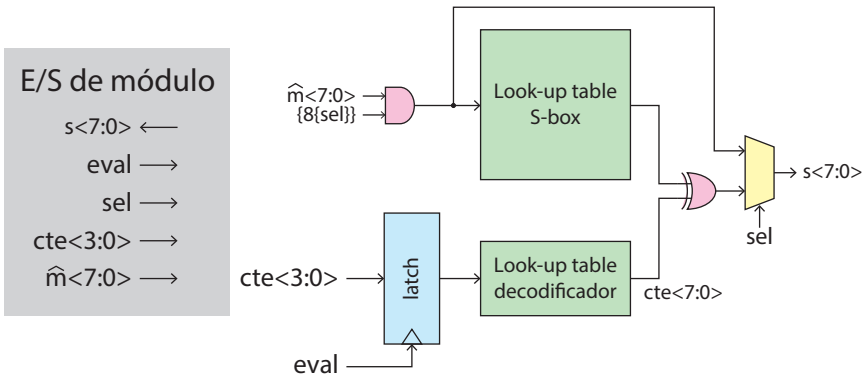


Figura 4.7: Diagrama esquemático del bloque de S-box.

Las entradas y salidas del módulo son la siguientes:

- **eval**: Esta señal se emplea para transmitir el dato de entrada a través de los *latches*.
- **sel**: Esta señal selecciona entre el dato cifrado y el dato de entrada para la salida de un multiplexor que corresponde a la salida del módulo.
- **cte**$< 3 : 0 >$: Este bus se emplea para decodificar la constante con la que se realizará la operación XOR.
- $\hat{m} < 7 : 0 >$: Bus del byte de entrada.
- **s**$< 7 : 0 >$: Bus de salida del multiplexor final.

En la Figura 4.8 se muestra la estructura del módulo de la S-box inversa y al igual que la S-box, sus principales componentes son la S-box inversa y la tabla de constantes. A diferencia del módulo de S-box, la operación XOR se realiza con el dato de entrada y este resultado entra a la tabla de búsqueda para sustituir el byte. La salida de dicho bloque entra al multiplexor junto con el dato de entrada para realizar el *by-pass* del módulo.

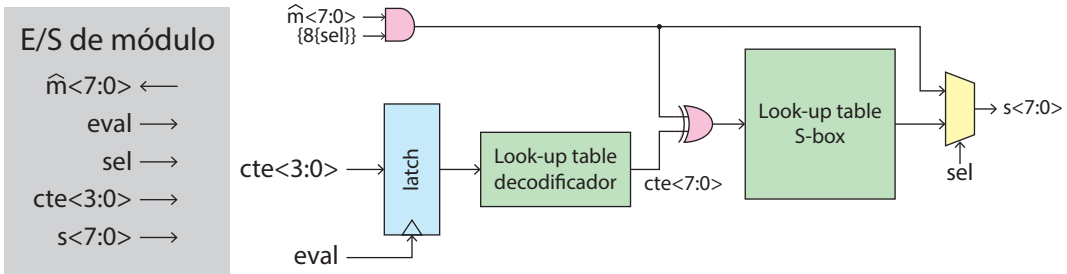


Figura 4.8: Diagrama esquemático del bloque de S-box^{-1}.

Las entradas y salidas son las mismas que el módulo de S-box pero los nombres de los buses del dato de entrada y salida diferente (**s**$< 7:0 >$ y $\hat{m} < 7:0 >$ respectivamente).

4.1.5. Registros de entrada y salida

La comunicación con los registros del sistema, se implementó mediante el protocolo SPI (*Serial Peripheral Interface*). El protocolo SPI es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos [27]. Este es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits en serie regulado por un reloj (comunicación sincrónica). Como se muestra en la Figura 4.9 se emplea una señal de reloj (**sck**), dato de entrada (**sdi**), dato de salida (**sdo**) y un pin de *chip select* (**cs**), que conecta o desconecta la operación del dispositivo con el que se desea establecer comunicación.

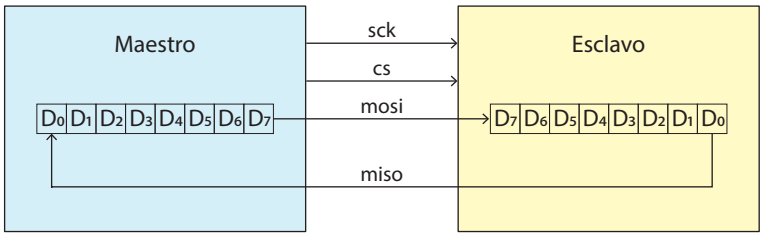


Figura 4.9: Esquema del protocolo SPI.

En la Figura 4.10 se muestra la arquitectura de los registros empleados para almacenar las condiciones iniciales del sistema y los datos de entrada. Se compone básicamente de dos registros, un registros SIPO (*Serial-Input Parallel-Output*) de 148 Flip-Flops, que guarda el dato de entrada, la semilla z, la constante para la S-box, la semilla x y y, y un registro PISO (*Parallel-Input Serial-Output*) el cual cuenta con multiplexores a las entradas de los flip-flops para poder cargar el byte procesado y para transmitir esa información de manera serial a través de los mismos flip-flops y con ello generar la señal serial de salida (**sdo**).

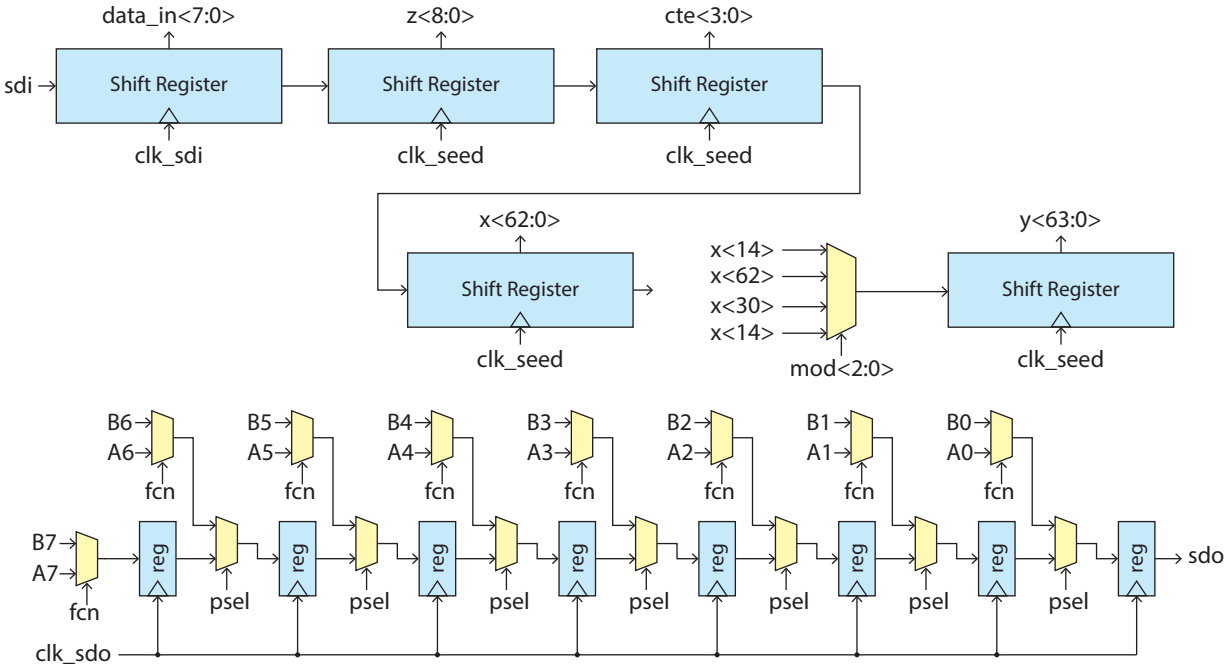


Figura 4.10: Diagrama esquemático de registros de corrimiento del sistema.

A la entrada del registro de la semilla y se empleó un multiplexor para ingresar datos que provienen del registro de x. En las entradas del multiplexor se encuentra la salida de los flip-flops que corresponden a $\mathbf{x}< 14 >$, $\mathbf{x}< 30 >$ y $\mathbf{x}< 62 >$, debido a los distintos modos de operación para el PRNG, donde el tamaño de las llaves puede variar. Con esto, solo los flip-flops necesarios tendrán información valida. Las entradas y salidas son las siguientes:

- **sdi**: Señal de entrada serial.
- **clk_sdi**: Señal de reloj para datos seriales.
- **clk_seed**: Señal de reloj de datos seriales de semillas.
- **mod**$< 2 : 0 >$: Bus para configuración de permutaciones y PRNG.
- **A**$< 7 : 0 >$: Bus de datos cifrados.
- **B**$< 7 : 0 >$: Bus de datos descifrados.
- **fcn**: Señal de función de operación (cifrado o descifrado).
- **psel**: Señal para cargar datos en paralelo o serial.
- **clk_sdo**: Señal de reloj para registros SIPO.

- **data_in**$< 7 : 0 >$: Bus de datos de entrada para procesar.
- **z**$< 8 : 0 >$: Semilla z.
- **cte**$< 3 : 0 >$: Dirección para tabla de búsqueda de constante de S-box.
- **x**$< 62 : 0 >$: Semilla x.
- **y**$< 63 : 0 >$: Semilla y
- **sdo**: Señal serial de salida.

4.1.6. Unidad de Control

La generación de señales internas para el control y configuración de los módulos, se lleva a cabo en una unidad de control implementada en el chip. Esta unidad se encarga de generar señales para evaluar un nuevo byte, cargar nuevos datos en registros, controlar la interfaz del protocolo de comunicación, entre otras tareas. A continuación se enlista las entradas y salidas del módulo para el control del chip:

- **sck**: Señal de reloj para datos seriales.
- **cs**: *Chip select*.
- **cfg**: Señal para indicar que el chip se encuentra en modo de configuración.
- **sdi**: Señal serial de entrada.
- **eval**: Señal de evaluación.
- **rst**: Esta indica cuando los módulos leen las condiciones iniciales o la retroalimentación.
- **clk_sdi**: Señal de reloj para registros de dato de entrada.
- **clk_seed**: Señal de reloj para registros de semillas.
- **clk_sdo**: Señal de reloj para señal de salida.
- **psel**: Señal de control para multiplexores de registros PISO.
- **cfg_byte**$< 7 : 0 >$: Bit de configuración del sistema de cifrado.

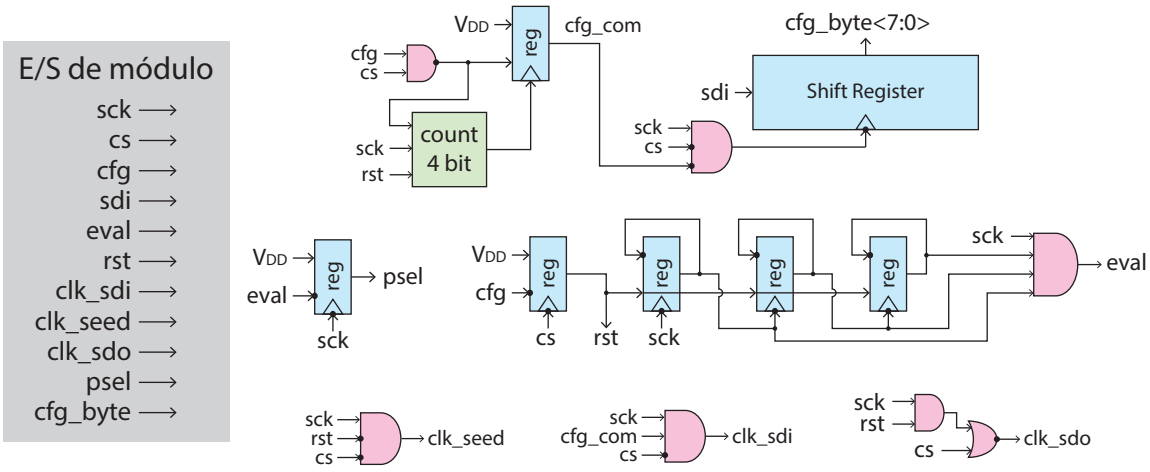


Figura 4.11: Diagrama esquemático de la unidad de control.

En la Figura 4.11 se muestra la arquitectura de la unidad de control, en la cual se puede observar que algunas de las señales son generadas utilizando una simple compuerta AND de 3 entradas, mientras que la carga del byte de configuración, involucra un registro SIPO y circuiteria extra para que este registre la información de los primeros 8 bits que se transmiten al chip. En este byte de configuración se almacena la información del modo de operación (2 bits) y la configuración del sistema con el que se procesará la información, es decir, a qué módulos se realizará el *by-pass* (3 bits). La distribución de los bits de configuración se muestra en la Tabla 4.2.

Bit	Función
1	mod<0>
2	mod<1>
3	Pre-procesado
4	S-box
5	Permutación
6	Reservado
7	Reservado
8	Reservado

Tabla 4.2: Byte de configuración del sistema de cifrado.

4.2. Reporte de resultados de síntesis

La herramienta Design Compiler, genera distintos reportes para cada descripción HDL sintetizada. Estos corresponden a reporte de tiempos, reporte de celdas empleadas, reporte

de área en silicio y reporte de consumo de energía.

La Figura 4.12 muestra el reporte de área, en el cual tenemos información de la cantidad de puertos, interconexiones, celdas combinacionales y secuenciales, *buffers* e inversores y referencias. Además muestra un resumen de la cantidad de área empleada por celdas combinacionales, secuenciales y total, así como la cantidad de área por cada bloque del cual se haya hecho una instancia.

En la Figura 4.13 se muestra un reporte de todas las celdas empleadas para el diseño sintetizado, el área que ocupa cada una de ellas y atributos de la celda como su jerarquía o si es una celda combinacional.

En la Figura 4.14 se muestra el reporte de tiempo del módulo, en el cual podemos ver las compuertas que interconectadas forman el *path* con el mayor tiempo de propagación, así como el menor tiempo de *slack* para el módulo.

En la Figura 4.15 se muestra el reporte de energía donde podemos ver tres apartados distintos, la energía que consume internamente el módulo, la energía usada en un flanco de reloj y la pérdida por fuga en los transistores.

```
****************************************
Report : area
Design : control_unit
Version: P-2019.03
Date   : Tue Mar 10 11:15:07 2020
****************************************

Library(s) Used:

    amis500cxascb (File: /home/delarosa/Tech/amis/lib/amis500cx/amis500cxascb_tc_5.0v_25c.db)

Number of ports:                           58
Number of nets:                            93
Number of cells:                           59
Number of combinational cells:             31
Number of sequential cells:                18
Number of macros/black boxes:               0
Number of buf/inv:                         18
Number of references:                      23

Combinational area:                39.569601
Buf/Inv area:                      17.410600
Noncombinational area:             86.524597
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  126.094198
Total area:                 undefined

Hierarchical area distribution
------------------------------

                                  Global cell area          Local cell area
                                  ------------------  --------------------------
Hierarchical cell                 Absolute   Percent  Combi-    Noncombi-  Black-
                                  Total      Total    national  national   boxes   Design
--------------------------------  ---------  -------  --------  ---------  ------  ---------
control_unit                       126.0942    100.0   39.5696    33.7656  0.0000  control_unit
d1                                   5.2759      4.2    0.0000     5.2759  0.0000  dff_4
m1                                   5.2759      4.2    0.0000     5.2759  0.0000  dff_9
m2                                   5.2759      4.2    0.0000     5.2759  0.0000  dff_8
m3                                   5.2759      4.2    0.0000     5.2759  0.0000  dff_7
m4                                   5.2759      4.2    0.0000     5.2759  0.0000  dff_6
m5                                   5.2759      4.2    0.0000     5.2759  0.0000  dff_5
o1                                   5.2759      4.2    0.0000     5.2759  0.0000  dff_3
o2                                   5.2759      4.2    0.0000     5.2759  0.0000  dff_2
o3                                   5.2759      4.2    0.0000     5.2759  0.0000  dff_1
sdo                                  5.2759      4.2    0.0000     5.2759  0.0000  dff_0
--------------------------------  ---------  -------  --------  ---------  ------  ---------
Total                                                   39.5696    86.5246  0.0000
```

Figura 4.12: Reporte de área generado en el proceso de sintetizado.

```
****************************************
Report : cell
Design : control_unit
Version: P-2019.03
Date   : Tue Mar 10 11:15:07 2020
****************************************

Attributes:
    b - black box (unknown)
    h - hierarchical
    n - noncombinational
    r - removable
    u - contains unmapped logic

Cell                    Reference       Library             Area  Attributes
--------------------------------------------------------------------------------
U3                      no42_b          amis500cxascb   2.901700
U4                      na22_b          amis500cxascb   1.582800
U6                      no32_b          amis500cxascb   2.374100
U7                      no22_b          amis500cxascb   1.582800
U8                      no22_b          amis500cxascb   1.582800
U9                      no22_b          amis500cxascb   1.582800
U10                     no22_b          amis500cxascb   1.582800
U11                     inv1_b          amis500cxascb   0.791400
U12                     inv1_b          amis500cxascb   0.791400
U13                     inv1_b          amis500cxascb   0.791400
U14                     inv1_b          amis500cxascb   0.791400
U16                     inv1_b          amis500cxascb   0.791400
U17                     inv1_b          amis500cxascb   0.791400
U18                     inv1_b          amis500cxascb   0.791400
U19                     inv1_b          amis500cxascb   0.791400
U20                     inv1_b          amis500cxascb   0.791400
U21                     inv1_b          amis500cxascb   0.791400
U22                     an21_b          amis500cxascb   1.582800
U24                     aa21_b          amis500cxascb   1.319000
U25                     aa31_b          amis500cxascb   1.846600
U26                     inv6_b          amis500cxascb   2.110300
U27                     inv1_b          amis500cxascb   0.791400
U33                     cvdd_b          amis500cxascb   1.055200
U34                     an41_b          amis500cxascb   1.846600
U35                     aa21_b          amis500cxascb   1.319000
U36                     iid6_b          amis500cxascb   2.637900
cfg_byte_reg[0]         df001_b         amis500cxascb   4.220700  n
cfg_byte_reg[1]         df001_b         amis500cxascb   4.220700  n
cfg_byte_reg[2]         df001_b         amis500cxascb   4.220700  n
cfg_byte_reg[3]         df001_b         amis500cxascb   4.220700  n
cfg_byte_reg[4]         df001_b         amis500cxascb   4.220700  n
cfg_byte_reg[5]         df001_b         amis500cxascb   4.220700  n
cfg_byte_reg[6]         df001_b         amis500cxascb   4.220700  n
cfg_byte_reg[7]         df001_b         amis500cxascb   4.220700  n
d1                      dff_4                           5.275900  h, n
m1                      dff_9                           5.275900  h, n
--------------------------------------------------------------------------------
Total 49 cells                                          126.094198
```

Figura 4.13: Reporte de celdas generado en el proceso de sintetizado.

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : control_unit
Version: P-2019.03
Date   : Tue Mar 10 11:15:07 2020
****************************************

Operating Conditions: tc_5.0v_25c   Library: amis500cxascb
Wire Load Model Mode: top

  Startpoint: sck (input port clocked by clk)
  Endpoint: eval (output port clocked by clk)
  Path Group: clk
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.30       0.30
  input external delay                     2.00       2.30 f
  sck (in)                                 0.00       2.30 f
  U31/Q (inv1_b)                           0.06       2.36 r
  U32/Q (inv1_b)                           0.26       2.62 f
  U35/Q (aa21_b)                           0.25       2.87 f
  U34/Q (an41_b)                           0.49       3.36 r
  U26/Q (inv6_b)                          13.01      16.38 f
  eval (out)                               0.00      16.38 f
  data arrival time                                  16.38

  clock clk (rise edge)                   20.00      20.00
  clock network delay (ideal)              0.30      20.30
  output external delay                   -1.65      18.65
  data required time                                 18.65
  -----------------------------------------------------------
  data required time                                 18.65
  data arrival time                                 -16.38
  -----------------------------------------------------------
  slack (MET)                                         2.27
```

Figura 4.14: Reporte de tiempo generado en el proceso de sintetizado.

```
****************************************
Report : power
        -analysis_effort low
Design : control_unit
Version: P-2019.03
Date   : Tue Mar 10 11:15:07 2020
****************************************


Library(s) Used:

    amis500cxascb (File: /home/delarosa/Tech/amis/lib/amis500cx/logic/amis500cxascb_tc_5.0v_25c.db)


Operating Conditions: tc_5.0v_25c   Library: amis500cxascb
Wire Load Model Mode: top


Global Operating Voltage = 5
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1uW


  Cell Internal Power  = 448.0626 uW    (7%)
  Net Switching Power  =   5.8690 mW   (93%)
                         ---------
Total Dynamic Power    =   6.3170 mW  (100%)

Cell Leakage Power     = 245.4730 nW

Information: report_power power group summary does not include estimated clock tree power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000              0.0000           0.0000  (   0.00%)
memory             0.0000           0.0000              0.0000           0.0000  (   0.00%)
black_box          0.0000           0.0000              0.0000           0.0000  (   0.00%)
clock_network      0.0000           0.0000              0.0000           0.0000  (   0.00%)
register           0.0000           0.0000              0.0000           0.0000  (   0.00%)
sequential         0.3718        3.0471e-02             0.2455           0.4025  (   6.37%)
combinational  7.6253e-02           5.8385          9.3866e-06           5.9148  (  93.63%)
--------------------------------------------------------------------------------------------------
Total              0.4481 mW        5.8690 mW          0.2455 uW         6.3173 mW
```

Figura 4.15: Reporte de energía generado en el proceso de sintetizado.

En la Tabla 4.3 se muestra un resumen de los reportes generados por el proceso de sintetizado de los módulos del sistema de cifrado y la unidad de control. Cabe destacar que en el proceso de elaboración del planeamiento de piso, colocación y enrutamiento llevado a cabo en el software *Encounter*, este puede optimizar los diseños sintetizados, por lo que los

valores mostrados pueden variar.

Módulo	Área (u)	Celdas	Tiempo (ns)	Energía (mW)
Psi	462.7	217	6.4	3.7
Phi	473	225	7.4	3.3
S-box	1047.8	629	6.6	6.1
S-box^{-1}	1092.1	665	8.8	7.9
Pre	152.7	70	3.5	1.1
Pre^{-1}	158.3	90	5.4	1.4
PRNG	2151.3	871	8.2	16.3
CU	126.1	49	16.3	6.3

Tabla 4.3: Resultados de los reportes generados en el proceso de sintetizado. La unidad u corresponde al área de una compuerta NAND.

4.3. Diseño físico

Este proceso se llevó a cabo de manera local (en cada módulo que compone el sistema) y global (colocando e interconectando los módulos del sistema). Cada módulo tiene una descripción de alto nivel usando Verilog y esta fue sintetizada para posteriormente realizar los procesos de planificación de piso, colocación y enrutamiento, mismos que fueron creados con el software *Encounter* de la suite de *Cadence*. En la Figura 4.16 se muestra la planificación de piso final. En la parte superior se encuentran los registros de las semillas x y y que se conectan directamente al módulo de PRNG. Los módulos de las permutaciones Ψ y Φ están posicionados en la parte inferior del PRNG, debido a que el bus de datos del número pseudo-aleatorio es demasiado grande, por lo que tener los módulos de cifrado y descifrado en una posición alejada al PRNG, involucrarían un área de interconexiones muy grande. Debajo de los módulos mencionados, se encuentran las cajas de sustitución y posterior a estas, el módulo de pre-procesado y su inverso, según sea el caso. En la parte inferior del chip se encuentran el resto de registros y en la parte central, la unidad de control. La parte central del chip, donde no se encuentra ningún módulo se empleo para realizar las interconexiones de la unidad de control con el resto de módulos, ademas del bus del byte de entrada y salida del módulo de cifrado y de descifrado respectivamente.

El diseño final fue sometido a la comprobación DRC, arrojando múltiples errores de diseño debido a que la librería estándar empleada no contaba con las ultimas actualizaciones a las reglas y estos fueron solucionados en su totalidad. Para la comprobación LVS se empleó el layout de la Figura 4.18 y el diagrama esquemático que se muestra en la Figura 4.17, en el cual dos registros (*SIPO_9_halfPitch* y *data_in*) fueron duplicados con el motivo de ahorrar espacio de interconexiones, ya que éste sería más grande que el propio tamaño de los registros. Ambos diseños son equivalentes.

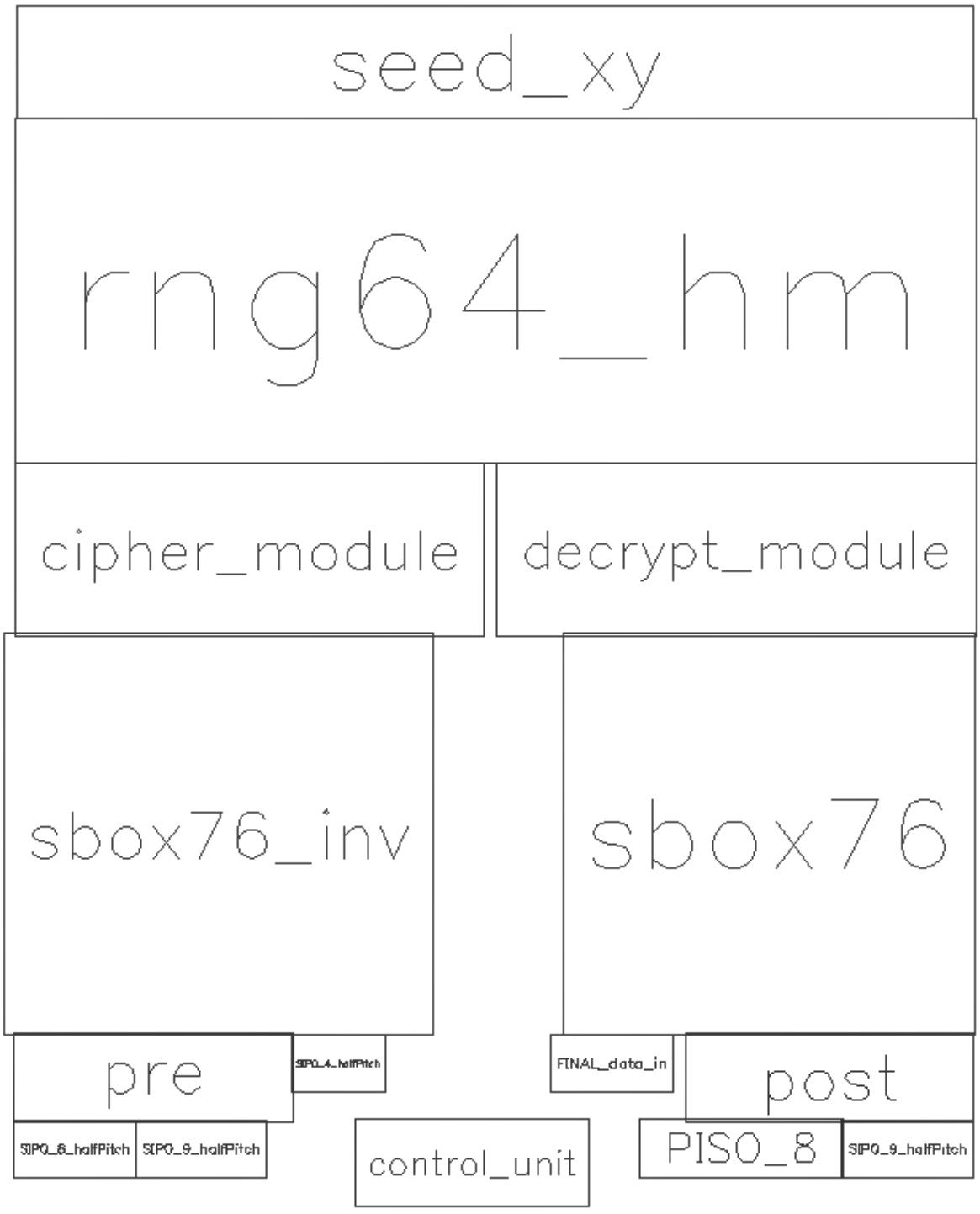


Figura 4.16: Planificación de piso del *core* del ESCA ASIC.

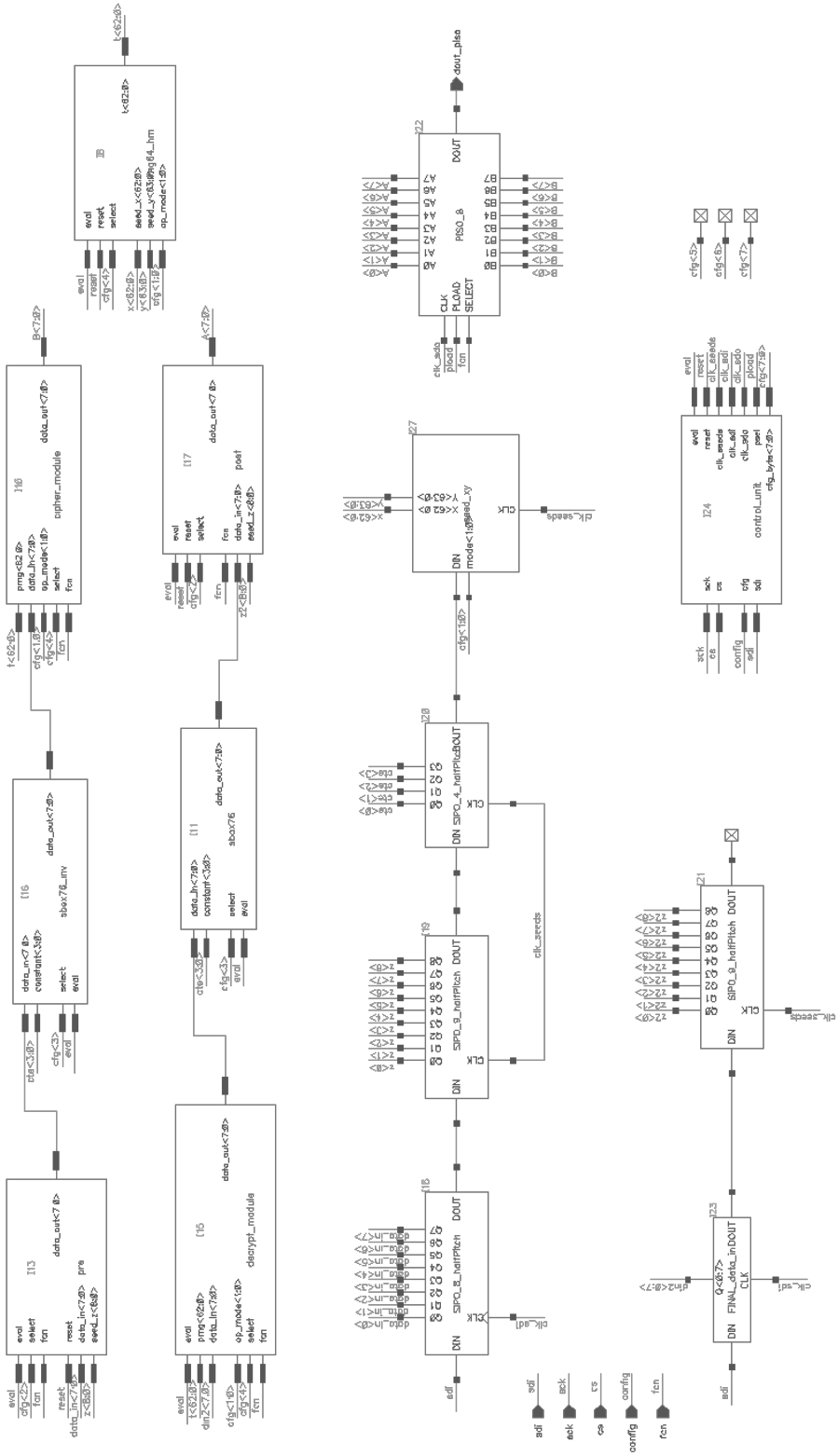


Figura 4.17: Diagrama esquemático del *core* del ESCA ASIC.

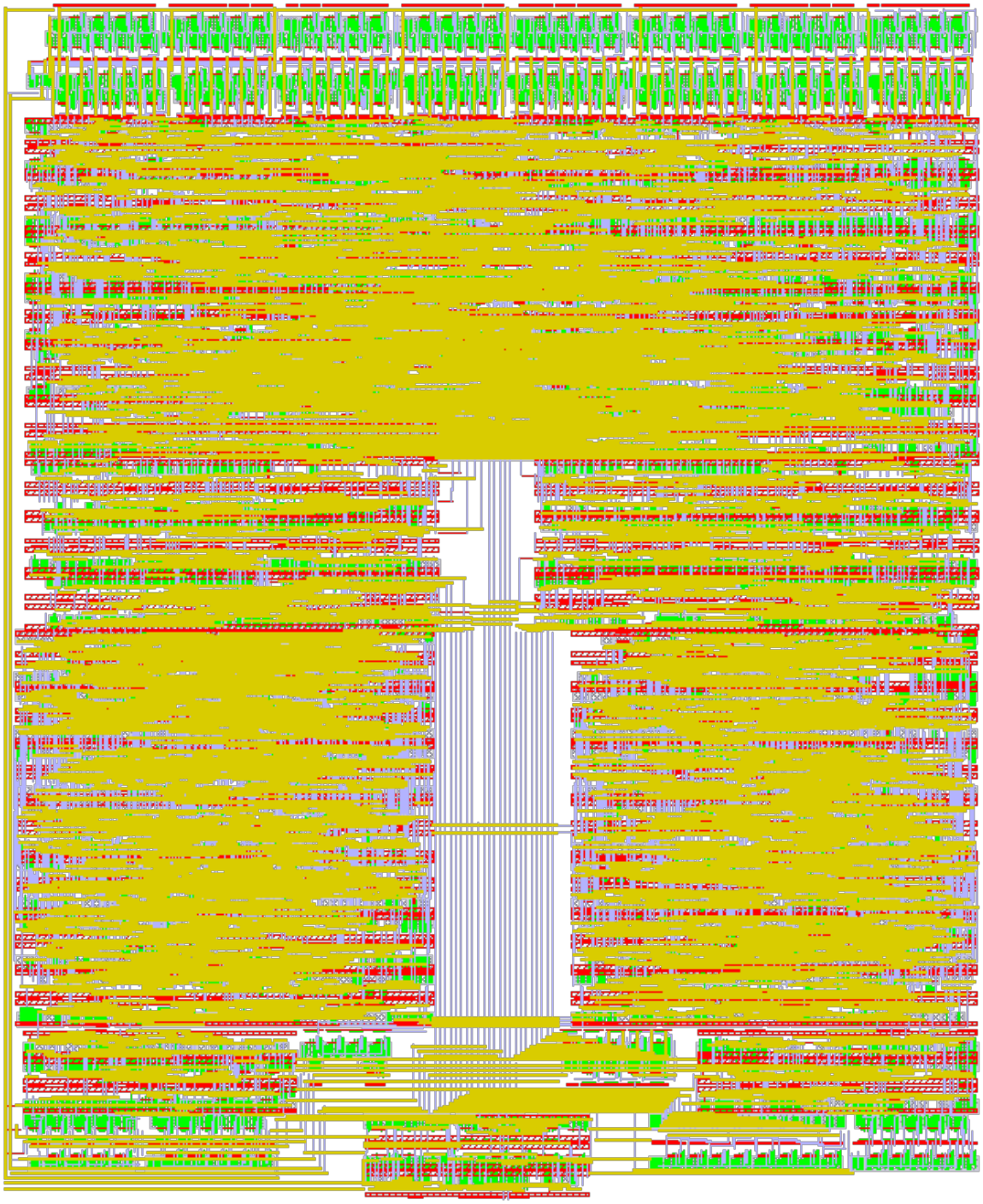


Figura 4.18: Diseño físico del *core* del ESCA ASIC.

4.4. Simulación

En la Figura 4.19 se muestra el esquema empleado para simular el diseño final del sistema, en el cual se emplea el símbolo creado a partir del diagrama esquemático que se muestra en la Figura 4.17. Para llevar acabo la simulación del diseño, se crean señales para estimular las entradas necesarias (*config*, *cs*, *fcn* y *sdi*), empleando fuentes de voltaje que pueden ser programadas de tal manera que sigan el comportamiento deseado. En el caso de la señal *sdi*, se lee un archivo que contiene los cambios de estado de la señal y los tiempos cuando ésta debe cambiar. Esta señal corresponde al dato serial que escribe en los registros se semillas y configuración necesaria para el funcionamiento del sistema de cifrado. Estos valores se muestran en la tabla 4.4 y son ingresados siguiendo el protocolo SPI en el modo 0, es decir, el primer valor a ingresar corresponde al bit más significativo del byte mientras que la lectura se realiza en la subida de reloj, el corrimiento de datos se realiza en la bajada de reloj y el *idle* de reloj es en estado 0.

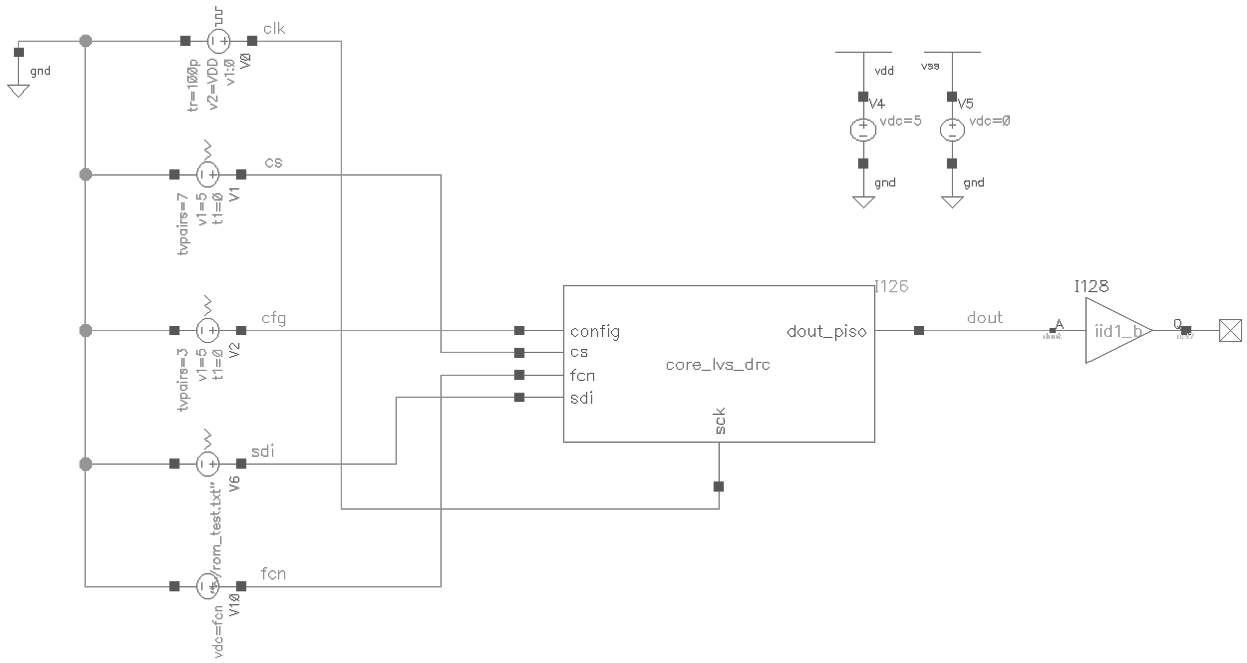


Figura 4.19: Diagrama esquemático para pruebas del sistema.

Se realizó una simulación ingresando los valores que se muestran en la tabla 4.4. Estos valores corresponden a un conjunto de semillas escogidas al azar, configurando el sistema para usar todos los módulos y el modo de operación de 63 para el PNRG y permutaciones.

En la siguiente lista se muestran los valores de la tabla 4.4 separados por semilla:

- Byte de configuración: 00011110

Bytes de entrada	Byte (decimal)	Función
00011110	30	Bits de configuración
00001010	10	4 bits (MSB) reservados y 4 bits (LSB) de Semilla Y
01010000	80	Semilla Y
00111000	56	Semilla Y
01111011	123	Semilla Y
00110101	53	Semilla Y
00011000	24	Semilla Y
01011100	92	Semilla Y
11011110	222	Semilla Y
11011100	220	4 bits (LSB) de Semilla Y y 4 bits (MSB) de Semilla X
01110101	117	Semilla X
01111010	122	Semilla X
10001001	137	Semilla X
10110001	177	Semilla X
00010010	18	Semilla X
01100000	96	Semilla X
10001000	136	Semilla X
00111100	60	3 bits (LSB) de X, 4 bits de CTE y 1 bit (MSB) de Z
10000111	135	8 bits de semilla Z
00000000	0	Primer dato a cifrar
00000010	2	Segundo dato a cifrar
00000011	3	Tercer dato a cifrar
00000100	4	Cuarto dato a cifrar
00000101	5	Quinto dato a cifrar

Tabla 4.4: Datos de configuración para cifrar datos.

- Semilla Y: 10100101000000111000011110110011010100011000010111001101111101101
- Semilla X: 11000111010101111010100010011011000100010010011000001000100000 1
- Semilla Z: 010000111
- Constante: 1110
- Primer dato de entrada: 00000000

En las Figuras 4.20 y 4.21 se muestran los diagramas de tiempo de las señales de entrada y salida del chip (**sck, fcn, cfg, cs, sdi** y **sdo**) y otras señales internas del circuito.

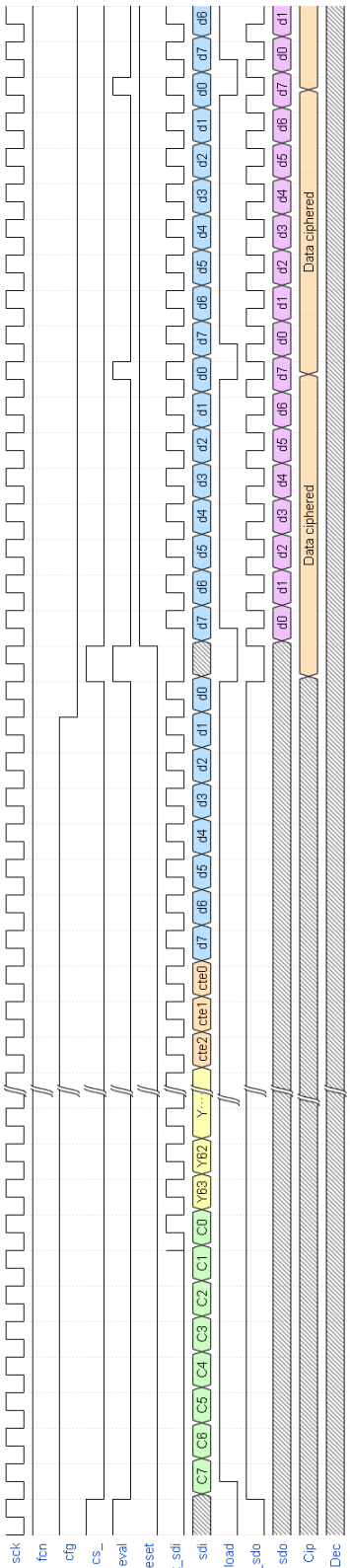


Figura 4.20: Diagramas de tiempo de operación de cifrado en el ESCA ASIC.

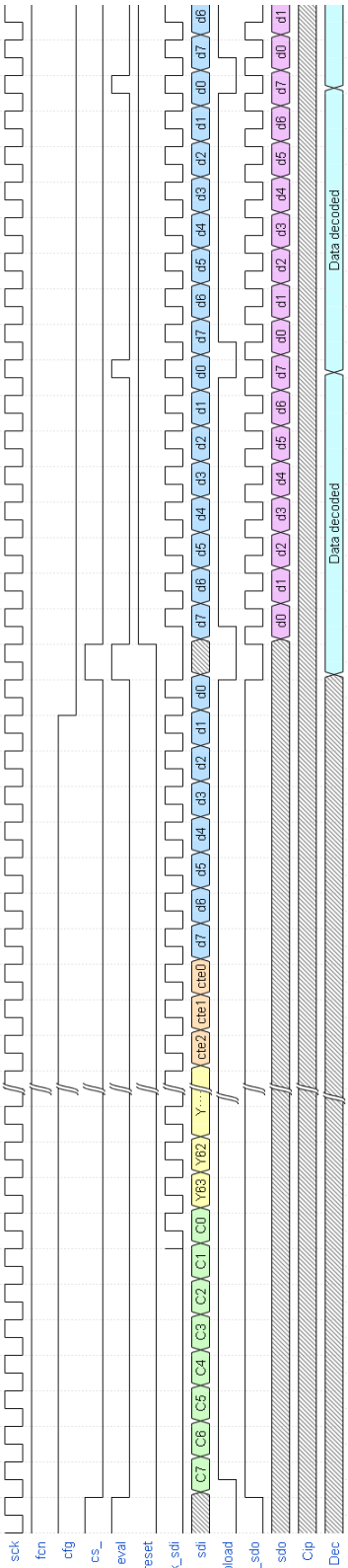


Figura 4.21: Diagramas de tiempo de operación de descifrado en el ESCA ASIC.

Dato de entrada	Cifrado	Dato de entrada	Descifrado
0	105	105	0
2	113	113	2
3	159	159	3
4	36	36	4
5	251	251	5

Tabla 4.5: Resultado de simulaciones de cifrado y descifrado.

Las señales que controlan el chip son las siguientes:

- **sck**: Señal de reloj.
- **fcn**: Señal para indicar el proceso de cifrado (1 lógico) o descifrado (0 lógico).
- **cfg**: Señal para indicar el modo de operación del chip, configuración (1) cifrado/descifrado (0).
- **cs**: Señal de *Chip Select* de protocolo SPI.
- **sdi**: Señal serial de entrada.

Para operar el chip, es necesario controlar las señales anteriormente enlistadas. La señal de reloj (**sck**), controlará la lectura y actualización de registros, así como la generación de otras señales internas. La señal **fcn**, como su nombre lo indica, es la encargada de indicar el proceso que se llevará a cabo (cifrar o descifrar) por lo que esta puede ser fijada desde el inicio del procedimiento de configuración. En este proceso también es necesario fijar la señal **cfg** en un estado 1 lógico, el cual indica que los registros de semillas, configuración y del dato de entrada, serán llenados con información valida, esta señal debe tomar el estado 0 lógico antes de que la señal **cs** tome un 1 lógico y que este en 0 lógico antes de que **cs** baje a 0 lógico. La señal **cs** indica si hay o no transmisión de datos a través del puerto **sdi**, que corresponde a los datos seriales de entrada.

Debido al costo computacional para realizar las simulaciones de todo el sistema, solo se ingresaron 5 valores para validar la funcionalidad del mismo. Los datos cifrados fueron ingresados al sistema (con la misma configuración y semillas) y los datos descifrados obtenidos corresponden a los iniciales, por lo que es posible asegurar que la funcionalidad de los 3 módulos, registros y resto del circuito, es correcta.

En la Figura 4.22 se muestra la simulación del sistema realizada en el software ADEL. Los resultados obtenidos en esta simulación muestran que el tiempo de propagación del *byte* cifrado, a partir de que la señal de evaluación (**eval**) es disparada, es de aproximadamente $10ns$. Este bus de datos es capturado en el registro PISO cuando la señal **eval** cambia de 1

a 0 lógico, por esto, la duración de la señal **eval** en alto debe ser mayor a dicho tiempo de propagación y dado a que la señal **eval** depende de la señal de reloj **sck**, esta señal de reloj también debe tener un periodo mayor a dos veces a $10ns$. De otro modo, los datos capturados en el registro PISO, podrían ser incorrectos. Debido a lo mencionado, se sugiere emplear una frecuencia de reloj no mayor a 33 Mhz, la cual contempla un periodo de $30ns$ que permita tener señales con un estado bien definido.

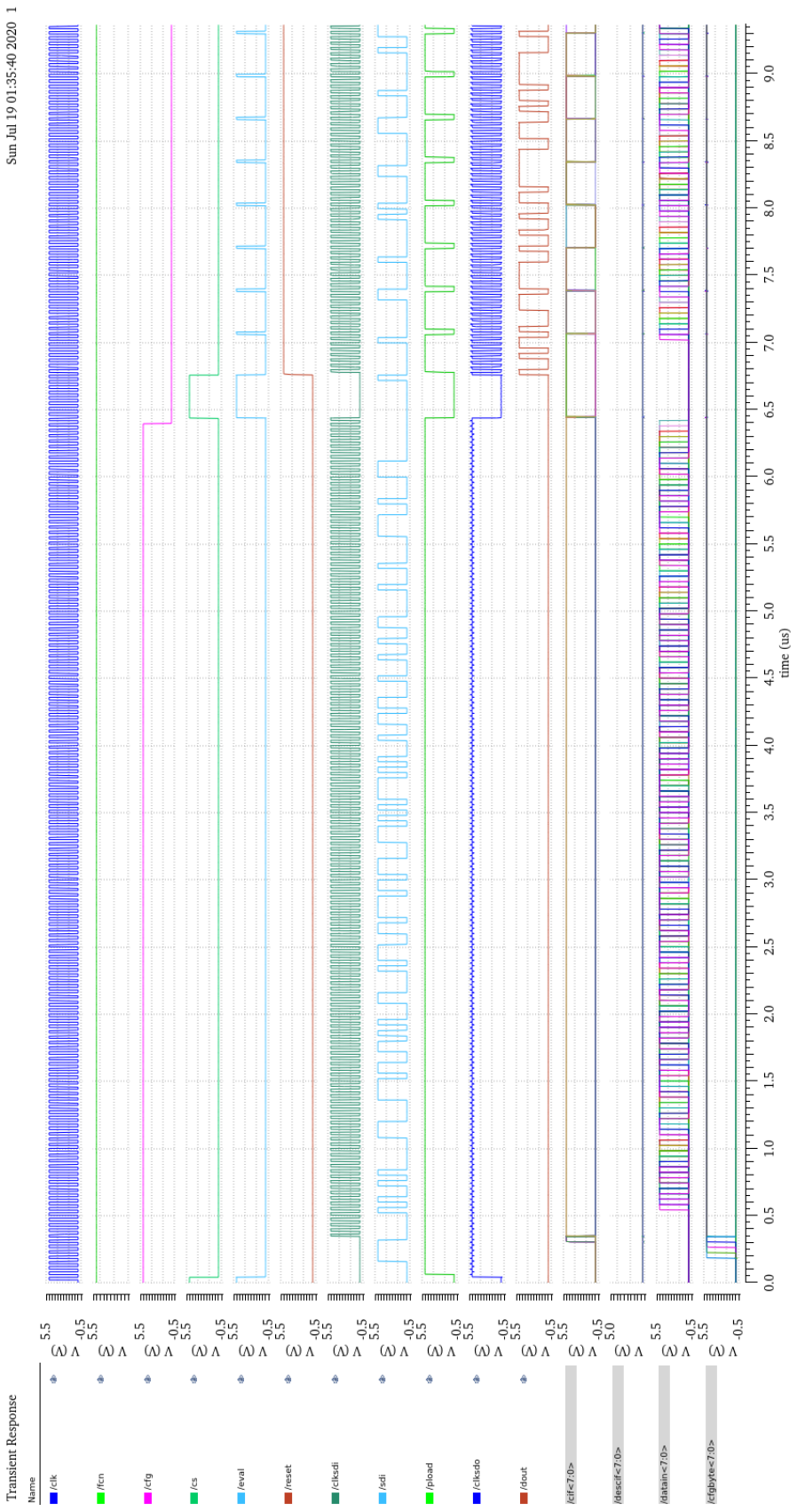


Figura 4.22: Diagramas de tiempo de operación de cifrado en el ESCA ASIC simulados en el software ADEL.

En la Figura 4.23 se muestra el consumo de energía resultante de la simulación del diseño empleando dos configuraciones distintas. En el modo 1, se configuró el sistema con todos los módulos y el tamaño de llave más grande (63 bits). En el modo de 2, se empleó únicamente el bloque Ψ con el tamaño de llave más pequeño (15 bits). En dicha figura se puede observar que el mayor gasto de energía es durante el proceso de configuración, donde todos los registros de las semillas y configuración están trabajando. Posterior a la configuración se puede observar que el consumo de energía del modo 2 es aproximadamente 37 % menor respecto al modo 1, debido a que la cantidad de dispositivos realizando operaciones o registros cambiando de estados, es menor.

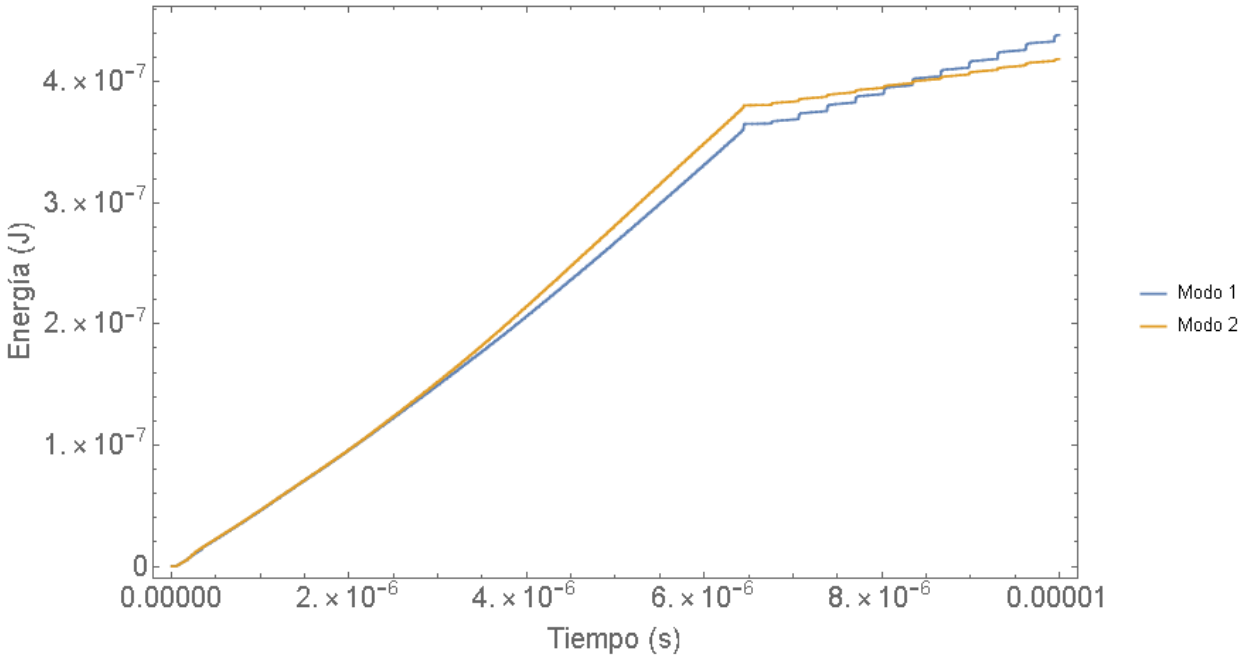


Figura 4.23: Consumo de energía del diseño físico.

En la Figura 4.24 se muestra el diseño físico final del ESCA ASIC, el cual se compone del *core* y un anillo de *pads* para entradas y salidas digitales, y *pads* para energizar el circuito (V_{DD} y GND).

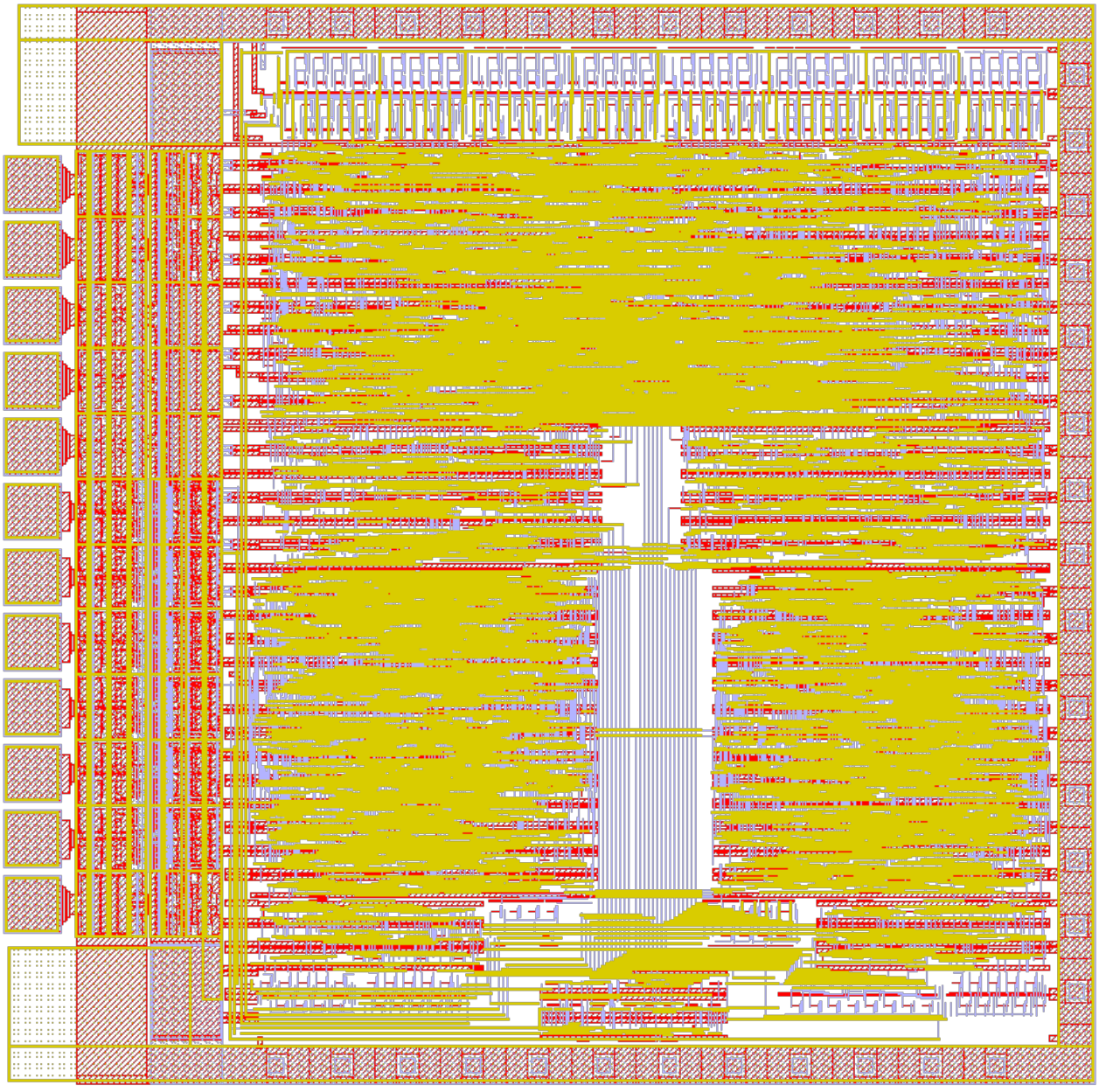


Figura 4.24: Diseño físico final del ESCA ASIC.

Capítulo 5

Chip multi-módulos

En este capítulo se presenta la implementación de un anillo oscilador de 19 etapas, un decodificador de 2 bits, un medio sumador de 2 bits implementado con compuertas NAND y otro usando con compuertas NOR. Estos circuitos fueron diseñados usando una tecnología de 5 μm y transistores nMOS. Debido a la limitación de dispositivos disponibles para fabricar, los diseños se basaron en un estilo de *pseudo-CMOS* que se describe en [28] y [29]. Este estilo de lógica emplea dos voltajes de suministro, V_{DD} y V_{SS}, este último siendo mas alto ($V_{DD} + 2V_{TH}$). Estos diseños hacen que los circuitos sean mas rápidos y el voltaje de pico a pico sea $\sim V_{DD}$, aunque esto implique mayor cantidad de área para su implementación.

El diseño de esquemas y *layouts*, se realizó empleando el software *Electric* [30] y la simulación de circuitos se llevo acabo en *LTspice* [31]. Las simulaciones pueden ser realizadas en dos modos distintos, conservador y solo tomando áreas y perímetros. El modo conservador realiza la extracción de parásitos, haciendo un circuito muy complejo y difícil de simular debido a las condiciones iniciales que debe encontrar en las capacitancias parasíticas, por lo que algunas simulaciones fueron realizadas tomando en cuenta únicamente áreas y perímetros para la extracciones de parásitos.

5.1. Anillo oscilador

Un anillo oscilador se compone de un número impar de inversores, donde la señal en cada nodo está cambiando entre el voltaje alto y bajo. Como se muestra en la Figura 5.1, el anillo se compone de 19 etapas (inversores).

Los símbolos fueron creados a partir del esquema que se muestra en la Figura 5.2a. El diseño implementado se propuso en [28], el cual consta de 4 transistores y se emplean dos voltajes de suministro. Cuando en el puerto **in** se coloca un voltaje bajo, los transistores

inferiores se apagan y el voltaje entre los transistores de la parte izquierda es aproximadamente $V_{SS} - V_{TH}$. El transistor que recibe este voltaje es encendido y la señal de salida es aproximadamente $V_{SS} - 2V_{TH}$. Por esto, VSS puede ser $(V_{DD} + 2V_{TH})$, para que el voltaje de salida sea lo mas alto posible. En la Figura 5.2b se muestra la implementación física del inversor. El área de color azul indica metal, el área rosa indica polisilicio, el área verde indica difusión y los pequeños recuadros indican una vía que conecta una capa con otra. El modelo de transistor, el tipo de tecnología y sus características, son desarrollados en el Centro de Ingeniería y Desarrollo Industrial (CIDESI). Las propiedades de los dispositivos y más información se muestran en [32].

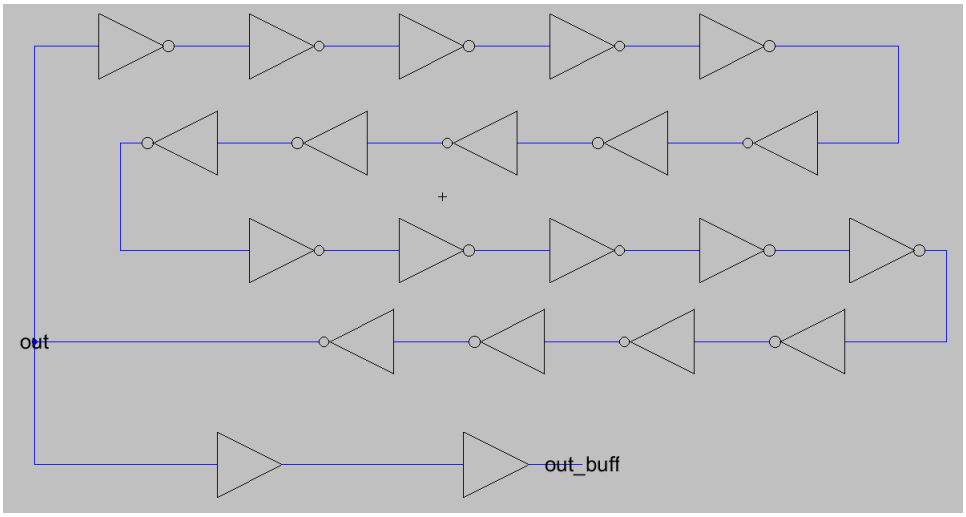


Figura 5.1: Diagrama esquemático de oscilador en forma de anillo de 19 etapas.

Los tamaños de los transistores se seleccionaron en base a simulaciones de Monte Carlo. Validar el correcto funcionamiento de ASICs, requiere evaluar los efectos de las tolerancias del proceso de fabricación mediante un análisis estadístico empleando el método de Monte Carlo. Éste consiste en realizar múltiples simulaciones eléctricas del circuito, variando de manera aleatoria aquellos parámetros qué se ven afectados por la fabricación. Una vez realizadas estas simulaciones, se observó para que tamaños se obtenía el mejor rendimiento o presentaba una menor probabilidad de fallar en caso de que hubiera variaciones en la fabricación, contra el tamaño de área necesaria para los dispositivos.

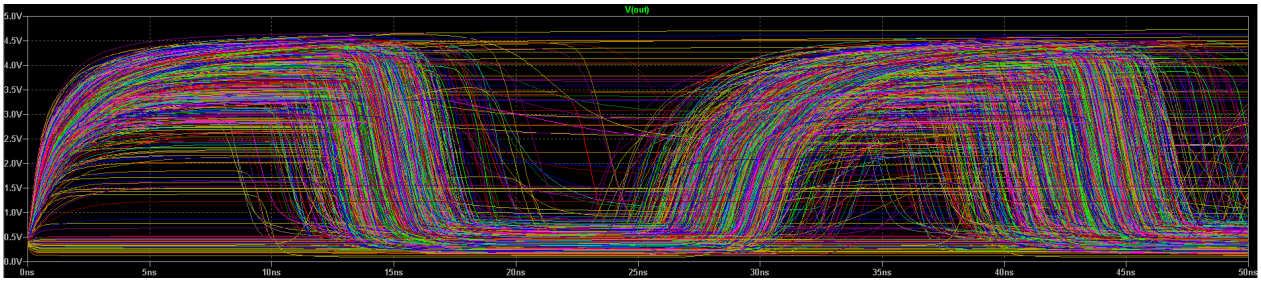
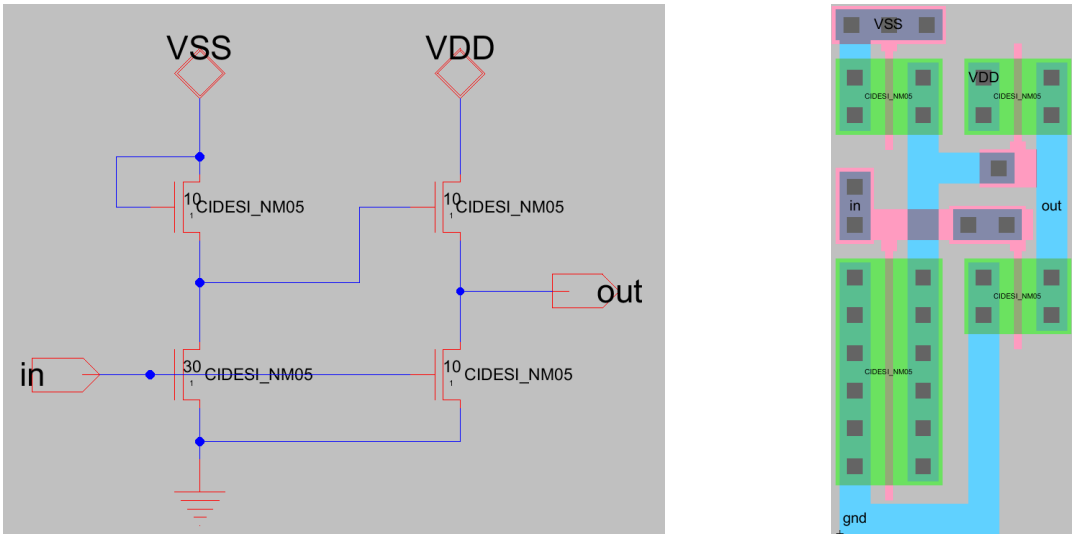


Figura 5.3: Simulación de Montecarlo de oscilador en forma de anillo.



(a) (b)

Figura 5.2: (a) Diagrama esquemático y (b) layout de un inversor. Las unidades están parametrizadas con $1\lambda = 5\mu m$.

En adición a las 19 etapas del anillo oscilador, se añadieron dos *buffers*. Estos componentes servirán para visualizar la señal a través de otros dispositivos que presentan cargas capacitivas o tolerancias adicionales a las del chip. En la Figura 5.4a se muestra el diseño esquemático para los dos *buffers* (4 inversores). El tamaño de los inversores fue aumentando en base a su comportamiento en simulaciones, hasta llegar a un tamaño final de 150λ. En la Figura 5.4b se muestra el *layout* final de los *buffers*.

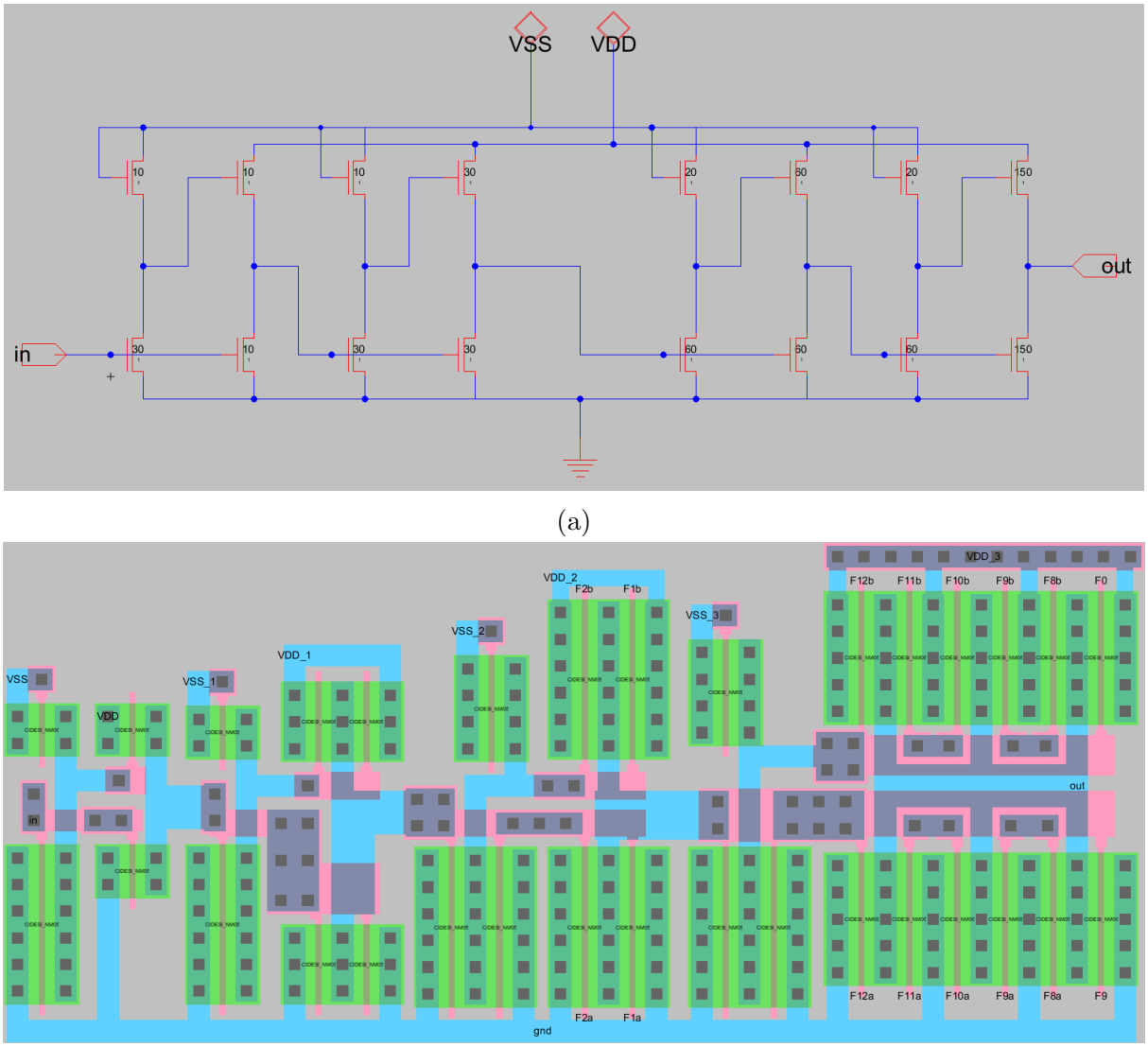


Figura 5.4: (a) Diagrama esquemático de un *buffer* de 2 etapas y (b) *Layout* de *buffer*.

En la Figura 5.5 se muestra el *layout* final para el chip de 4 mm^2, el cual consta del anillo oscilador de 19 etapas con los *buffers* de salida. Los recuadros en color azul (capa de metal) con puntos negros (capa de pasivación) se emplean como pads del chip. A través de estos es posible energizar el sistema y escribir o leer señales de el mismo.

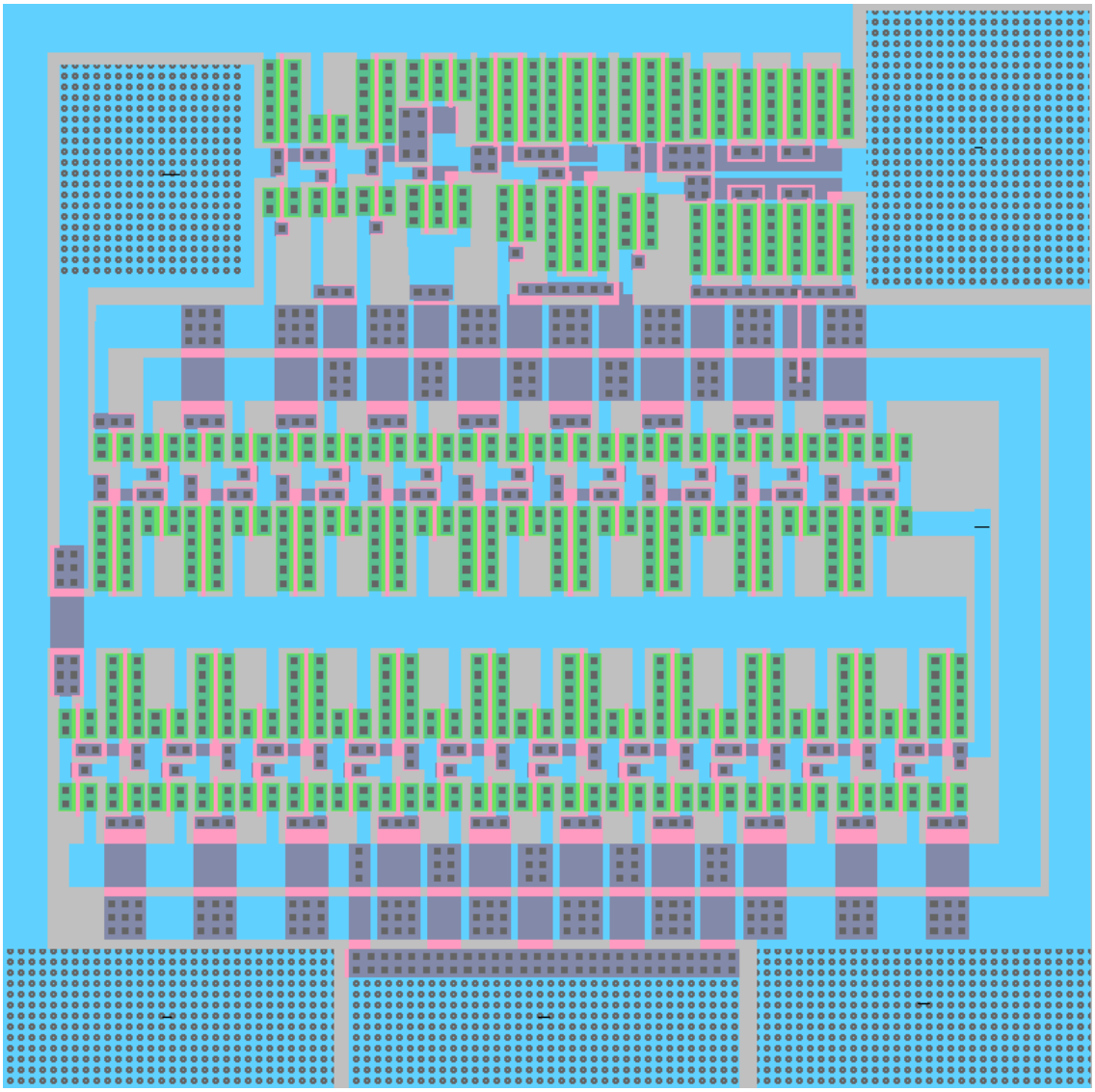
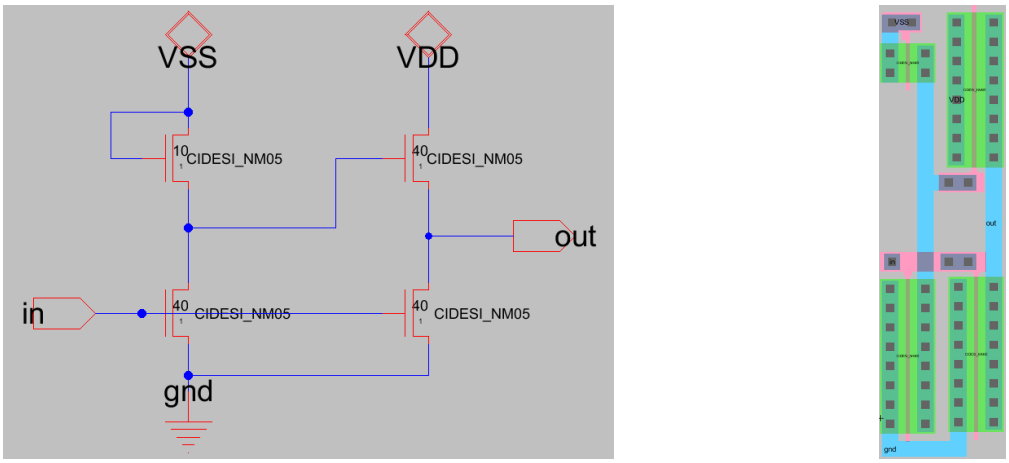


Figura 5.5: Diseño físico final del chip con un oscilador en forma de anillo de 19 etapas.

5.2. Celdas

Los diseños de los sumadores y el decodificador emplean compuertas NAND, NOR e INV, por lo que se elaboraron las 3 celdas para emplearlas en diseños mas grandes. La compuerta INV (inversor), posee una entrada y una salida, su propósito es generar una salida inversa a su entrada. Si la entrada se encuentra en 1 lógico, a la salida se genera un 0 lógico y para el caso contrario, si la entrada se encuentra en 0 lógico a la salida se genera un 1 lógico. En la Figura 5.6a se muestra el diagrama esquemático del inversor, implementado en base a un diseño de pseudo-CMOS. En la Figura 5.6b se muestra se muestra el *layout* del inversor, del

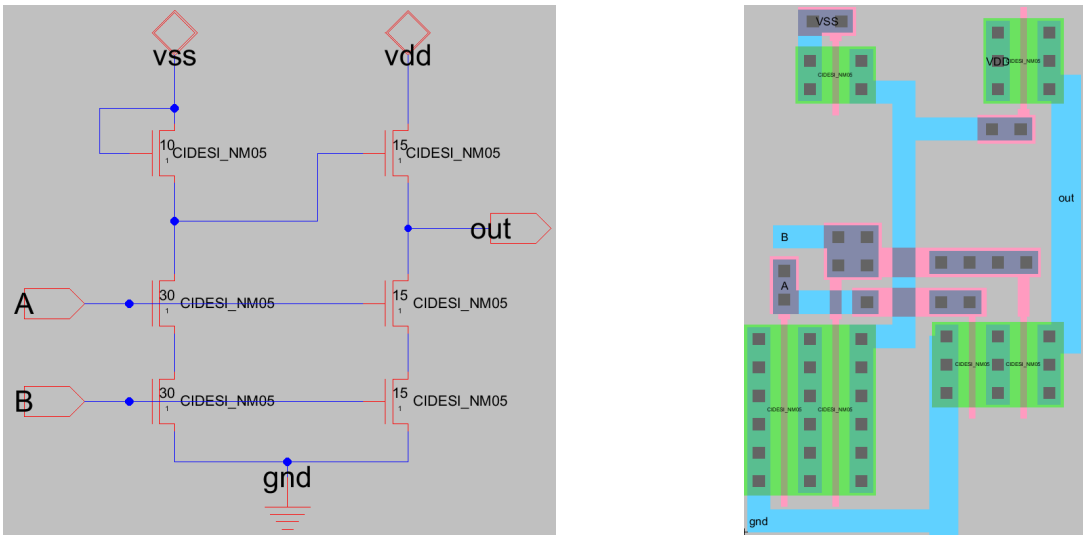
cual se puede observar que la nube *pull-up* está muy por encima de la nube *pull-down*, esto con el propósito de realizar interconexiones a través de la compuerta.



(a) (b)

Figura 5.6: (a) Diagrama esquemático y (b) layout de un inversor.

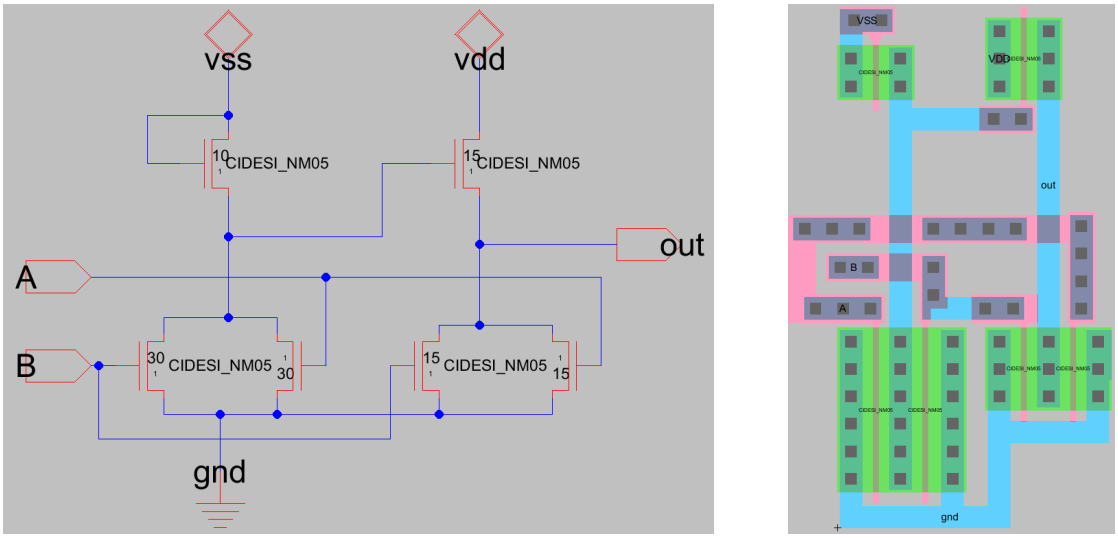
La compuerta NAND (también conocida como AND negada o inversa o NOT-AND) implementada tiene 2 entradas, al estar activas ambas, la salida se encuentra 0 lógico, cualquier otra variación con respecto a las entradas, la salida se mantiene en 1 lógico. En la Figura 5.7a se muestra el diagrama esquemático, diseñado con el estilo de pseudo-CMOS y en la Figura 5.7b se muestra el layout de la compuerta.



(a) (b)

Figura 5.7: (a) Diagrama esquemático y (b) layout de compuerta NAND de 2 entradas.

La compuerta NOR es una combinación de las compuertas OR y NOT (la versión inversa de la compuerta OR). Cuando sus entradas tienen un 0 lógico, su salida estará en un estado activo, pero si alguna de las entradas pasa a un estado activo, su salida tendrá un 0 lógico. La compuerta diseñada es de 2 entradas como se muestra en el diagrama esquemático (5.8a). En la Figura 5.8b se muestra el *layout* final de la compuerta que al igual que el resto de celdas, las nubes *pull-up* y *pull-down* están separadas para tener área disponible de interconexiones a través de las celdas.



(a) (b)

Figura 5.8: (a) Diagrama esquemático y (b) layout de compuerta NOR.

5.3. Decodificador

Un decodificador es un circuito combinacional que convierte un código binario de N bits de entrada y M bits de salida ($M = N^2$), de tal manera que cada bit tendrá un valor activo para cada posible combinación de los N bits de entrada. En la Figura 5.9 se muestra el diagrama esquemático de un decodificador de 2 a 4 bits que emplea únicamente compuertas NAND e INV, y en 5.1 se muestra la tabla de verdad para dicho circuito.

A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Tabla 5.1: Tabla de verdad de un decodificador de 2 bits.

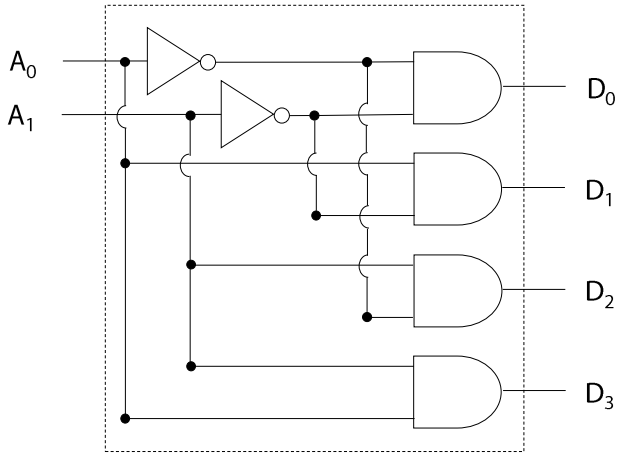


Figura 5.9: Diagrama esquemático de un decodificador de 2 bits.

En la Figura 5.10 se muestra el planeamiento de piso para el decodificador de 2 bits. Este diseño consta de dos filas, en la parte central se encuentra una sola de linea de GND para todas las compuertas empleadas. A los extremos de las dos filas se forman "anillos", cada uno para los dos suministros de voltaje y de este modo compactar el diseño. Las 4 salidas están conectadas a *buffers* para que en caso de que sea necesario, estos puedan suministrar la corriente suficiente que las salidas de las compuertas.

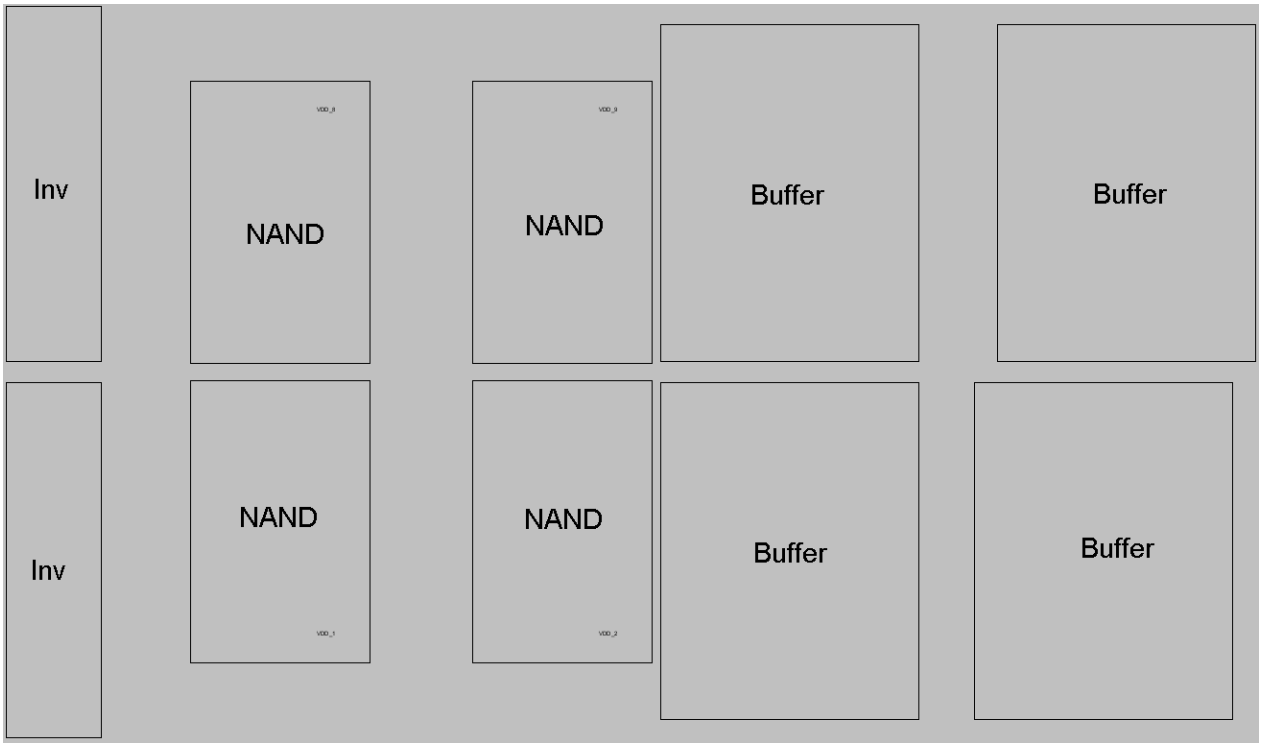


Figura 5.10: Planemiento de piso para un decodificador de 2 bits.

En la Figura 5.11 se muestra el *layout* final, de la cual se puede observar que el enrrutamiento de señales fue complicado debido a que el proceso solo cuenta con una capa de metal, debido a esto, en adicioón al metal, también se empleo la capa de polisilicio para el proceso de enrutamiento.

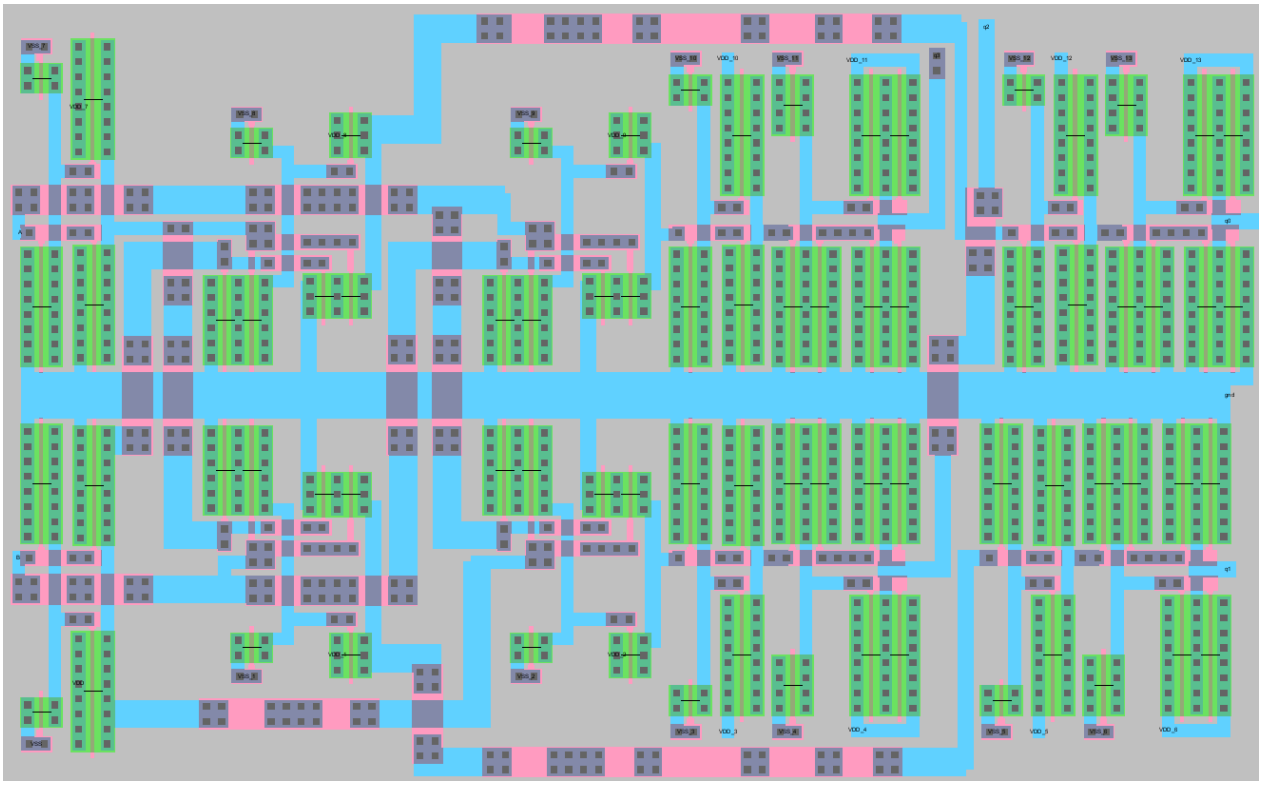


Figura 5.11: Diseño físico final para un decodificador de 2 bits.

La simulación del decodificador (Figura 5.12) muestra el correcto funcionamiento y se llevo acabo con un modo de simulación conservador para obtener la representación mas realista al circuito cuando este ya este fabricado. Las primeras dos señales (azul y roja) que se encuentran en la parte superior representan las señales de entrada. El resto de señales representan las salidas del decodificador.

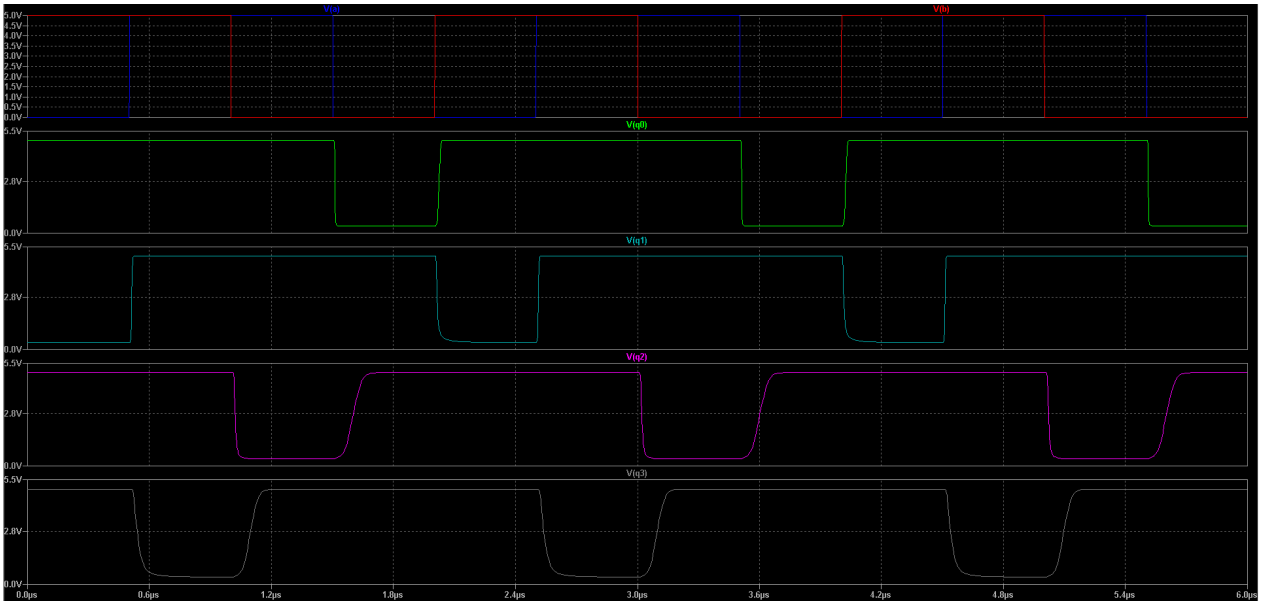


Figura 5.12: Simulación de un decodificador de 2 bits.

5.4. Medio sumador

Un medio sumador es un circuito capaz de sumar dos bits y producir un bit de acarreo (*carry*) de salida. Las combinaciones posibles se muestran en la siguiente tabla:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabla 5.2: Tabla de verdad de un medio sumador de 2 bits.

El medio sumador puede ser implementado empleando únicamente compuertas NAND (Figura 5.13a) o compuertas NOR (Figura 5.13b). Ambos diseños emplean 5 compuertas y es la cantidad mínima posible para realizar las implementaciones. La primera compuerta NAND toma las 2 entradas de 1 bit. El resultado de esa compuerta es la entrada de 3 compuertas junto con la entrada original. De estas 3 compuertas NAND, 2 generarán la salida que se dará como entrada a la compuerta NAND conectada al final. La compuerta conectada al final generará el bit de suma. De las 3 compuertas NAND consideradas, la tercera compuerta NAND generará el bit de *carry*. Para el diseño restante, se utilizan tres compuertas NOR al inicio y la salida de dos de estas compuertas NOR entran la cuarta compuerta. La salida de

la segunda compuerta NOR entra la compuerta conectada al final. Esto generará la suma de los 2 bits y de las 3 compuertas NAND consideradas, la tercera compuerta NAND generará el bit de *carry*.

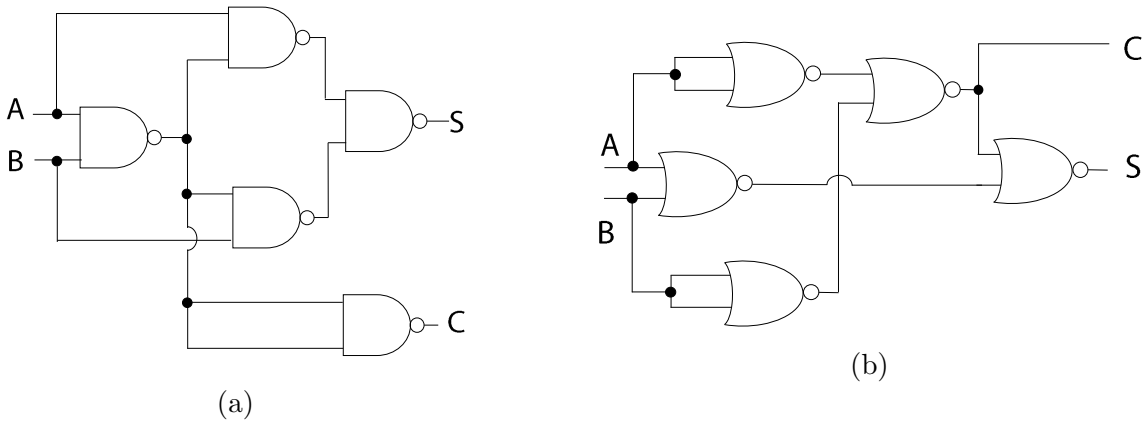


Figura 5.13: (a) Diagrama esquemático de un medio sumador de 2 bits empleando compuertas NAND. (b) Diagrama esquemático de un medio sumador de 2 bits empleando compuertas NOR.

El planeamiento de piso del medio sumador compuesto por compuertas NAND (Figura 5.14) se compone de dos filas, por el centro del diseño se encuentra una linea de GND y en la periferia las líneas de suministro de voltaje. Las salidas del sumador están conectadas a *buffers* al igual que los diseños anteriores. En la Figura 5.15 se muestra el diseño final del *layout*.

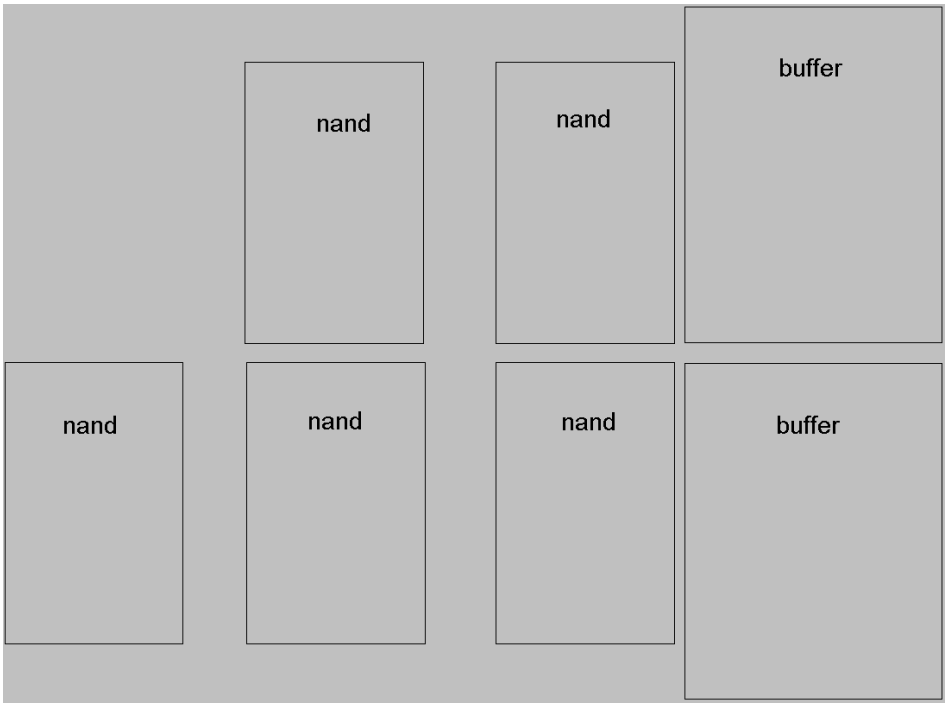


Figura 5.14: Planeamiento de piso para un medio sumador de 2 bits hecho con compuertas NAND.

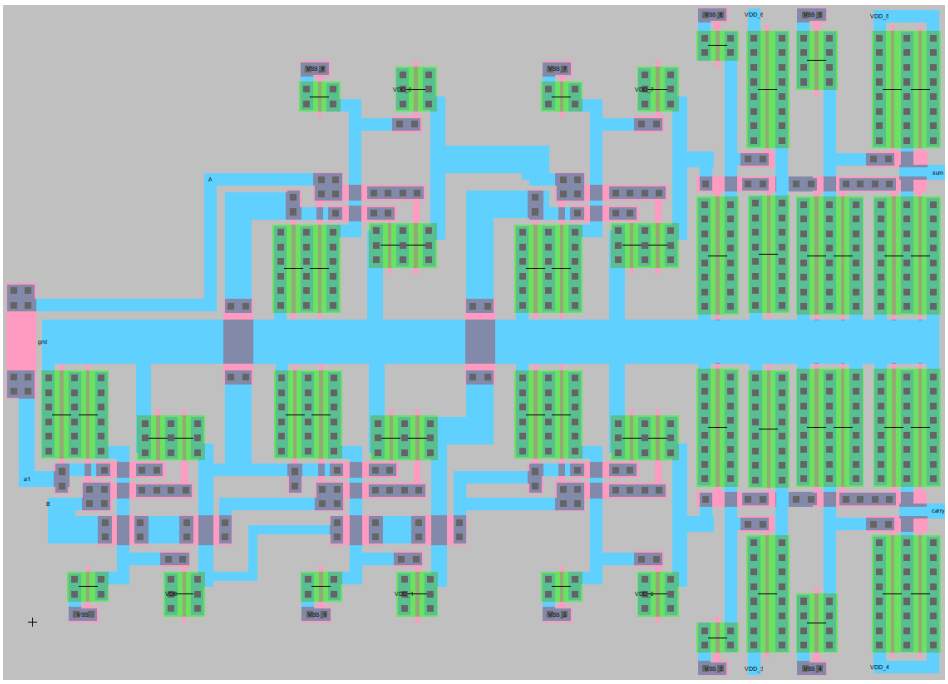


Figura 5.15: Diseño físico final de un medio sumador de 2 bits (NANDs).

Las simulaciones muestran un correcto funcionamiento del medio sumador. Las señales que se encuentran en la parte superior (ver Figura 5.16) pertenecen a las entradas, la señal central corresponde a la suma y la última señal pertenece al valor de *carry*.

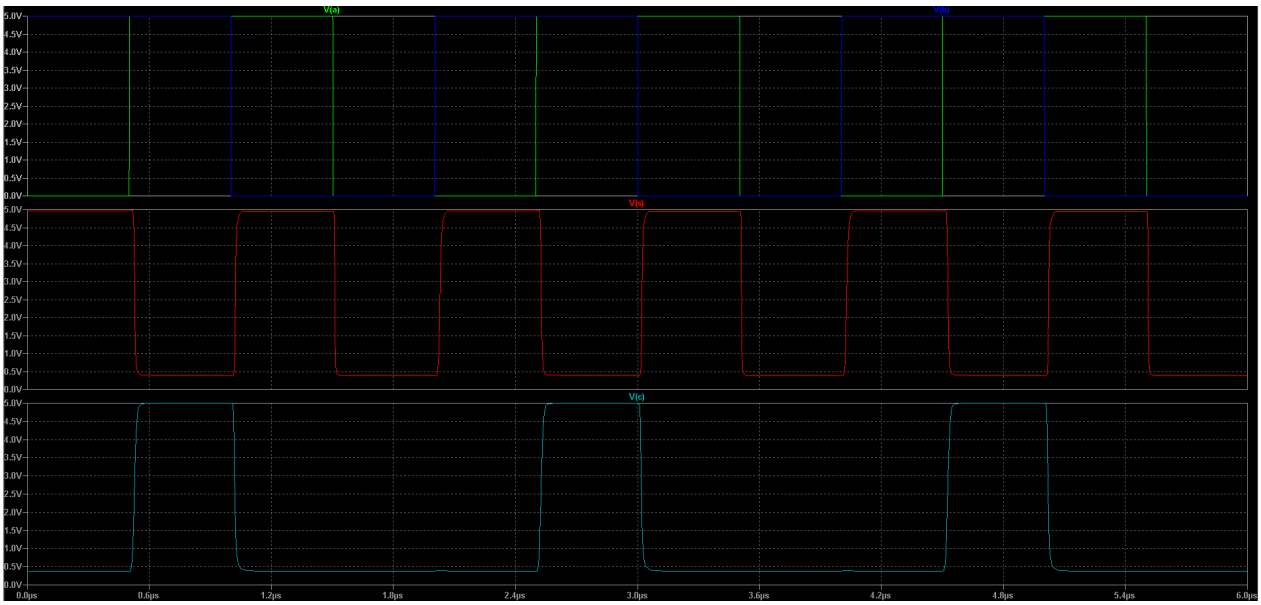


Figura 5.16: Simulación de un medio sumador de 2 bits (NANDs).

Al igual que los demás diseños, el planeamiento de piso para el medio sumador compuesto por NORs, se conforma de 2 filas, por la parte central se encuentra la conexión a GND y por la periferia las interconexiones para V_{DD} y V_{SS}. En la Figura 5.18 se muestra el *layout* final del medio sumador.

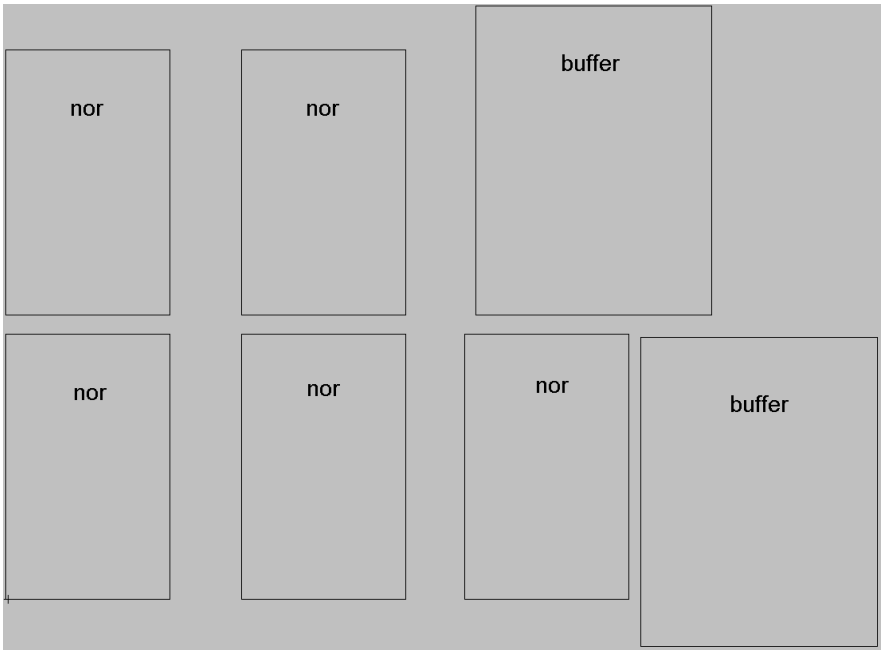


Figura 5.17: Planeamiento de piso para un medio sumador de 2 bits hecho con compuertas NOR.

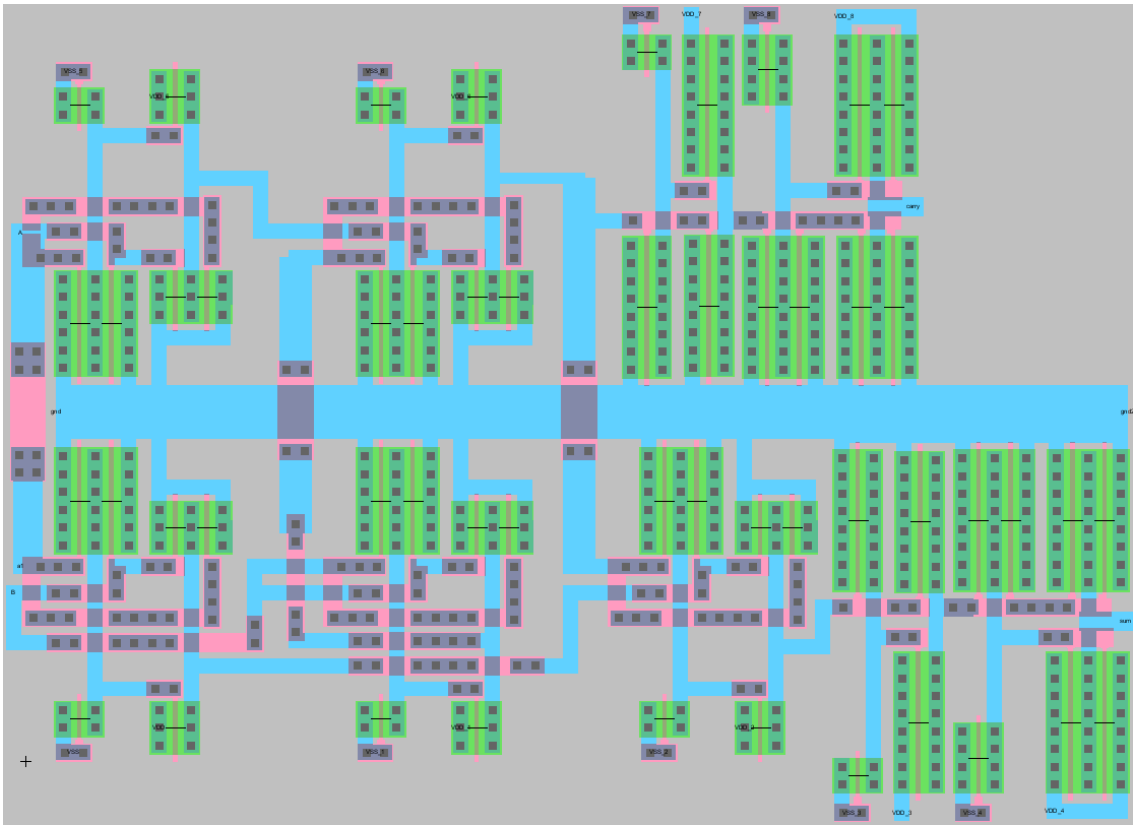


Figura 5.18: Diseño físico final de un medio sumador de 2 bits (NORs).

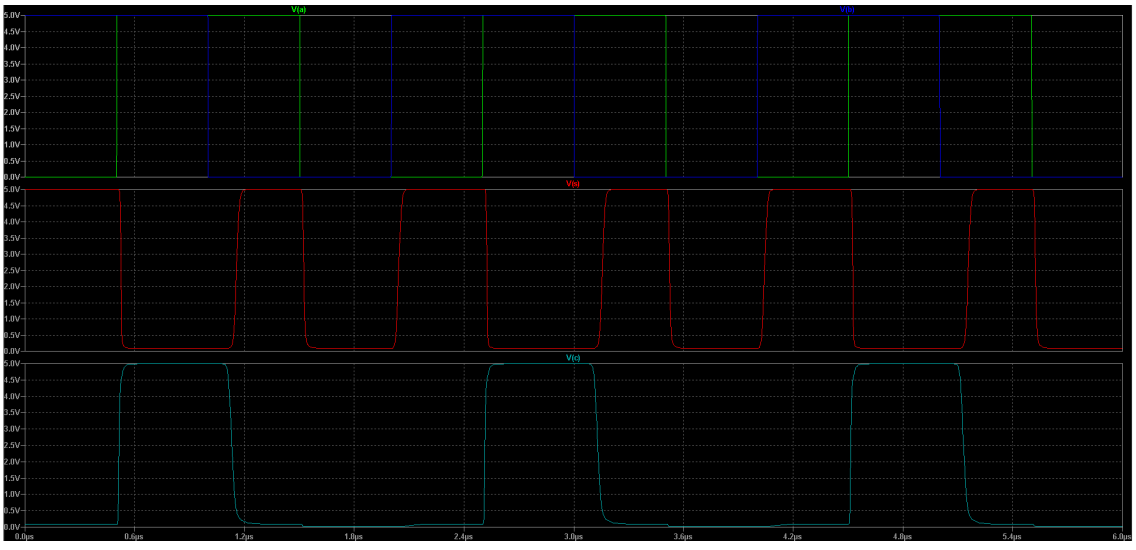


Figura 5.19: Simulación de un medio sumador de 2 bits (NORs).

Las simulaciones correspondientes a dicho diseño (Figura 5.19) muestran el correcto fun-

cionamiento del sumador y se puede apreciar que la señal correspondiente a la suma, tiene un mayor tiempo de propagación a diferencia de la implementación con compuertas NANDs, esto puede ser ocasionado a las capacitancias y resistencias parasiticas provenientes del enrutamiento con polisilicio en el diseño del sumador y de la celda NOR.

5.5. Chip

Los diseños anteriormente descritos: sumadores, decodificador y anillo oscilador, fueron colocados en un solo chip de 16 mm^2. En la Figura 5.20 se muestra el planeamiento de piso y en la Figura 5.21 se muestra el *layout* final del chip. Dicho diseño cuenta con 3 pares de *pads* para (V_{DD} y V_{SS}), donde el par para el anillo oscilador esta desconectado del resto. Los 2 pares restantes se emplean para energizar los sumadores y el decodificador. El pad de GND se comparte para todos los módulos. El pad **A1** y **B1** corresponden a las entradas de los 2 sumadores, la salidas del sumador implementado con compuertas NAND corresponden a **S** (suma) y **C** (*carry*) y las salidas del sumador implementado con compuertas NOR corresponden a **S1** (suma) y **C1** (*carry*). Los pads **A** y **B** corresponden a las entradas del decodificador y sus salidas corresponden a los pads **Q0**, **Q1**, **Q3** y **Q4**.

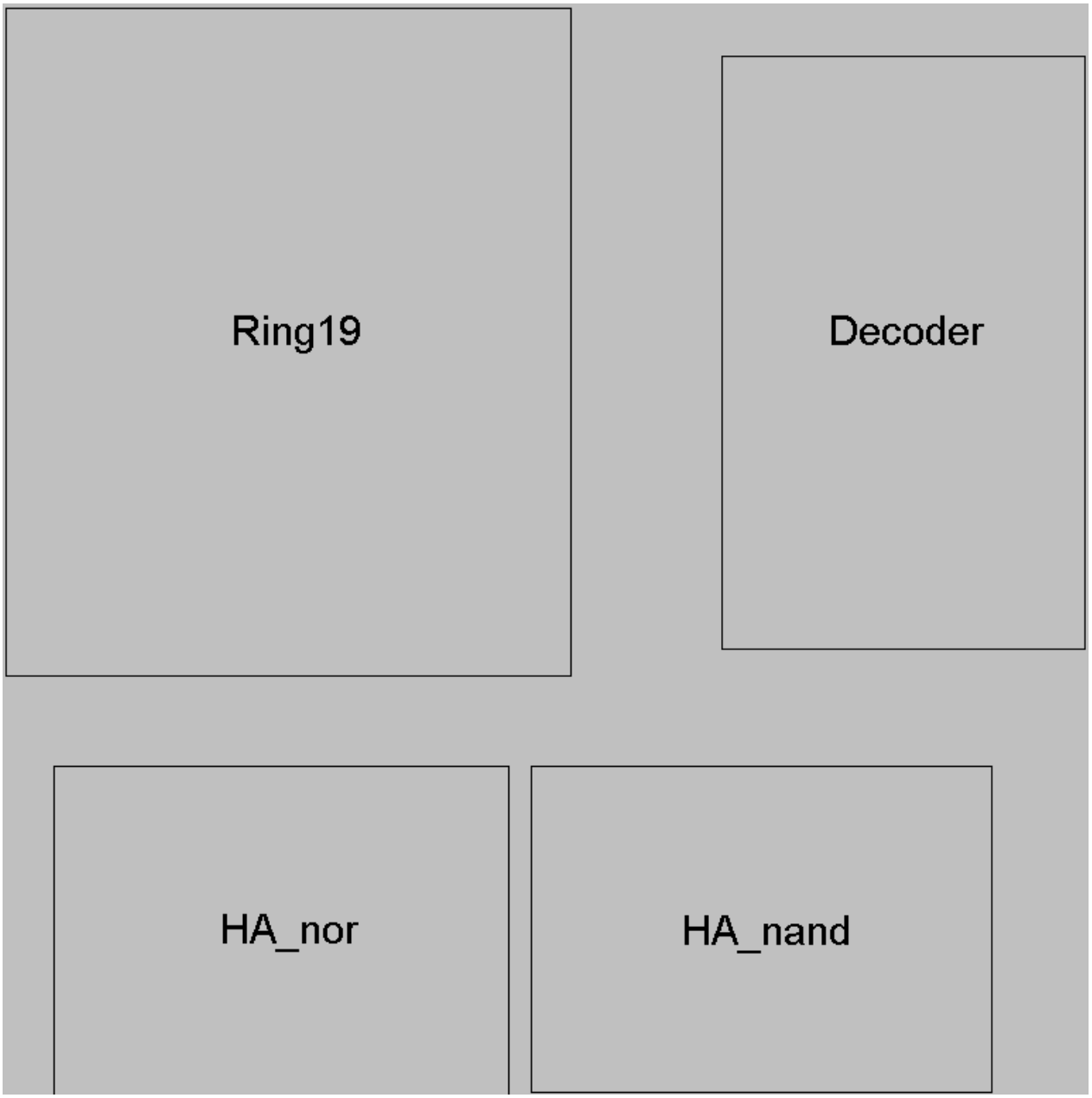


Figura 5.20: Planeamiento de piso de chip multi-módulos.

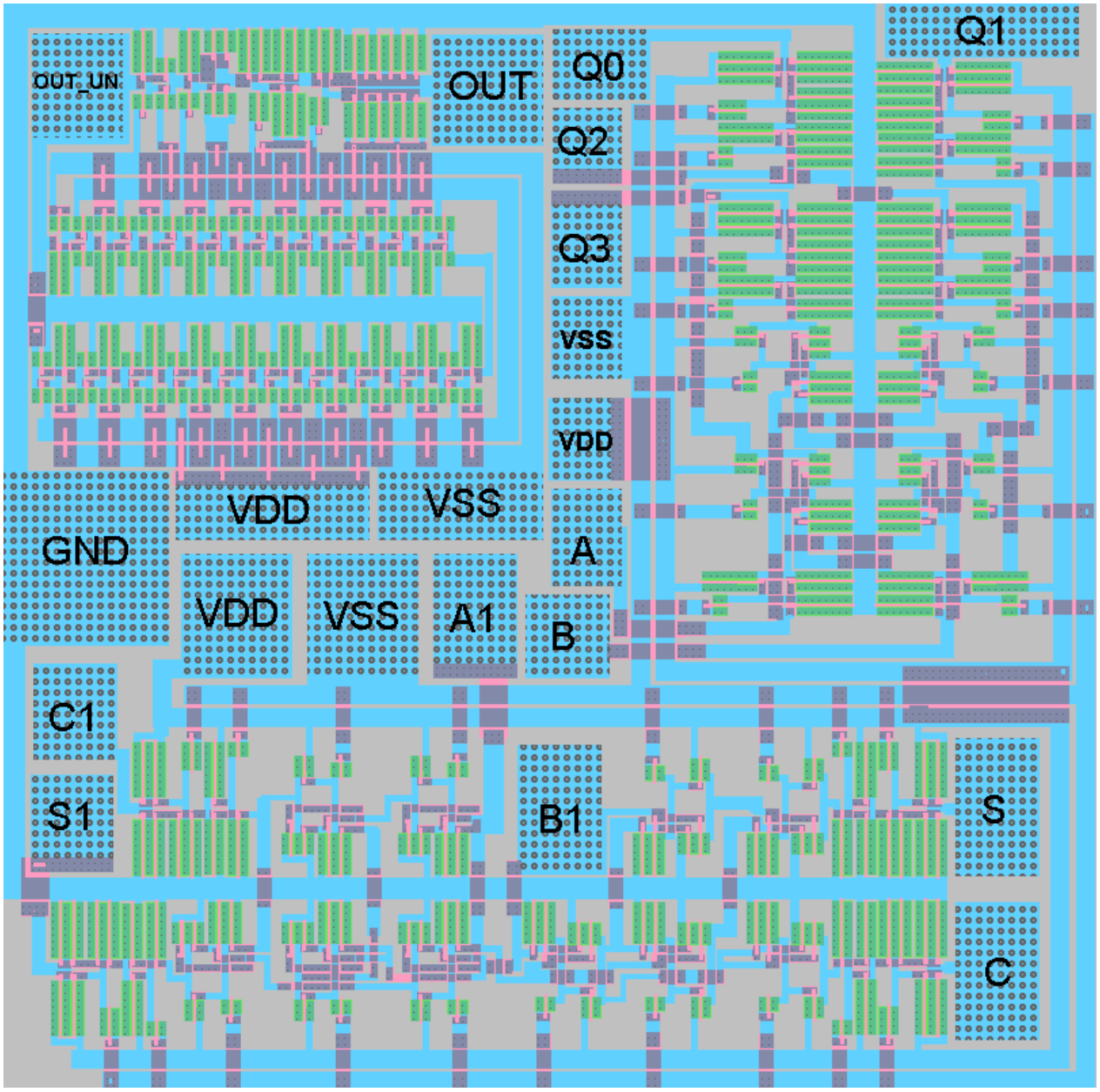


Figura 5.21: Diseño físico final de chip multi-módulos.

Capítulo 6

Conclusiones

En esta tesis se presentaron dos diseños y sus respectivas implementaciones en tecnología de Circuito Integrado de Aplicación Específica (ASIC). Uno corresponde al Sistema de Encriptación por Sincronización de Autómatas Celulares (ESCA) y el otro ASIC se compone de distintos módulos: un medio sumador de 2 bits implementado con compuertas NAND y otro con compuertas NOR, un decodificador de 2 bits y un oscilador en forma de anillo de 19 etapas. La motivación de realizar dichas implementaciones fue demostrar el desarrollo de ASICs en la Universidad Autónoma de San Luis Potosí, para la elaboración de instrumentación de bajo costo o instrumentos específicos necesarios en diferentes laboratorios en beneficio del Instituto de Investigación en Comunicación Óptica y de toda la universidad. Además de conocer las bases teóricas para el desarrollo de estos tipos de circuitos integrados, desde su etapa inicial hasta la implementación siguiendo el flujo de diseño VLSI.

Se describió el sistema de cifrado implementado, su bases matemáticas las cuales provienen de la sincronización de autómatas celulares, la unidad básica cifradora con la cual se generan las ecuaciones de un solo tiempo de la gran mayoría de módulos, la implementación de la propuesta presentada del generador de números pseudo-aleatorios modular y la descripción y estructura de los módulos de cifrado, pre-procesado, S-box, sus respectivos módulos inversos. Respecto a la tecnología CMOS, se presentaron las bases teóricas de la tecnología, partiendo desde una unión P-N hasta los modos de operación de los dispositivos CMOS. Se introdujo a la lógica digital y al flujo de diseño en tecnología VLSI, llevado de la mano de un ejemplo de la implementación de un sumador completo de 8 bits. Posteriormente, se describió el proceso de implementación del ESCA ASIC a través de una breve explicación en cada paso en el flujo de diseño VLSI además de las verificaciones (LVS y DRC) realizadas al diseño final y a cada módulo. Se mostraron los resultados de la simulación del diseño final, las especificaciones del dispositivo y el *set-up* para configurar el chip e introducir los valores iniciales necesarios en los registros que corresponden a las semillas y configuración. De igual manera, se describió el proceso de implementación de dos medios sumadores de 2 bits, implementados en compuertas NAND y NOR, un decodificador de 2 bits y un oscilador en forma de anillo de 19 etapas,

así como las simulaciones de cada módulo. Las celdas NAND, NOR e INV fueron diseños *full-custom* debido a que en este proceso aún no se ha desarrollado una librería estándar. Con las celdas verificadas funcionalmente, se desarrollaron los sumadores, el decodificador y el oscilador.

La implementación del ESCA ASIC se conformó de los módulos que corresponden al sistema de cifrado, registros para almacenar condiciones iniciales y configuración del chip, y una unidad de control. Todos los módulos a excepción de los registros SIPO y PISO, fueron descritos en Verilog y posteriormente sintetizados por *Design Compiler*, lo cual permitió que cualquier cambio funcional en el módulo, pudiera ser modificado desde el nivel más alto para sintetizarlo nuevamente y verificar si el problema fue resuelto o el comportamiento es el adecuado. Debido a la complejidad de dichos módulos, era inviable la realización *full-custom* de cada módulo, a diferencia de los registros SIPO y PISO que fueron creados de manera individual (diagrama esquemático y *layout*), colocando y conectando celda por celda, debido a su poca complejidad. Se genero el *layout* (planeamiento de piso y enrutamiento hechos automáticamente) de los diseños sintetizados empleando el software Encounter. Todos los diseños contaban con inconsistencias en las reglas de diseño, debido a que la librería estándar empleada no contaba con las últimas actualizaciones de las reglas, sin embargo, estos desajustes fueron solucionados manualmente y todos los bloques cumplen con dichas reglas. El proceso de planeamiento de piso del ESCA ASIC, se llevó a cabo de manera artesanal, teniendo como principal objetivo reducir la distancia de interconectado de los módulos, principalmente el bus de datos de 63 bits que corresponde al número pseudo-aleatorio. Se realizó la comprobación LVS entre el diagrama esquemático y el *layout* del chip, verificando que las interconexiones fueran correctas.

La simulación del *core*, se llevo acabo en el software *Analog Design Environment Layout* (ADEL), al igual que las simulaciones individuales de los módulos. Se observó que el tiempo de propagación de los datos a procesar es aproximadamente 10 ns siendo este un punto a mejorar en el siguiente diseño.

La implementación del ESCA ASIC confirma la factibilidad de la implementación del sistema de cifrado en tecnología ASIC, dejando como posibles líneas de investigación y trabajo a futuro la caracterización de los chips fabricados, así como la optimización del diseño actual. En la literatura se encuentran reportadas implementaciones de cajas de sustitución a través de los campos de galois (2(4)), con el propósito de disminuir el área de la implementación y la disminución de latencia. El proceso de síntesis para los bloques de cifrado, caja de sustitución, pre-procesado y PRNG se llevó a cabo sin ningún tipo de limitación de área o potencia, por lo que es factible explorar diseños de ultra bajo consumo energético e implementar técnicas para reducir el consumo de energía o diseños más pequeños para aplicaciones específicas como en dispositivos *wearables*, IoT o instrumentos científicos que requieran la protección de información. Para evaluar el costo energético y de tiempo, es posible implementar el sistema en un FPGA y comparar los resultados de ambas implementaciones y así validar las ventajas de la tecnología ASIC y de las implementaciones del sistema de cifrado en FPGA reportadas.

Los diseños implementados en el chip multi-proyectos tienen el propósito de caracterizar el proceso de fabricación e incursionar en el diseño de VLSI a través de la implementación de distintos diseños usando únicamente transistores nMOS, por lo que su evolución a partir de la caracterización de los dispositivos fabricados, puede tomar distintas líneas de investigación o aplicación, por ejemplo, la creación de celdas que trabajen en el régimen de *Sub-threshold* o diseños de alta velocidad. El trabajo consistió en la realización de celdas (diagrama esquemático y *layout*) de un inversor, una compuerta NAND de 2 entradas, una compuerta NOR de 2 entradas y *buffers* de 2 etapas. Estas celdas fueron empleadas para la elaboración de 4 módulos previamente mencionados (sumadores, decodificador y oscilador en forma de anillo). Todos los diseños cumplieron las verificaciones necesarias (LVS y DRC) y fueron simulados para comprobar su funcionalidad, obteniendo como resultado el correcto funcionamiento de los diseños. Los módulos y celdas pueden ser optimizadas y a su vez, la elaboración de más celdas para construir una librería estándar más completa de celdas tipo nMOS o la creación de IPs para diseños más grandes. Dos chips fueron enviados para su fabricación, el primero corresponde al oscilador en forma de anillo en un área de $4\,\mathrm{mm}^2$ y otro con el resto de diseños en un die de $16\,\mathrm{mm}^2$.

Debido a la poca infraestructura que hay en las instituciones públicas en México, el surgimiento de este tipo de procesos, beneficia a la comunidad que realiza este tipo de circuitos integrados en distintos aspectos como la disminución de costos de fabricación o el fomento y divulgación del trabajo que se realiza en esta tecnología, permitiendo el crecimiento de la comunidad y la captación de recursos.

Apéndice A

Algoritmos de un solo tiempo

A.1. Expresiones booleanas de un solo tiempo de la función $c = \psi_m(c)$ de tamaño 15

$c_1 = x_{15} \oplus x_{13} \oplus x_9 \oplus x_1 \oplus m_1$

$c_2 = x_{14} \oplus x_{13} \oplus x_2 \oplus m_2$

$c_3 = x_{14} \oplus x_{13} \oplus x_3 \oplus m_1 \oplus m_3$

$c_4 = x_{12} \oplus x_4 \oplus m_4$

$c_5 = x_{13} \oplus x_5 \oplus m_3 \oplus m_5$

$c_6 = x_{14} \oplus x_6 \oplus m_2 \oplus m_6$

$c_7 = x_{15} \oplus x_7 \oplus m_1 \oplus m_3 \oplus m_5 \oplus m_7$

$c_8 = x_8 \oplus m_8$

A.2. Expresiones booleanas de un solo tiempo de la función $m = \phi_x(m)$ de tamaño 15

$m_1 = x_{15} \oplus x_{13} \oplus x_9 \oplus x_1 \oplus c_1$

$m_2 = x_{14} \oplus x_{10} \oplus x_2 \oplus c_2$

$m_3 = x_{13} \oplus x_{11} \oplus x_9 \oplus x_3 \oplus x_1 \oplus c_1 \oplus c_3$

$m_4 = x_{12} \oplus x_4 \oplus c_4$

$m_5 = x_{11} \oplus x_9 \oplus x_5 \oplus x_3 \oplus x_1 \oplus c_1 \oplus c_3 \oplus c_5$

$m_6 = x_{10} \oplus x_6 \oplus x_2 \oplus c_2 \oplus c_6$

$m_7 = x_9 \oplus x_7 \oplus x_5 \oplus x_1 \oplus c_1 \oplus c_5 \oplus c_7$

$m_8 = x_8 \oplus c_8$

A.3. Expresiones booleanas de un solo tiempo de la función $c = \psi_m(c)$ de tamaño 31

$c_1 = x_{31} \oplus x_{29} \oplus x_{25} \oplus x_{17} \oplus x_1 \oplus m_1$

$c_2 = x_{30} \oplus x_{26} \oplus x_{18} \oplus x_2 \oplus m_2$

$c_3 = x_{31} \oplus x_{27} \oplus x_{19} \oplus x_3 \oplus m_1 \oplus m_3$

$c_4 = x_{28} \oplus x_{20} \oplus x_4 \oplus m_4$

$c_5 = x_{29} \oplus x_{21} \oplus x_5 \oplus m_3 \oplus m_5$

$c_6 = x_{29} \oplus x_{22} \oplus x_6 \oplus m_2 \oplus m_6$

$c_7 = x_{30} \oplus x_{22} \oplus x_6 \oplus m_1 \oplus m_3 \oplus m_5 \oplus m_7$

$c_8 = x_{24} \oplus x_8 \oplus m_8$

A.4. Expresiones booleanas de un solo tiempo de la función $m = \phi_x(m)$ de tamaño 31

$m_1 = x_1 \oplus x_{17} \oplus x_{25} \oplus x_{31} \oplus c_1$

$m_2 = x_2 \oplus x_{18} \oplus x_{26} \oplus x_{30} \oplus c_2$

$m_3 = x_1 \oplus x_3 \oplus x_{17} \oplus x_{19} \oplus x_{25} \oplus x_{27} \oplus x_{29} \oplus c_1 \oplus c_3$

$m_4 = x_4 \oplus x_{20} \oplus x_{28} \oplus c_4$

$m_5 = x_1 \oplus x_3 \oplus x_5 \oplus x_{17} \oplus x_{19} \oplus x_{21} \oplus x_{25} \oplus x_{27} \oplus c_1 \oplus c_3 \oplus c_5$

$m_6 = x_2 \oplus x_6 \oplus x_{18} \oplus x_{22} \oplus x_{26} \oplus c_2 \oplus c_6$

$m_7 = x_1 \oplus x_5 \oplus x_7 \oplus x_{17} \oplus x_{21} \oplus x_{23} \oplus x_{25} \oplus c_1 \oplus c_5 \oplus c_7$

$m_8 = x_8 \oplus x_{24} \oplus c_8$

A.5. Expresiones booleanas de un solo tiempo de la función $c = \psi_m(c)$ de tamaño 63

$c_1 = x_1 \oplus x_{33} \oplus x_{49} \oplus x_{57} \oplus x_{61} \oplus m_1$

$c_2 = x_2 \oplus x_{34} \oplus x_{50} \oplus x_{58} \oplus x_{62} \oplus m_2$

$c_3 = x_3 \oplus x_{35} \oplus x_{51} \oplus x_{59} \oplus x_{63} \oplus m_1 \oplus m_3$

$c_4 = x_4 \oplus x_{36} \oplus x_{52} \oplus x_{60} \oplus m_4$

$c_5 = x_5 \oplus x_{37} \oplus x_{53} \oplus x_{61} \oplus m_3 \oplus m_5$

$c_6 = x_6 \oplus x_{38} \oplus x_{54} \oplus x_{62} \oplus m_2 \oplus m_6$

$c_7 = x_7 \oplus x_{39} \oplus x_{55} \oplus x_{63} \oplus m_1 \oplus m_3 \oplus m_5 \oplus m_7$

$c_8 = x_7 \oplus x_{40} \oplus x_{56} \oplus m_8$

A.6. Expresiones booleanas de un solo tiempo de la función $m = \phi_x(m)$ de tamaño 63

$m_1 = x_1 \oplus x_{33} \oplus x_{49} \oplus x_{57} \oplus x_{61} \oplus x_{63} \oplus c_1$

$m_2 = x_2 \oplus x_{34} \oplus x_{50} \oplus x_{58} \oplus x_{62} \oplus c_2$

$m_3 = x_1 \oplus x_3 \oplus x_{33} \oplus x_{35} \oplus x_{49} \oplus x_{51} \oplus x_{57} \oplus x_{59} \oplus x_{61} \oplus c_1 \oplus c_3$

$m_4 = x_4 \oplus x_{36} \oplus x_{52} \oplus x_{60} \oplus c_4$

$m_5 = x_1 \oplus x_3 \oplus x_5 \oplus x_{33} \oplus x_{35} \oplus x_{37} \oplus x_{49} \oplus x_{51} \oplus x_{53} \oplus x_{57} \oplus x_{59} \oplus c_1 \oplus c_3 \oplus c_5$

$m_6 = x_2 \oplus x_6 \oplus x_{34} \oplus x_{38} \oplus x_{50} \oplus x_{54} \oplus x_{58} \oplus c_2 \oplus c_6$

$m_7 = x_1 \oplus x_5 \oplus x_7 \oplus x_{33} \oplus x_{37} \oplus x_{39} \oplus x_{49} \oplus x_{53} \oplus x_{55} \oplus x_{57} \oplus c_1 \oplus c_5 \oplus c_7$

$m_8 = x_8 \oplus x_{40} \oplus x_{56} \oplus c_8$

A.7. Ecuaciones de un solo tiempo de las permutaciones para el PRNG de 63 bits

$$t_0 = x_0 \oplus y_1$$

$$t_1 = x_1 \oplus y_0 \oplus y_2$$

$$t_2 = x_0 \oplus x_2 \oplus y_3$$

$$t_3 = x_3 \oplus y_0 \oplus y_2 \oplus y_4$$

$$t_4 = x_0 \oplus x_2 \oplus x_4 \oplus y_1 \oplus y_5$$

$$t_5 = x_1 \oplus x_5 \oplus y_0 \oplus y_4 \oplus y_6$$

$$t_6 = x_0 \oplus x_4 \oplus x_6 \oplus y_7$$

$$t_7 = x_7 \oplus y_0 \oplus y_4 \oplus y_6 \oplus y_8$$

$$t_8 = x_0 \oplus x_4 \oplus x_6 \oplus x_8 \oplus y_1 \oplus y_5 \oplus y_9$$

$$t_9 = x_1 \oplus x_5 \oplus x_9 \oplus y_0 \oplus y_2 \oplus y_4 \oplus y_8 \oplus y_{10}$$

$$t_{10} = x_0 \oplus x_2 \oplus x_4 \oplus x_8 \oplus x_{10} \oplus y_3 \oplus y_{11}$$

$$t_{11} = x_3 \oplus x_{11} \oplus y_0 \oplus y_2 \oplus y_8 \oplus y_{10} \oplus y_{12}$$

$$t_{12} = x_0 \oplus x_2 \oplus x_8 \oplus x_{10} \oplus x_{12} \oplus y_1 \oplus y_9 \oplus y_{13}$$

$$t_{13} = x_1 \oplus x_9 \oplus x_{13} \oplus y_0 \oplus y_8 \oplus y_{12} \oplus y_{14}$$

$$t_{14} = x_0 \oplus x_8 \oplus x_{12} \oplus x_{14} \oplus y_{15}$$

$$t_{15} = x_{15} \oplus y_0 \oplus y_8 \oplus y_{12} \oplus y_{14} \oplus y_{16}$$

$$t_{16} = x_0 \oplus x_8 \oplus x_{12} \oplus x_{14} \oplus x_{16} \oplus y_1 \oplus y_9 \oplus y_{13} \oplus y_{17}$$

$$t_{17} = x_1 \oplus x_9 \oplus x_{13} \oplus x_{17} \oplus y_0 \oplus y_2 \oplus y_8 \oplus y_{10} \oplus y_{12} \oplus y_{16} \oplus y_{18}$$

$$t_{18} = x_0 \oplus x_2 \oplus x_8 \oplus x_{10} \oplus x_{12} \oplus x_{16} \oplus x_{18} \oplus y_3 \oplus y_{11} \oplus y_{19}$$

$$t_{19} = x_3 \oplus x_{11} \oplus x_{19} \oplus y_0 \oplus y_2 \oplus y_4 \oplus y_8 \oplus y_{10} \oplus y_{16} \oplus y_{18} \oplus y_{20}$$

$$t_{20} = x_0 \oplus x_2 \oplus x_4 \oplus x_8 \oplus x_{10} \oplus x_{16} \oplus x_{18} \oplus x_{20} \oplus y_1 \oplus y_5 \oplus y_9 \oplus y_{17} \oplus y_{21}$$

$$t_{21} = x_1 \oplus x_5 \oplus x_9 \oplus x_{17} \oplus x_{21} \oplus y_0 \oplus y_4 \oplus y_6 \oplus y_8 \oplus y_{16} \oplus y_{20} \oplus y_{22}$$

$$t_{22} = x_0 \oplus x_4 \oplus x_6 \oplus x_8 \oplus x_{16} \oplus x_{20} \oplus x_{22} \oplus y_7 \oplus y_{23}$$

$$t_{23} = x_7 \oplus x_{23} \oplus y_0 \oplus y_4 \oplus y_6 \oplus y_{16} \oplus y_{20} \oplus y_{22} \oplus y_{24}$$

$$t_{24} = x_0 \oplus x_4 \oplus x_6 \oplus x_{16} \oplus x_{20} \oplus x_{22} \oplus x_{24} \oplus y_1 \oplus y_5 \oplus y_{17} \oplus y_{21} \oplus y_{25}$$

$$t_{25} = x_1 \oplus x_5 \oplus x_{17} \oplus x_{21} \oplus x_{25} \oplus y_0 \oplus y_2 \oplus y_4 \oplus y_{16} \oplus y_{18} \oplus y_{20} \oplus y_{24} \oplus y_{26}$$

$$t_{26} = x_0 \oplus x_2 \oplus x_4 \oplus x_{16} \oplus x_{18} \oplus x_{20} \oplus x_{24} \oplus x_{26} \oplus y_3 \oplus y_{19} \oplus y_{27}$$

$t_{27} = x_3 \oplus x_{19} \oplus x_{27} \oplus y_0 \oplus y_2 \oplus y_{16} \oplus y_{18} \oplus y_{24} \oplus y_{26} \oplus y_{28}$

$t_{28} = x_0 \oplus x_2 \oplus x_{16} \oplus x_{18} \oplus x_{24} \oplus x_{26} \oplus x_{28} \oplus y_1 \oplus y_{17} \oplus y_{25} \oplus y_{29}$

$t_{29} = x_1 \oplus x_{17} \oplus x_{25} \oplus x_{29} \oplus y_0 \oplus y_{16} \oplus y_{24} \oplus y_{28} \oplus y_{30}$

$t_{30} = x_0 \oplus x_{16} \oplus x_{24} \oplus x_{28} \oplus x_{30} \oplus y_{31}$

$t_{31} = x_{31} \oplus y_0 \oplus y_{16} \oplus y_{24} \oplus y_{28} \oplus y_{30} \oplus y_{32}$

$t_{32} = x_0 \oplus x_{16} \oplus x_{24} \oplus x_{28} \oplus x_{30} \oplus x_{32} \oplus y_1 \oplus y_{17} \oplus y_{25} \oplus y_{29} \oplus y_{33}$

$t_{33} = x_1 \oplus x_{17} \oplus x_{25} \oplus x_{29} \oplus x_{33} \oplus y_0 \oplus y_2 \oplus y_{16} \oplus y_{18} \oplus y_{24} \oplus y_{26} \oplus y_{28} \oplus y_{32} \oplus y_{34}$

$t_{34} = x_0 \oplus x_2 \oplus x_{16} \oplus x_{18} \oplus x_{24} \oplus x_{26} \oplus x_{28} \oplus x_{32} \oplus x_{34} \oplus y_3 \oplus y_{19} \oplus y_{27} \oplus y_{35}$

$t_{35} = x_3 \oplus x_{19} \oplus x_{27} \oplus x_{35} \oplus y_0 \oplus y_2 \oplus y_4 \oplus y_{16} \oplus y_{18} \oplus y_{20} \oplus y_{24} \oplus y_{26} \oplus y_{32} \oplus y_{34} \oplus y_{36}$

$t_{36} = x_0 \oplus x_2 \oplus x_4 \oplus x_{16} \oplus x_{18} \oplus x_{20} \oplus x_{24} \oplus x_{26} \oplus x_{32} \oplus x_{34} \oplus x_{36} \oplus y_1 \oplus y_5 \oplus y_{17} \oplus y_{21} \oplus y_{25} \oplus y_{33} \oplus y_{37}$

$t_{37} = x_1 \oplus x_5 \oplus x_{17} \oplus x_{21} \oplus x_{25} \oplus x_{33} \oplus x_{37} \oplus y_0 \oplus y_4 \oplus y_6 \oplus y_{16} \oplus y_{20} \oplus y_{22} \oplus y_{24} \oplus y_{32} \oplus y_{36} \oplus y_{38}$

$t_{38} = x_0 \oplus x_4 \oplus x_6 \oplus x_{16} \oplus x_{20} \oplus x_{22} \oplus x_{24} \oplus x_{32} \oplus x_{36} \oplus x_{38} \oplus y_7 \oplus y_{23} \oplus y_{39}$

$t_{39} = x_7 \oplus x_{23} \oplus x_{39} \oplus y_0 \oplus y_4 \oplus y_6 \oplus y_8 \oplus y_{16} \oplus y_{20} \oplus y_{22} \oplus y_{32} \oplus y_{36} \oplus y_{38} \oplus y_{40}$

$t_{40} = x_0 \oplus x_4 \oplus x_6 \oplus x_8 \oplus x_{16} \oplus x_{20} \oplus x_{22} \oplus x_{32} \oplus x_{36} \oplus x_{38} \oplus x_{40} \oplus y_1 \oplus y_5 \oplus y_9 \oplus y_{17} \oplus y_{21} \oplus y_{33} \oplus y_{37} \oplus y_{41}$

$t_{41} = x_1 \oplus x_5 \oplus x_9 \oplus x_{17} \oplus x_{21} \oplus x_{33} \oplus x_{37} \oplus x_{41} \oplus y_0 \oplus y_2 \oplus y_4 \oplus y_8 \oplus y_{10} \oplus y_{16} \oplus y_{18} \oplus y_{20} \oplus y_{32} \oplus y_{34} \oplus y_{36} \oplus y_{40} \oplus y_{42}$

$t_{42} = x_0 \oplus x_2 \oplus x_4 \oplus x_8 \oplus x_{10} \oplus x_{16} \oplus x_{18} \oplus x_{20} \oplus x_{32} \oplus x_{34} \oplus x_{36} \oplus x_{40} \oplus x_{42} \oplus y_3 \oplus y_{11} \oplus y_{19} \oplus y_{35} \oplus y_{43}$

$t_{43} = x_3 \oplus x_{11} \oplus x_{19} \oplus x_{35} \oplus x_{43} \oplus y_0 \oplus y_2 \oplus y_8 \oplus y_{10} \oplus y_{12} \oplus y_{16} \oplus y_{18} \oplus y_{32} \oplus y_{34} \oplus y_{40} \oplus y_{42} \oplus y_{44}$

$t_{44} = x_0 \oplus x_2 \oplus x_8 \oplus x_{10} \oplus x_{12} \oplus x_{16} \oplus x_{18} \oplus x_{32} \oplus x_{34} \oplus x_{40} \oplus x_{42} \oplus x_{44} \oplus y_1 \oplus y_9 \oplus y_{13} \oplus y_{17} \oplus y_{33} \oplus y_{41} \oplus y_{45}$

$t_{45} = x_1 \oplus x_9 \oplus x_{13} \oplus x_{17} \oplus x_{33} \oplus x_{41} \oplus x_{45} \oplus y_0 \oplus y_8 \oplus y_{12} \oplus y_{14} \oplus y_{16} \oplus y_{32} \oplus y_{40} \oplus y_{44} \oplus y_{46}$

$t_{46} = x_0 \oplus x_8 \oplus x_{12} \oplus x_{14} \oplus x_{16} \oplus x_{32} \oplus x_{40} \oplus x_{44} \oplus x_{46} \oplus y_{15} \oplus y_{47}$

$t_{47} = x_{15} \oplus x_{47} \oplus y_0 \oplus y_8 \oplus y_{12} \oplus y_{14} \oplus y_{32} \oplus y_{40} \oplus y_{44} \oplus y_{46} \oplus y_{48}$

$t_{48} = x_0 \oplus x_8 \oplus x_{12} \oplus x_{14} \oplus x_{32} \oplus x_{40} \oplus x_{44} \oplus x_{46} \oplus x_{48} \oplus y_1 \oplus y_9 \oplus y_{13} \oplus y_{33} \oplus y_{41} \oplus y_{45} \oplus y_{49}$

$t_{49} = x_1 \oplus x_9 \oplus x_{13} \oplus x_{33} \oplus x_{41} \oplus x_{45} \oplus x_{49} \oplus y_0 \oplus y_2 \oplus y_8 \oplus y_{10} \oplus y_{12} \oplus y_{32} \oplus y_{34} \oplus y_{40} \oplus y_{42} \oplus y_{44} \oplus y_{48} \oplus y_{50}$

$t_{50} = x_0 \oplus x_2 \oplus x_8 \oplus x_{10} \oplus x_{12} \oplus x_{32} \oplus x_{34} \oplus x_{40} \oplus x_{42} \oplus x_{44} \oplus x_{48} \oplus x_{50} \oplus y_3 \oplus y_{11} \oplus y_{35} \oplus y_{43} \oplus y_{51}$

$t_{51} = x_3 \oplus x_{11} \oplus x_{35} \oplus x_{43} \oplus x_{51} \oplus y_0 \oplus y_2 \oplus y_4 \oplus y_8 \oplus y_{10} \oplus y_{32} \oplus y_{34} \oplus y_{36} \oplus y_{40} \oplus y_{42} \oplus y_{48} \oplus y_{50} \oplus y_{52}$

$t_{52} = x_0 \oplus x_2 \oplus x_4 \oplus x_8 \oplus x_{10} \oplus x_{32} \oplus x_{34} \oplus x_{36} \oplus x_{40} \oplus x_{42} \oplus x_{48} \oplus x_{50} \oplus x_{52} \oplus y_1 \oplus y_5 \oplus y_9 \oplus y_{33} \oplus y_{37} \oplus y_{41} \oplus y_{49} \oplus y_{53}$

$t_{53} = x_1 \oplus x_5 \oplus x_9 \oplus x_{33} \oplus x_{37} \oplus x_{41} \oplus x_{49} \oplus x_{53} \oplus y_0 \oplus y_4 \oplus y_6 \oplus y_8 \oplus y_{32} \oplus y_{36} \oplus y_{38} \oplus y_{40} \oplus y_{48} \oplus y_{52} \oplus y_{54}$

$t_{54} = x_0 \oplus x_4 \oplus x_6 \oplus x_8 \oplus x_{32} \oplus x_{36} \oplus x_{38} \oplus x_{40} \oplus x_{48} \oplus x_{52} \oplus x_{54} \oplus y_7 \oplus y_{39} \oplus y_{55}$

$t_{55} = x_7 \oplus x_{39} \oplus x_{55} \oplus y_0 \oplus y_4 \oplus y_6 \oplus y_{32} \oplus y_{36} \oplus y_{38} \oplus y_{48} \oplus y_{52} \oplus y_{54} \oplus y_{56}$

$t_{56} = x_0 \oplus x_4 \oplus x_6 \oplus x_{32} \oplus x_{36} \oplus x_{38} \oplus x_{48} \oplus x_{52} \oplus x_{54} \oplus x_{56} \oplus y_1 \oplus y_5 \oplus y_{33} \oplus y_{37} \oplus y_{49} \oplus y_{53} \oplus y_{57}$

$t_{57} = x_1 \oplus x_5 \oplus x_{33} \oplus x_{37} \oplus x_{49} \oplus x_{53} \oplus x_{57} \oplus y_0 \oplus y_2 \oplus y_4 \oplus y_{32} \oplus y_{34} \oplus y_{36} \oplus y_{48} \oplus y_{50} \oplus y_{52} \oplus y_{56} \oplus y_{58}$

$t_{58} = x_0 \oplus x_2 \oplus x_4 \oplus x_{32} \oplus x_{34} \oplus x_{36} \oplus x_{48} \oplus x_{50} \oplus x_{52} \oplus x_{56} \oplus x_{58} \oplus y_3 \oplus y_{35} \oplus y_{51} \oplus y_{59}$

$t_{59} = x_3 \oplus x_{35} \oplus x_{51} \oplus x_{59} \oplus y_0 \oplus y_2 \oplus y_{32} \oplus y_{34} \oplus y_{48} \oplus y_{50} \oplus y_{56} \oplus y_{58} \oplus y_{60}$

$t_{60} = x_0 \oplus x_2 \oplus x_{32} \oplus x_{34} \oplus x_{48} \oplus x_{50} \oplus x_{56} \oplus x_{58} \oplus x_{60} \oplus y_1 \oplus y_{33} \oplus y_{49} \oplus y_{57} \oplus y_{61}$

$t_{61} = x_1 \oplus x_{33} \oplus x_{49} \oplus x_{57} \oplus x_{61} \oplus y_0 \oplus y_{32} \oplus y_{48} \oplus y_{56} \oplus y_{60} \oplus y_{62}$

$t_{62} = x_0 \oplus x_{32} \oplus x_{48} \oplus x_{56} \oplus x_{60} \oplus x_{62} \oplus y_{63}$

A.8. Ecuaciones de un pre-procesado

$m_1 = \hat{m}_1 \oplus z_2$

$m_2 = \hat{m}_2 \oplus z_1 \oplus z_3$

$m_3 = \hat{m}_1 \oplus \hat{m}_3 \oplus z_2 \oplus z_4$

$m_4 = \hat{m}_4 \oplus z_1 \oplus z_3 \oplus z_5$

$m_5 = \hat{m}_5 \oplus \hat{m}_3 \oplus z_2 \oplus z_4 \oplus z_6$

$m_6 = \hat{m}_6 \oplus \hat{m}_2 \oplus z_3 \oplus z_5 \oplus z_7$

$m_7 = \hat{m}_7 \oplus \hat{m}_5 \oplus \hat{m}_3 \oplus \hat{m}_1 \oplus z_4 \oplus z_6 \oplus z_8$

$m_8 = \hat{m}_8 \oplus z_1 \oplus z_5 \oplus z_7 \oplus z_9$

A.9. Ecuaciones de un pre-procesado inverso

$\hat{m}_1 = m_1 \oplus z_2$

$\hat{m}_2 = m_2 \oplus z_1 \oplus z_3$

$\hat{m}_3 = m_1 \oplus m_3 \oplus z_4$

$\hat{m}_4 = m_4 \oplus z_1 \oplus z_3 \oplus z_5$

$\hat{m}_5 = m_1 \oplus m_3 \oplus m_5 \oplus z_2 \oplus z_6$

$\hat{m}_6 = m_6 \oplus m_2 \oplus z_1 \oplus z_5 \oplus z_7$

$\hat{m}_7 = m_7 \oplus m_5 \oplus m_1 \oplus z_8$

$\hat{m}_8 = m_8 \oplus z_1 \oplus z_5 \oplus z_7 \oplus z_9$

A.10. S-box

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	00	E5	7E	AA	C2	6F	D4	25	1F	9D	BB	E8	16	D9	6C	0B
1	B4	85	F1	88	AD	6D	83	97	09	0E	EB	D7	8B	86	B0	63
2	5D	FC	35	D3	7A	0D	C8	56	A4	FA	B6	D6	78	41	7C	4B
3	BD	07	CD	AC	B3	80	26	3C	F8	89	05	17	1D	EE	7B	2A
4	E3	90	87	44	28	94	66	2F	82	27	FD	5B	92	76	53	43
5	19	DC	CA	72	50	C6	1B	84	8F	3D	67	8A	CF	57	EA	FF
6	E0	D5	70	0A	EF	C9	99	24	2D	65	48	14	5C	C0	95	2C
7	C7	4D	F5	A2	7D	3E	34	13	B9	31	CE	A3	BF	9A	9C	79
8	33	A6	0C	21	38	F6	1A	A5	91	59	4C	11	06	CB	E1	52
9	45	D2	61	E4	BA	10	AE	C1	01	FE	42	7F	2E	75	6A	1C
A	F9	D8	96	60	9F	5E	02	D0	1E	46	5F	47	F4	29	08	93
B	B8	39	A8	64	3B	4F	C5	37	E2	4A	6E	55	8E	8C	B7	4E
C	03	DB	2B	FB	0F	40	8D	E9	F3	C3	AF	58	B1	18	51	73
D	EC	32	36	E7	DA	6B	04	AB	DE	9B	12	BC	71	49	D1	DD
E	62	F0	A7	5A	3A	B5	54	BE	F2	30	98	E6	15	A1	74	F7
F	A0	20	68	C4	DF	23	69	3F	ED	77	81	22	CC	A9	B2	9E

Tabla A.1: Elementos de la S-box implementada en forma de una matriz de 16×16.

A.11. S-box inversa

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	00	98	A6	C0	D6	3A	8C	31	AE	18	63	0F	82	25	19	C4
1	95	8B	DA	77	6B	EC	0C	3B	CD	50	86	56	9F	3C	A8	08
2	F1	83	FB	F5	67	07	36	49	44	AD	3F	C2	6F	68	9C	47
3	E9	79	D1	80	76	22	D2	B7	84	B1	E4	B4	37	59	75	F7
4	C5	2D	9A	4F	43	90	A9	AB	6A	DD	B9	2F	8A	71	BF	B5
5	54	CE	8F	4E	E6	BB	27	5D	CB	89	E3	4B	6C	20	A5	AA
6	A3	92	E0	1F	B3	69	46	5A	F2	F6	9E	D5	0E	15	BA	05
7	62	DC	53	CF	EE	9D	4D	F9	2C	7F	24	3E	2E	74	02	9B
8	35	FA	48	16	57	11	1D	42	13	39	5B	1C	BD	C6	BC	58
9	41	88	4C	AF	45	6E	A2	17	EA	66	7D	D9	7E	09	FF	A4
A	F0	ED	73	7B	28	87	81	E2	B2	FD	03	D7	33	14	96	CA
B	1E	CC	FE	34	10	E5	2A	BE	B0	78	94	0A	DB	30	E7	7C
C	6D	97	04	C9	F3	B6	55	70	26	65	52	8D	FC	32	7A	5C
D	A7	DE	91	23	06	61	2B	1B	A1	0D	D4	C1	51	DF	D8	F4
E	60	8E	B8	40	93	01	EB	D3	0B	C7	5E	1A	D0	F8	3D	64
F	E1	12	E8	C8	AC	72	85	EF	38	A0	29	C3	21	4A	99	5F

Tabla A.2: Elementos de la S-box^{-1} implementada en forma de una matriz de 16×16.

Apéndice B

Descripciones HDL del sistema ESCA

B.1. Módulo Ψ

A continuación se muestra la descripción HDL del modulo Ψ, el cual se conforma de 3 bloques para cifrar datos con distintos tamaños de llave.

```
module cipher_module (prng, data_in, op_mode, select, fcn, data_out);

input [62:0] prng;
input [7:0] data_in;
input [1:0] op_mode;
input select, fcn;
output [7:0] data_out;

wire [7:0] data, data_64, data_32, data_16, cip64, cip32, cip16, cip;
wire [62:0] prng64;
wire [30:0] prng32;
wire [14:0] prng16;
wire [2:0] mode;
assign data = data_in & {8{(select)}};

assign data_64 = data & {8{(mode[2])}};
assign data_32 = data & {8{(mode[1])}};
assign data_16 = data & {8{(mode[0])}};
assign prng64 = {63{fcn}} & prng & {63{(mode[2])}};
assign prng32 = {31{fcn}} & prng[30:0] & {31{(mode[1])}};
assign prng16 = {15{fcn}} & prng[14:0] & {15{(mode[0])}};

cip_64  U1 (.x(prng64), .m(data_64), .c(cip64));
cip_32  U2 (.x(prng32), .m(data_32), .c(cip32));
cip_16  U3 (.x(prng16), .m(data_16), .c(cip16));

assign cip = cip64|cip32|cip16;
```

```
28 assign data_out = (cip&{8{select}}) | (data_in&{8{~select}});
29
30 decoder D1 (.A1(op_mode[1]), .A0(op_mode[0]), .X2(mode[2]), .X1(mode[1]),
       .X0(mode[0]));
31
32 endmodule
33
34 module cip_64 (
35   input wire [62:0] x,
36   input wire [7:0] m,
37   output wire [7:0] c
38   );
39
40 assign  c[0]=x[0]^x[32]^x[48]^x[56]^x[60]^x[62]^m[0];
41 assign  c[1]=x[1]^x[33]^x[49]^x[57]^x[61]^m[1];
42 assign  c[2]=x[2]^x[34]^x[50]^x[58]^x[62]^m[0]^m[2];
43 assign  c[3]=x[3]^x[35]^x[51]^x[59]^m[3];
44 assign  c[4]=x[4]^x[36]^x[52]^x[60]^m[2]^m[4];
45 assign  c[5]=x[5]^x[37]^x[53]^x[61]^m[1]^m[5];
46 assign  c[6]=x[6]^x[38]^x[54]^x[62]^m[0]^m[2]^m[4]^m[6];
47 assign  c[7]=x[7]^x[39]^x[55]^m[7];
48
49 endmodule
50
51 module cip_32 (
52   input wire [30:0] x,
53   input wire [7:0] m,
54   output wire [7:0] c
55   );
56
57 assign c[0] = x[0] ^ x[16] ^ x[24] ^ x[28] ^ x[30] ^ m[0];
58 assign c[1] = x[1] ^ x[17] ^ x[25] ^ x[29] ^ m[1];
59 assign c[2] = x[2] ^ x[18] ^ x[26] ^ x[30] ^ m[0] ^ m[2];
60 assign c[3] = x[3] ^ x[19] ^ x[27] ^ m[3];
61 assign c[4] = x[4] ^ x[20] ^ x[28] ^ m[2] ^ m[4];
62 assign c[5] = x[5] ^ x[21] ^ x[29] ^ m[1] ^ m[5];
63 assign c[6] = x[6] ^ x[22] ^ x[30] ^ m[0] ^ m[2] ^ m[4] ^ m[6];
64 assign c[7] = x[7] ^ x[23] ^ m[7];
65
66 endmodule
67
68 module cip_16 (
69   input wire [14:0] x,
70   input wire [7:0] m,
71   output wire [7:0] c
72   );
73
74 assign  c[0] = x[0]^x[8]^x[12]^x[14] ^ m[0] ;
75 assign  c[1] = x[1]^x[9]^x[13] ^ m[1] ;
76 assign  c[2] = x[2]^x[10]^x[14] ^ m[0]^m[2] ;
77 assign  c[3] = x[3]^x[11] ^ m[3] ;
78 assign  c[4] = x[4]^x[12] ^ m[2]^m[4] ;
```

```
assign  c[5] = x[5]^x[13] ^ m[1]^m[5] ;
assign  c[6] = x[6]^x[14] ^ m[0]^m[2]^m[4]^m[6] ;
assign  c[7] = x[7] ^ m[7] ;

endmodule
```

B.2. Módulo Φ

A continuación se muestra la descripción HDL del modulo Φ, el cual se conforma de 3 bloques para descifrar datos con distintos tamaños de llave.

```
module decrypt_module (eval, prng, data_in, op_mode, select, fcn, data_out
    );

input eval;
input [62:0] prng;
input [7:0] data_in;
input [1:0] op_mode;
input select,fcn;
output [7:0] data_out;

wire clk_eff;
wire [7:0] dec,data_64,data_32,data_16,dec64,dec32,dec16;
wire [62:0] prng64;
wire [30:0] prng32;
wire [14:0] prng16;
wire [7:0] data;
wire [2:0] mode;

assign clk_eff = eval & select & ~fcn;

assign data_64 = data & {8{mode[2]}} & {8{~fcn}};
assign data_32 = data & {8{mode[1]}} & {8{~fcn}};
assign data_16 = data & {8{mode[0]}} & {8{~fcn}};

assign prng64 = prng & {63{mode[2]}}  & {63{~fcn}};
assign prng32 = prng[30:0] & {31{mode[1]}} & {31{~fcn}};
assign prng16 = prng[14:0] & {15{mode[0]}} & {15{~fcn}};

dec_64  U17 (.x(prng64), .c(data_64), .m(dec64));
dec_32  U18 (.x(prng32), .c(data_32), .m(dec32));
dec_16  U19 (.x(prng16), .c(data_16), .m(dec16));

assign dec = dec64|dec32|dec16;
assign data_out = (dec&{8{select}}) | (data&{8{~select}});

d_latch I1 (.q(data[0]),.d(data_in[0]),.en(eval & ~fcn));
d_latch I2 (.q(data[1]),.d(data_in[1]),.en(eval & ~fcn));
```

```
d_latch I3 (.q(data[2]),.d(data_in[2]),.en(eval & ~fcn));
d_latch I4 (.q(data[3]),.d(data_in[3]),.en(eval & ~fcn));
d_latch I5 (.q(data[4]),.d(data_in[4]),.en(eval & ~fcn));
d_latch I6 (.q(data[5]),.d(data_in[5]),.en(eval & ~fcn));
d_latch I7 (.q(data[6]),.d(data_in[6]),.en(eval & ~fcn));
d_latch I8 (.q(data[7]),.d(data_in[7]),.en(eval & ~fcn));

decoder D1 (.A1(op_mode[1]), .A0(op_mode[0]), .X2(mode[2]), .X1(mode[1]),
   .X0(mode[0]));
endmodule

module dec_64 (
  input [62:0] x,
  input [7:0] c,
  output [7:0] m
  );

assign  m[0] = x[0]^x[32]^x[48]^x[56]^x[60]^x[62] ^ c[0] ;
assign  m[1] = x[1]^x[33]^x[49]^x[57]^x[61] ^ c[1] ;
assign  m[2] = x[0]^x[2]^x[32]^x[34]^x[48]^x[50]^x[56]^x[58]^x[60] ^ c[0]^
   c[2] ;
assign  m[3] = x[3]^x[35]^x[51]^x[59] ^ c[3] ;
assign  m[4] = x[0]^x[2]^x[4]^x[32]^x[34]^x[36]^x[48]^x[50]^x[52]^x[56]^x
   [58] ^ c[0]^c[2]^c[4] ;
assign  m[5] = x[1]^x[5]^x[33]^x[37]^x[49]^x[53]^x[57] ^ c[1]^c[5] ;
assign  m[6] = x[0]^x[4]^x[6]^x[32]^x[36]^x[38]^x[48]^x[52]^x[54]^x[56] ^
   c[0]^c[4]^c[6] ;
assign  m[7] = x[7]^x[39]^x[55] ^ c[7] ;

endmodule

module dec_32 (
  input [30:0] x,
  input [7:0] c,
  output [7:0] m
  );

assign m[0] = x[0] ^ x[16] ^ x[24] ^ x[28] ^ x[30]  ^ c[0];
assign m[1] = x[1] ^ x[17] ^ x[25] ^ x[29] ^ c[1];
assign m[2] = x[0] ^ x[2] ^ x[16] ^ x[18] ^ x[24] ^ x[26] ^ x[28] ^ c[0] ^
    c[2];
assign m[3] = x[3] ^ x[19] ^ x[27] ^ c[3];
assign m[4] = x[0] ^ x[2] ^ x[4] ^ x[16] ^ x[18] ^ x[20] ^ x[24] ^ x[26] ^
    c[0] ^ c[2] ^ c[4];
assign m[5] = x[1] ^ x[5] ^ x[17] ^ x[21] ^ x[25] ^ c[1] ^ c[5];
assign m[6] = x[0] ^ x[4] ^ x[6] ^ x[16] ^ x[20] ^ x[22] ^ x[24] ^ c[0] ^
   c[4] ^ c[6];
assign m[7] = x[7] ^ x[23] ^ c[7];

endmodule

module dec_16 (
```

```
82   input [14:0] x,
83   input [7:0] c,
84   output [7:0] m
85   );
86
87 assign  m[0] = x[0]^x[8]^x[12]^x[14] ^ c[0] ;
88 assign  m[1] = x[1]^x[9]^x[13] ^ c[1] ;
89 assign  m[2] = x[0]^x[2]^x[8]^x[10]^x[12] ^ c[0]^c[2] ;
90 assign  m[3] = x[3]^x[11] ^ c[3] ;
91 assign  m[4] = x[0]^x[2]^x[4]^x[8]^x[10] ^ c[0]^c[2]^c[4] ;
92 assign  m[5] = x[1]^x[5]^x[9] ^ c[1]^c[5] ;
93 assign  m[6] = x[0]^x[4]^x[6]^x[8] ^ c[0]^c[4]^c[6] ;
94 assign  m[7] = x[7] ^ c[7] ;
95
96 endmodule
```

B.3. Módulo PRNG

A continuación se muestra la descripción HDL del módulo PRNG, el cual se conforma de 3 bloques para la construcción de la llave de tamaños mas grande, y a su vez también son usados para elaborar las llaves de menor tamaño.

```
1  module rng64_hm (eval,reset,select,seed_x,seed_y,op_mode,t,t2);
2
3  input eval;
4  input reset;
5  input select;
6  input [62:0] seed_x;
7  input [63:0] seed_y;
8  input [1:0] op_mode;
9  output [62:0] t;
10 output [62:0] t2;
11
12 assign t2=t;
13
14 reg [62:0] x;
15 reg [63:0] y;
16 wire [14:0] a1;
17 wire [15:0] b1;
18 wire [30:0] a2;
19 wire [31:0] b2;
20 wire [62:0] a3;
21
22 wire [63:0] b3;
23 wire clk1,clk2,clk3;
24 wire [2:0] mode;
25
26 always @ (posedge clk1) begin
```

```
27    if (!reset) begin
28      x[14:0] <= seed_x[14:0];
29      y[15:0] <= seed_y[15:0];
30    end
31    else begin
32      x[14:0] <= t[14:0];
33
34      y[15] <= ((mode[2]|mode[1])&x[15]) | (mode[0]&y[0]);
35      y[14:0] <= x[14:0];
36
37    end
38  end
39
40  always @(posedge clk2) begin
41    if (!reset) begin
42      x[30:15] <= seed_x[30:15];
43      y[31:16]<= seed_y[31:16];
44    end
45    else begin
46      x[30:15] <= t[30:15];
47
48      y[31] <= (mode[2]&x[31]) | (mode[1]&y[0]);
49      y[30:16] <= x[30:16];
50    end
51  end
52
53  always @(posedge clk3) begin
54    if (!reset) begin
55      x[62:31] <= seed_x[62:31];
56      y[63:32] <= seed_y[63:32];
57    end
58    else begin
59      x[62:31] <= t[62:31];
60
61      y[63] <= y[0];
62      y[62:32] <= x[62:32];
63    end
64  end
65
66  assign clk1=(eval&(mode[0]|mode[1]|mode[2]))&select;
67  assign clk2=(eval&(mode[1]|mode[2]))&select;
68  assign clk3=(eval&(mode[2]))&select;
69
70  assign a1 = (x[14:0] & {15{(mode[0]|mode[1]|mode[2])}});
71  assign b1 = (y[15:0] & {16{(mode[0]|mode[1]|mode[2])}});
72  assign a2 = (x[30:0] & {31{mode[1]|mode[2]}});
73  assign b2 = (y[31:0] & {32{mode[1]|mode[2]}});
74  assign a3 = (x[62:0] & {63{mode[2]}});
75  assign b3 = (y[63:0] & {64{mode[2]}});
76
77  rng16 U1 (.x(a1), .y(b1), .t(t[14:0]));
78  rng32 U2 (.x(a2), .y(b2), .t(t[30:15]));
```

```
79 rng64 U3 (.x(a3), .y(b3), .t(t[62:31]));
80
81 decoder D1 (.A1(op_mode[1]), .A0(op_mode[0]), .X2(mode[2]), .X1(mode[1]),
       .X0(mode[0]));
82
83 endmodule
84
85 module rng16 (x,y,t);
86
87 input [14:0] x;
88 input [15:0] y;
89 output [14:0] t;
90
91 assign t[0]=x[0]^y[1];
92 assign t[1]=x[1]^y[0]^y[2];
93 assign t[2]=x[0]^x[2]^y[3];
94 assign t[3]=x[3]^y[0]^y[2]^y[4];
95 assign t[4]=x[0]^x[2]^x[4]^y[1]^y[5];
96 assign t[5]=x[1]^x[5]^y[0]^y[4]^y[6];
97 assign t[6]=x[0]^x[4]^x[6]^y[7];
98 assign t[7]=x[7]^y[0]^y[4]^y[6]^y[8];
99 assign t[8]=x[0]^x[4]^x[6]^x[8]^y[1]^y[5]^y[9];
100 assign t[9]=x[1]^x[5]^x[9]^y[0]^y[2]^y[4]^y[8]^y[10];
101 assign t[10]=x[0]^x[2]^x[4]^x[8]^x[10]^y[3]^y[11];
102 assign t[11]=x[3]^x[11]^y[0]^y[2]^y[8]^y[10]^y[12];
103 assign t[12]=x[0]^x[2]^x[8]^x[10]^x[12]^y[1]^y[9]^y[13];
104 assign t[13]=x[1]^x[9]^x[13]^y[0]^y[8]^y[12]^y[14];
105 assign t[14]=x[0]^x[8]^x[12]^x[14]^y[15];
106
107 endmodule
108
109 module rng32 (x,y,t);
110
111 input [30:0] x;
112 input [31:0] y;
113 output [30:15] t;
114
115 assign t[15]=x[15]^y[0]^y[8]^y[12]^y[14]^y[16];
116 assign t[16]=x[0]^x[8]^x[12]^x[14]^x[16]^y[1]^y[9]^y[13]^y[17];
117 assign t[17]=x[1]^x[9]^x[13]^x[17]^y[0]^y[2]^y[8]^y[10]^y[12]^y[16]^y[18];
118 assign t[18]=x[0]^x[2]^x[8]^x[10]^x[12]^x[16]^x[18]^y[3]^y[11]^y[19];
119 assign t[19]=x[3]^x[11]^x[19]^y[0]^y[2]^y[4]^y[8]^y[10]^y[16]^y[18]^y[20];
120 assign t[20]=x[0]^x[2]^x[4]^x[8]^x[10]^x[16]^x[18]^x[20]^y[1]^y[5]^y[9]^y
       [17]^y[21];
121 assign t[21]=x[1]^x[5]^x[9]^x[17]^x[21]^y[0]^y[4]^y[6]^y[8]^y[16]^y[20]^y
       [22];
122 assign t[22]=x[0]^x[4]^x[6]^x[8]^x[16]^x[20]^x[22]^y[7]^y[23];
123 assign t[23]=x[7]^x[23]^y[0]^y[4]^y[6]^y[16]^y[20]^y[22]^y[24];
124 assign t[24]=x[0]^x[4]^x[6]^x[16]^x[20]^x[22]^x[24]^y[1]^y[5]^y[17]^y[21]^
       y[25];
125 assign t[25]=x[1]^x[5]^x[17]^x[21]^x[25]^y[0]^y[2]^y[4]^y[16]^y[18]^y[20]^
       y[24]^y[26];
```

```
126 assign t[26]=x[0]^x[2]^x[4]^x[16]^x[18]^x[20]^x[24]^x[26]^y[3]^y[19]^y
    [27];
127 assign t[27]=x[3]^x[19]^x[27]^y[0]^y[2]^y[16]^y[18]^y[24]^y[26]^y[28];
128 assign t[28]=x[0]^x[2]^x[16]^x[18]^x[24]^x[26]^x[28]^y[1]^y[17]^y[25]^y
    [29];
129 assign t[29]=x[1]^x[17]^x[25]^x[29]^y[0]^y[16]^y[24]^y[28]^y[30];
130 assign t[30]=x[0]^x[16]^x[24]^x[28]^x[30]^y[31];
131
132 endmodule
133
134 module rng64 (x,y,t);
135
136 input [62:0] x;
137 input [63:0] y;
138 output [62:31] t;
139
140 assign t[31]=x[31]^y[0]^y[16]^y[24]^y[28]^y[30]^y[32];
141 assign t[32]=x[0]^x[16]^x[24]^x[28]^x[30]^x[32]^y[1]^y[17]^y[25]^y[29]^y
    [33];
142 assign t[33]=x[1]^x[17]^x[25]^x[29]^x[33]^y[0]^y[2]^y[16]^y[18]^y[24]^y
    [26]^y[28]^y[32]^y[34];
143 assign t[34]=x[0]^x[2]^x[16]^x[18]^x[24]^x[26]^x[28]^x[32]^x[34]^y[3]^y
    [19]^y[27]^y[35];
144 assign t[35]=x[3]^x[19]^x[27]^x[35]^y[0]^y[2]^y[4]^y[16]^y[18]^y[20]^y
    [24]^y[26]^y[32]^y[34]^y[36];
145 assign t[36]=x[0]^x[2]^x[4]^x[16]^x[18]^x[20]^x[24]^x[26]^x[32]^x[34]^x
    [36]^y[1]^y[5]^y[17]^y[21]^y[25]^y[33]^y[37];
146 assign t[37]=x[1]^x[5]^x[17]^x[21]^x[25]^x[33]^x[37]^y[0]^y[4]^y[6]^y[16]^
    y[20]^y[22]^y[24]^y[32]^y[36]^y[38];
147 assign t[38]=x[0]^x[4]^x[6]^x[16]^x[20]^x[22]^x[24]^x[32]^x[36]^x[38]^y
    [7]^y[23]^y[39];
148 assign t[39]=x[7]^x[23]^x[39]^y[0]^y[4]^y[6]^y[8]^y[16]^y[20]^y[22]^y[32]^
    y[36]^y[38]^y[40];
149 assign t[40]=x[0]^x[4]^x[6]^x[8]^x[16]^x[20]^x[22]^x[32]^x[36]^x[38]^x
    [40]^y[1]^y[5]^y[9]^y[17]^y[21]^y[33]^y[37]^y[41];
150 assign t[41]=x[1]^x[5]^x[9]^x[17]^x[21]^x[33]^x[37]^x[41]^y[0]^y[2]^y[4]^y
    [8]^y[10]^y[16]^y[18]^y[20]^y[32]^y[34]^y[36]^y[40]^y[42];
151 assign t[42]=x[0]^x[2]^x[4]^x[8]^x[10]^x[16]^x[18]^x[20]^x[32]^x[34]^x
    [36]^x[40]^x[42]^y[3]^y[11]^y[19]^y[35]^y[43];
152 assign t[43]=x[3]^x[11]^x[19]^x[35]^x[43]^y[0]^y[2]^y[8]^y[10]^y[12]^y
    [16]^y[18]^y[32]^y[34]^y[40]^y[42]^y[44];
153 assign t[44]=x[0]^x[2]^x[8]^x[10]^x[12]^x[16]^x[18]^x[32]^x[34]^x[40]^x
    [42]^x[44]^y[1]^y[9]^y[13]^y[17]^y[33]^y[41]^y[45];
154 assign t[45]=x[1]^x[9]^x[13]^x[17]^x[33]^x[41]^x[45]^y[0]^y[8]^y[12]^y
    [14]^y[16]^y[32]^y[40]^y[44]^y[46];
155 assign t[46]=x[0]^x[8]^x[12]^x[14]^x[16]^x[32]^x[40]^x[44]^x[46]^y[15]^y
    [47];
156 assign t[47]=x[15]^x[47]^y[0]^y[8]^y[12]^y[14]^y[32]^y[40]^y[44]^y[46]^y
    [48];
157 assign t[48]=x[0]^x[8]^x[12]^x[14]^x[32]^x[40]^x[44]^x[46]^x[48]^y[1]^y
    [9]^y[13]^y[33]^y[41]^y[45]^y[49];
```

```
158 assign t[49]=x[1]^x[9]^x[13]^x[33]^x[41]^x[45]^x[49]^y[0]^y[2]^y[8]^y[10]^
    y[12]^y[32]^y[34]^y[40]^y[42]^y[44]^y[48]^y[50];
159 assign t[50]=x[0]^x[2]^x[8]^x[10]^x[12]^x[32]^x[34]^x[40]^x[42]^x[44]^x
    [48]^x[50]^y[3]^y[11]^y[35]^y[43]^y[51];
160 assign t[51]=x[3]^x[11]^x[35]^x[43]^x[51]^y[0]^y[2]^y[4]^y[8]^y[10]^y[32]^
    y[34]^y[36]^y[40]^y[42]^y[48]^y[50]^y[52];
161 assign t[52]=x[0]^x[2]^x[4]^x[8]^x[10]^x[32]^x[34]^x[36]^x[40]^x[42]^x
    [48]^x[50]^x[52]^y[1]^y[5]^y[9]^y[33]^y[37]^y[41]^y[49]^y[53];
162 assign t[53]=x[1]^x[5]^x[9]^x[33]^x[37]^x[41]^x[49]^x[53]^y[0]^y[4]^y[6]^y
    [8]^y[32]^y[36]^y[38]^y[40]^y[48]^y[52]^y[54];
163 assign t[54]=x[0]^x[4]^x[6]^x[8]^x[32]^x[36]^x[38]^x[40]^x[48]^x[52]^x
    [54]^y[7]^y[39]^y[55];
164 assign t[55]=x[7]^x[39]^x[55]^y[0]^y[4]^y[6]^y[32]^y[36]^y[38]^y[48]^y
    [52]^y[54]^y[56];
165 assign t[56]=x[0]^x[4]^x[6]^x[32]^x[36]^x[38]^x[48]^x[52]^x[54]^x[56]^y
    [1]^y[5]^y[33]^y[37]^y[49]^y[53]^y[57];
166 assign t[57]=x[1]^x[5]^x[33]^x[37]^x[49]^x[53]^x[57]^y[0]^y[2]^y[4]^y[32]^
    y[34]^y[36]^y[48]^y[50]^y[52]^y[56]^y[58];
167 assign t[58]=x[0]^x[2]^x[4]^x[32]^x[34]^x[36]^x[48]^x[50]^x[52]^x[56]^x
    [58]^y[3]^y[35]^y[51]^y[59];
168 assign t[59]=x[3]^x[35]^x[51]^x[59]^y[0]^y[2]^y[32]^y[34]^y[48]^y[50]^y
    [56]^y[58]^y[60];
169 assign t[60]=x[0]^x[2]^x[32]^x[34]^x[48]^x[50]^x[56]^x[58]^x[60]^y[1]^y
    [33]^y[49]^y[57]^y[61];
170 assign t[61]=x[1]^x[33]^x[49]^x[57]^x[61]^y[0]^y[32]^y[48]^y[56]^y[60]^y
    [62];
171 assign t[62]=x[0]^x[32]^x[48]^x[56]^x[60]^x[62]^y[63];
172
173 endmodule
```

B.4. Módulo de S-box

A continuación se muestra la descripción HDL del módulo de S-box, el cual se conforma de dos tablas de búsqueda, la caja de sustitución y la constante para realizar la suma. La tabla de búsqueda de la caja de sustitución no se muestra completa para mejorar la comprensión del lector, consultar el archivo digital si sea desea ver la tabla completa.

```
1  module sbox76(data_in, constant, select, eval, data_out);
2
3    input wire [7:0] data_in;
4    input wire [3:0] constant;
5    input wire select, eval;
6    output wire [7:0] data_out;
7
8  reg [7:0] sbox,cte;
9  wire [7:0] data;
10 reg [3:0] latch_constant;
11
```

```
12 assign data = data_in&{8{(select)}};
13
14 always @(data)
15     case (data)
16 8'd 0 :sbox=8'd 0 ;
17
18 // Resto de lineas omitidas, para ver la descripcion completa, consultar
      los archivos digitales.
19
20 8'd 95 :sbox=8'd 255 ;
21     endcase
22
23
24 always @(latch_constant)
25     case (latch_constant)
26       4'd0 : cte = 8'd1;
27       4'd1 : cte = 8'd3;
28       4'd2 : cte = 8'd20;
29       4'd3 : cte = 8'd31;
30       4'd4 : cte = 8'd35;
31       4'd5 : cte = 8'd36;
32       4'd6 : cte = 8'd37;
33       4'd7 : cte = 8'd64;
34       4'd8 : cte = 8'd68;
35       4'd9 : cte = 8'd71;
36       4'd10 : cte = 8'd85;
37       4'd11 : cte = 8'd87;
38       4'd12 : cte = 8'd109;
39       4'd13 : cte = 8'd118;
40       4'd14 : cte = 8'd126;
41       4'd15 : cte = 8'd129;
42     endcase
43
44 assign data_out = ((sbox ^ cte)&{8{select}}) | (data_in&{8{~select}});
45
46 always @(constant, eval) begin
47     if (eval)
48         latch_constant <= constant;
49 end
50
51 endmodule
```

B.5. Módulo de S-box^{-1}

A continuación se muestra la descripción HDL del módulo de S-box[1], el cual se conforma de dos tablas de búsqueda, la caja de sustitución y la constante para realizar la suma. La tabla de búsqueda de la caja de sustitución no se muestra completa para mejorar la comprensión

del lector, consultar el archivo digital si sea desea ver la tabla completa.

```
module sbox76_inv(data_in, constant, select, eval, data_out);

  input wire [7:0] data_in;
  input wire [3:0] constant;
  input wire select, eval;
  output wire [7:0] data_out;

reg [7:0] sbox_inv,cte;
wire [7:0] temp,data_gate;
reg [3:0] latch_constant;

always @(temp)
    case (temp)
8'd0:sbox_inv=8'd0;

// Resto de lineas omitidas, para ver la descripcion completa, consultar
   los archivos digitales.

8'd255:sbox_inv=8'd95;
    endcase


always @(latch_constant)
    case (latch_constant)
      4'd0 : cte = 8'd1;
      4'd1 : cte = 8'd3;
      4'd2 : cte = 8'd20;
      4'd3 : cte = 8'd31;
      4'd4 : cte = 8'd35;
      4'd5 : cte = 8'd36;
      4'd6 : cte = 8'd37;
      4'd7 : cte = 8'd64;
      4'd8 : cte = 8'd68;
      4'd9 : cte = 8'd71;
      4'd10 : cte = 8'd85;
      4'd11 : cte = 8'd87;
      4'd12 : cte = 8'd109;
      4'd13 : cte = 8'd118;
      4'd14 : cte = 8'd126;
      4'd15 : cte = 8'd129;
    endcase

assign data_gate = data_in&{8{select}};
assign temp = data_gate ^ cte;
assign data_out = (sbox_inv&{8{select}}) | (data_in&{8{~select}});

always @(constant, eval) begin
    if (eval)
        latch_constant <= constant;
end
```

```
50
51 endmodule
```

B.6. Módulo de Pre-procesamiento

A continuación se muestra la descripción HDL del módulo de Pre-procesamiento.

```
1  module pre(
2    input eval,select,fcn,reset,
3    input wire [7:0] data_in,
4    input wire [8:0] seed_z,
5    output [7:0] data_out
6    );
7
8  reg [8:0] z;
9  wire clk_eff,aux1;
10 wire [7:0] pre;
11 reg [7:0] data;
12
13 assign clk_eff = eval & select & fcn;
14
15
16 always @ (posedge clk_eff) begin
17   if (!reset)
18     z <= seed_z;
19   else
20     z <= {z[0],pre};
21 end
22
23 assign pre[0] = data[0] ^ z[1];
24 assign pre[1] = data[1] ^ z[0] ^ z[2];
25 assign pre[2] = data[0] ^ data[2] ^ z[3];
26 assign pre[3] = data[3] ^ z[0] ^ z[2] ^ z[4];
27 assign pre[4] = data[4] ^ data[2] ^ data[0] ^ z[1] ^ z[5];
28 assign pre[5] = data[5] ^ data[1] ^ z[0] ^ z[4] ^ z[6];
29 assign pre[6] = data[6] ^ data[4] ^ data[0] ^ z[7];
30 assign pre[7] = data[7] ^ z[0] ^ z[4] ^ z[6] ^ z[8];
31
32 assign data_out = ((pre & {8{select}}) | (data& {8{~select}}));
33
34 assign aux1=(eval&fcn);
35
36 always @(data_in, aux1) begin
37     if (aux1)
38         data <= data_in;
39 end
40
41 endmodule
```

B.7. Módulo de Pre-procesamiento inverso

A continuación se muestra la descripción HDL del módulo de Pre-procesamiento inverso.

```
module post(
  input eval,reset,select,fcn,
  input wire [7:0] data_in,
  input wire [8:0] seed_z,
  output [7:0] data_out
  );

reg [8:0] z;
wire [7:0] data,data_temp;
wire clk_eff;

assign clk_eff = eval & select & ~fcn;
assign data_temp = data_in & {8{(select & ~fcn)}};

always @ (posedge clk_eff) begin
  if (!reset)
    z <= seed_z;
  else
    z <= {z[0],data_temp};
end

assign  data[0] = data_temp[0] ^ z[1];
assign  data[1] = data_temp[1] ^ z[0] ^ z[2];
assign  data[2] = data_temp[0] ^ data_temp[2] ^ z[1] ^ z[3];
assign  data[3] = data_temp[3] ^ z[0] ^ z[2] ^ z[4];
assign  data[4] = data_temp[4] ^ data_temp[2] ^ z[1] ^ z[3] ^ z[5];
assign  data[5] = data_temp[5] ^ data_temp[1] ^ z[2] ^ z[4] ^ z[6];
assign  data[6] = data_temp[6] ^ data_temp[4] ^ data_temp[2] ^ data_temp
   [0] ^ z[3] ^ z[5] ^ z[7];
assign  data[7] = data_temp[7] ^ z[0] ^ z[4] ^ z[6] ^ z[8];

assign data_out = (data & {8{select}}) | (data_in & {8{~select}});

endmodule
```

B.8. Módulo de Unidad de Control

A continuación se muestra la descripción HDL del módulo de la Unidad de Control.

```
module control_unit(sck, cs, cfg, sdi, eval, reset, clk_seeds, clk_sdi,
   clk_sdo, psel, cfg_byte);

input sck;
input cs;
```

```
input cfg;
input sdi;
output eval;
output reset;
output clk_sdi;
output clk_seeds;
output clk_sdo;
output psel;
output reg [7:0] cfg_byte;

wire cfg_complete,reset_cfg,n1,n2,n3,n4,pclk,p1,p2,p3,clk_cfg;

assign clk_cfg = sck & ~cfg_complete & ~cs;

always @ (posedge clk_cfg) begin
  cfg_byte <= cfg_byte << 1;
  cfg_byte[0] <= sdi;
end

assign reset_cfg = ~(cfg&cs);

dff m1 (.d(~n1), .rstn(reset_cfg), .clk(sck & ~reset), .q(n1));
dff m2 (.d(~n2), .rstn(reset_cfg), .clk(~n1 & ~reset), .q(n2));
dff m3 (.d(~n3), .rstn(reset_cfg), .clk(~n2 & ~reset), .q(n3)); //Contador
    4 bits para byte de configuracion
dff m4 (.d(~n4), .rstn(reset_cfg), .clk(~n3 & ~reset), .q(n4));

assign pclk=~n1&~n2&~n3&n4;

dff m5 (.d(1'b1), .rstn(reset_cfg), .clk(pclk), .q(cfg_complete));

dff d1 (.d(1'b1), .rstn(~cfg), .clk(~cs), .q(reset));

dff o1 (.d(~p1), .rstn(reset), .clk(~sck), .q(p1));
dff o2 (.d(~p2), .rstn(reset), .clk(~p1), .q(p2));
dff o3 (.d(~p3), .rstn(reset), .clk(~p2), .q(p3));

assign eval = (sck&p1&p2&p3)|cs;
assign clk_seeds = sck & ~reset & ~cs;
assign clk_sdi = sck & cfg_complete & ~cs;
assign clk_sdo = ~((sck & reset) | cs);

dff sdo (.d(1'b1), .rstn(~eval), .clk(sck), .q(psel));

endmodule
```

B.9. Flip-Flop D

A continuación se muestra la descripción HDL de un Flip-Flop tipo D.

```
module dff ( input d,
             input rstn,
             input clk,
             output reg q);

  always @ (posedge clk or negedge rstn)
       if (!rstn)
          q <= 0;
       else
          q <= d;
endmodule
```

B.10. Latch D

A continuación se muestra la descripción HDL de un Latch tipo D.

```
module d_latch (q,d,en);

output reg q;
input d;
input en;

always @(d, en) begin
    if (en)
        q <= d;
end
endmodule
```

B.11. Decodificador de 2 bits

A continuación se muestra la descripción HDL de un decodificador de 2 bits.

```
module decoder(A1, A0, X2, X1, X0);
   output X0;
   output X1;
   output X2;

   input  A1;
   input  A0;

```

```
9     wire    A1n;
10    wire    A0n;
11
12    not(A1n, A1);
13    not(A0n, A0);
14
15    and(X0, A1n, A0n);
16    and(X1, A1n, A0);
17    and(X2, A1, A0n);
18
19 endmodule
```

Bibliografía

[1] Karsten König and Andreas Ostendorf. *Optically induced nanostructures: biomedical and technical applications*. De Gruyter, 2015.

[2] Neil HE Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[3] Joseph Weber. Detection and generation of gravitational waves. *Physical Review*, 117(1):306, 1960.

[4] Cian O'Luanaigh. New results indicate that new particle is a higgs boson. *CERN-http://home. cern/about/updates/2013/03/new-results-indicate-new-particle-higgs-boson (acessado em 21/08/2016)*, 2013.

[5] Prasanta Misra. *Physics of condensed matter*. Academic Press, 2011.

[6] Gordon E Moore. Cramming more components onto integrated circuits. electronics 38 (8): 114–117, 1965.

[7] Swarup Bhunia and Mark Tehranipoor. *Hardware security: a hands-on learning approach*. Morgan Kaufmann, 2018.

[8] Sanu K Mathew, Farhana Sheikh, Michael Kounavis, Shay Gueron, Amit Agarwal, Steven K Hsu, Himanshu Kaul, Mark A Anders, and Ram K Krishnamurthy. 53 gbps native gf$(2^4)^2$ composite-field aes-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors. *IEEE Journal of Solid-State Circuits*, 46(4):767–776, 2011.

[9] Panu Hamalainen, Timo Alho, Marko Hannikainen, and Timo D Hamalainen. Design and implementation of low-area and low-power aes encryption hardware core. In *9th EUROMICRO conference on digital system design (DSD'06)*, pages 577–583. IEEE, 2006.

[10] Franois Mace, Franois-Xavier Standaert, Jean-Jacques Quisquater, et al. Asic implementations of the block cipher sea for constrained applications. In *Proceedings of the Third International Conference on RFID Security-RFIDSec*, volume 2007, pages 103–114, 2007.

[11] Gustavus J Simmons. A survey of information authentication. *Proceedings of the IEEE*, 76(5):603–620, 1988.

[12] Jesús Urıas, Gelasio Salazar, and Edgardo Ugalde. Synchronization of cellular automaton pairs. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 8(4):814–818, 1998.

[13] Jesus Urias, Edgardo Ugalde, and Gelasio Salazar. A cryptosystem based on cellular automata. *Chaos (Woodbury, NY)*, 8(4):819–822, 1998.

[14] Marcela Mejıa and Jesús Urıas. An asymptotically perfect pseudorandom generator. *Discrete and Continuous Dynamical Sys*, 7:115–126, 2001.

[15] MT Ramirez-Torres, JS Murguia, and M Mejía Carlos. Image encryption with an improved cryptosystem based on a matrix approach. *International Journal of Modern Physics C*, 25(10):1450054, 2014.

[16] JS Murguía, G Flores-Erana, M Mejía Carlos, and Haret Codratian Rosu. Matrix approach of an encryption system based on cellular automata and its numerical implementation. *International Journal of Modern Physics C*, 23(11):1250078, 2012.

[17] JA Aboytes-González, JS Murguía, M Mejía-Carlos, H González-Aguilar, and MT Ramírez-Torres. Design of a strong s-box based on a matrix approach. *Nonlinear Dynamics*, 94(3):2003–2012, 2018.

[18] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601, 1983.

[19] Marcela Mejía Carlos. *Encriptación por Sincronización en Autómatas Celulares*. PhD thesis, Universidad Autónoma de San Luis Potosí, Instituto de Investigación en Comunicación Óptica, 12 2001.

[20] Jesús Urıas, Gelasio Salazar, and Edgardo Ugalde. Synchronization of cellular automaton pairs. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 8(4):814–818, 1998.

[21] Jesus Urias. Cryptography primitives based on a cellular automaton. In *Coding Theory, cryptography and related areas*, pages 244–248. Springer, 2000.

[22] Massimo Alioto. Trends in hardware security: From basics to asics. *IEEE Solid-State Circuits Magazine*, 11(3):56–74, 2019.

[23] A Prathiba and VS Bhaaskaran. Lightweight s-box architecture for secure internet of things. *Information*, 9(1):13, 2018.

[24] James A Samson, David L Ederer, Thomas Lucatorto, and Marc De Graef. *Vacuum ultraviolet spectroscopy I*. Academic press, 1998.

[25] MT Ramírez-Torres, M Mejía-Carlos, and JS Murguía. Numerical implementation of a real-time encryption system. *Procedia Engineering*, 35:182–191, 2012.

[26] Marco Tulio Ramírez-Torres, JS Murguia, and Marcela Mejía-Carlos. Fpga implementation of a reconfigurable image encryption system. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, pages 1–4. IEEE, 2014.

[27] Cem Unsalan and Bora Tar. *Digital system design with FPGA: implementation using Verilog and VHDL*. McGraw-Hill Education, 2017.

[28] Tsung-Ching Huang, Kenjiro Fukuda, Chun-Ming Lo, Yung-Hui Yeh, Tsuyoshi Sekitani, Takao Someya, and Kwang-Ting Cheng. Pseudo-cmos: A design style for low-cost and robust flexible electronics. *IEEE Transactions on Electron Devices*, 58(1):141–150, 2010.

[29] Tsung-Ching Huang and Kwang-Ting Cheng. Design for low power and reliable flexible electronics: Self-tunable cell-library design. *Journal of Display Technology*, 5(6):206–215, 2009.

[30] Steven M. Rubin. *Static Free Software*, 2017 (accessed April 1, 2020).

[31] Inc Analog Devices. *LTspice*, (accessed April 1, 2020).

[32] CIDESI. *Fabricación de Circuitos Integrados*, (accessed April 1, 2020).

[33] Erik Brunvand. *Digital VLSI chip design with Cadence and Synopsys CAD tools*. Addison-Wesley, 2010.

[34] Robert F Sproull and Ivan E Sutherland. Logical effort: Designing for speed on the back of an envelope, 1991.