Universidad de Sevilla

Grado en Física

Departamento de Electrónica y Electromagnetismo

Trabajo de Fin de Grado:

Diseño microelectrónico de circuitos criptográficos de altas prestaciones y evaluación de su seguridad.

Tutores: Antonio José Acosta Jiménez

Erica Tena Sánchez

Autor: Ignacio María Delgado Lozano

Septiembre 2017

Índice

1. Introducción.

Los dispositivos electrónicos presentes, a día de hoy, en nuestra vida cotidiana tales como teléfonos móviles y tarjetas de crédito manejan información secreta que se pretende que no sea revelada a terceros, protegiendo así la privacidad e intimidad de los usuarios de dichos dispositivos. La seguridad tecnológica, por medio de la criptografía, se abre paso en un mundo en que cada vez más gente utiliza dispositivos que deben ser seguros ante el posible ataque de terceras personas que quieran utilizar la información que se pretende proteger con fines alevosos. Esta transformación de la sociedad hace necesaria la intervención de la comunidad científica para encontrar soluciones que permitan mantener la seguridad en cuanto a información privada se refiere, es decir, que nos permita tener dispositivos electrónicos seguros.

Estos dispositivos electrónicos utilizan ciertos recursos criptográficos para mantener la información a salvo. Técnicamente, los algoritmos criptográficos son funciones matemáticas que permiten encriptar la información a través de una clave personal, proporcionando a la salida un texto encriptado que solo puede ser descifrado por quien sea poseedor de la clave secreta. Estos algoritmos pueden mostrar una robustez prácticamente perfecta a nivel teórico. No obstante, aun cuando todas las medidas teóricas han sido tomadas para implementar algoritmos que matemática, teórica y formalmente no revelen los datos que se deben ocultar, existen ciertos mecanismos para penetrar la seguridad de los dispositivos criptográficos debido a su implementación física. Así, los ataques laterales o *Side Channel Attacks* (SCAs), en ocasiones, consiguen revelar información secreta pese a la teórica invulnerabilidad de los dispositivos [1]-[3].

La metodología para llevar a cabo un ataque SCA consiste en obtener información a partir de alguna medida física que pueda hacerse del circuito criptográfico, tales como el consumo de potencia [3] o el retraso entre entrada y salida del bloque criptográfico [1]. Haciendo un análisis de datos exhaustivo, y siendo cuidadosos con el procesado de los mismos puede llegarse a revelar la clave secreta del dispositivo [2].

Dentro de los ataques SCA, destacan los llamados DPA (*Differential Power Analysis)*. Estos ataques se basan en la dependencia del dato procesado con el consumo de potencia para, a través de un análisis estadístico, obtener la clave secreta ya que las puertas digitales consumen potencia en función del dato que conmutan [2]. En cualquier puerta imaginable, pueden darse cuatro situaciones a la entrada: que la entrada pase de un valor bajo “0” a un valor alto “1”,

que pase de “1” a “0”, que se mantenga en “0” o que se mantenga en “1”. Estos cambios a la entrada producirán diferentes situaciones a la salida, dependiendo de la función implementada que provocarán un cierto consumo dinámico de potencia. Si obtenemos la traza del consumo de potencia de un dispositivo, es posible inferir qué transición ha tenido lugar y, por tanto, saber el valor de entrada al dispositivo. Los ataques DPA a dispositivos criptográficos utilizan este método para obtener la información secreta realizando un análisis estadístico de un gran número de trazas de potencia del bloque criptográfico para un alto número de patrones diferentes, mostrando una gran efectividad desde que a finales de los 90 se demostrara la viabilidad de este procedimiento [1], [3]-[5].

Es por ello, que si queremos mantener a salvo nuestra información, se deben buscar alternativas para que estos ataques no puedan predecir la clave que se está utilizando. Algunas técnicas consisten en enmascarar la relación entre consumo de potencia y dato procesado a partir de la introducción de retrasos aleatorios, u operaciones arbitrarias que añaden un consumo adicional al esperado por el algoritmo original, es lo que se conoce como *masking* [2], [6]. Por otro lado, existen otras técnicas que buscan no revelar ningún tipo de información lateral, las llamadas técnicas de ocultamiento o *hiding* tratan de que no exista ningún tipo de correlación entre el dato procesado y el consumo de potencia [2], [4]. Para este fin, necesitamos que nuestro circuito criptográfico tenga un consumo constante o, al menos, con las menores variaciones posibles. Obtener un circuito cuyo consumo sea completamente constante es imposible, sin embargo, consideraremos que dicho circuito es seguro cuando el tiempo y los recursos requeridos por un ataque DPA, o de cualquier otro tipo, para hallar la clave secreta superen en coste a la información que se pretende revelar. Este propósito no siempre es fácil de alcanzar, por ejemplo, el clásico estilo lógico CMOS que es utilizado en multitud de aplicaciones electrónicas muestra una gran dependencia entre dato procesado y consumo de potencia siendo muy vulnerable a nivel criptográfico [2], [4].

El objetivo de este trabajo consistirá, entonces, en el diseño y comparación de sistemas criptográficos resistentes a ataques laterales. Como medio para lograr este objetivo, tendremos que diseñar celdas lógicas en un estilo seguro y apto para aplicaciones criptográficas. El proyecto se enmarca dentro de la línea que ha venido siguiendo el grupo de investigación que lo dirige, habiéndose hallado múltiples mejoras, en el ámbito de la criptografía, en los últimos años como las que pueden encontrarse en [4], [5]. Siguiendo en esa línea, este proyecto se hace necesario ya que analiza y compara sistemas criptográficos

"lightweight" que optimizan recursos como el consumo de potencia o el área utilizando un menor número de bits que los sistemas que se venían utilizando tradicionalmente [7].

Por tanto, las tareas a realizar serán las siguientes: encontrar un estilo lógico que proporcione la máxima seguridad posible en aplicaciones criptográficas, diseñar puertas lógicas una vez se haya seleccionado dicho estilo y, con ellas, proceder al diseño de bloques clave en la encriptación de información secreta. Por último, estos bloques serán comparados entre sí intentándose determinar cuál de ellos presenta menos vulnerabilidades.

Hecha esta introducción, podemos decir que nuestro trabajo, en el capítulo 2, empezará por centrarse en la elección y justificación de un estilo que sea válido para aplicaciones criptográficas, es decir, que tenga un consumo de potencia lo más simétrico posible y en el diseño de celdas lógicas en el estilo seleccionado. En el capítulo 3, se diseñarán partes de circuitos criptográficos que son claves a la hora de codificar la información, las llamadas S-Box (*Substitution Box*). De este modo, y una vez hayamos seleccionado un estilo lógico acorde con nuestra aplicación, se desarrollarán en Cadence a partir de las especificaciones de dos de los cifradores de bloque que se encuentran en el estado del arte de la criptografía, dos Sboxes de 4 bits y una de 8 bits y se comprobará su funcionalidad de manera automatizada a partir de un código realizado en MatLab. En el capítulo 4, se estudiará el comportamiento de dichas Sboxes a partir de parámetros estadísticos y se hallará una estimación de la seguridad de cada una de ellas, centrándonos especialmente en la comparación entre la Sbox de 8 bits y las de 4 bits y haciendo un análisis sobre si es rentable en términos de seguridad y consumo energético implementar una Sbox de 8 bits en lugar de dos de 4 bits. Por último, en el capítulo 5 sintetizamos las conclusiones de este proyecto y analizamos las posibles vías y líneas futuras de investigación que nos permitan tener un mayor conocimiento acerca de la seguridad de nuestras celdas criptográficas y, sobre todo, que nos ayuden a incrementar dicha seguridad en un futuro cercano.

2. Diseño de celdas en un estilo lógico útil para aplicaciones criptográficas.

Uno de los primeros aspectos importantes que debemos decidir a la hora de realizar este trabajo es la elección de un diseño y arquitectura lógicos que sea apto para la aplicación en circuitos criptográficos y que permita un uso eficiente de encriptación a partir de celdas seguras. Hasta el año 1998, cuando Kocher et al. [3] mostraron que a través de un ataque basado en el consumo de potencia se podía revelar la clave de un dispositivo criptográfico, se venía utilizando el estilo lógico CMOS. A partir de ese momento, se demostró que dicho estilo lógico presentaba una notoria dependencia entre el consumo de potencia y el dato procesado, con lo cual, pronto pudo observarse que no se trataba de un estilo adecuado para utilizarse en criptografía. En consecuencia, se hizo necesario buscar estilos lógicos alternativos que permitiesen alcanzar los objetivos demandados por la aplicación criptográfica, como por ejemplo las soluciones SecLib (*Secure Library*) [8] , DyMCL (*Dynamic Current Mode Logic*) [9] , LSCML (*Low-Swing Current Mode Logic*) [10] o SABL (*Sense Amplifier Based Logic*) [11].

Una de las propuestas más asentadas y reconocidas para realizar circuitos criptográficos seguros es la concerniente a las familias DPL *(Dual Rail with Precharge Logic*) [9], [11], [12]. En este tipo de circuitos se utilizan dos estados de funcionamiento diferentes: el de precarga (llevan a las salidas a un valor fijo después de cada evaluación) y evaluación (llevan a las salidas a unos valores dependientes de las entradas y de la función lógica que se quiera implementar), utilizando una lógica de doble raíl, esto es, para cada entrada y salida de la celda se proporciona: la señal en cuestión Q y su complementaria $\bar{Q}$. El diseño lógico que ha mostrado ser más eficiente y seguro ante ataques alevosos que pretendan revelar la clave es el SABL [4], [11], [13]. Por tanto, el diseño de nuestras celdas lógicas vendrá dado por esta arquitectura, que tomaremos como referencia y que utilizaremos a lo largo de todo nuestro trabajo.

Dado que los ataques contra los que queremos hacer circuitos resistentes son aquellos basados en el consumo de potencia, especialmente los DPA (*Differential Power Analysis*), deberemos hacer un análisis orientativo sobre el consumo de potencia de una celda lógica. La potencia dinámica de una celda lógica puede aproximarse por [2]:

$$P_D = \alpha \cdot C \cdot f_{clk} \cdot {V_{dd}}^2 \qquad (1)$$

En esta ecuación, podemos observar que existen dos parámetros que no dependen del dato procesado, estos son: la frecuencia de reloj f_{clk} así como la tensión de alimentación V_{dd}. Sin embargo, el parámetro α (actividad de conmutación) y C (capacidad de carga) sí tienen dependencia con el dato procesado. Como nuestro objetivo es conseguir que el consumo de potencia sea independiente del dato procesado, tendremos que intentar que α y C sean constantes para cualquier transición que se pueda dar en la celda lógica. Una de las soluciones para llevar a cabo este propósito es la construcción de celdas diferenciales, dinámicas y completamente simétricas.

En primer lugar, a través de un estilo dinámico y diferencial se puede conseguir que $\alpha = 1$ para cualquier situación que podamos pensar en nuestra celda lógica. El procedimiento es el siguiente: tenemos dos estados de operación diferentes, a saber, la evaluación y la precarga actuando con lógica de doble raíl tal y como discutimos anteriormente. En la fase de precarga, ambas salidas Q y $\bar{Q}$ toman el mismo valor. Cuando pasamos al estado de evaluación, la salida Q tomará el valor definido por las entradas y la función lógica que implemente la celda en cuestión que estemos tratando y $\bar{Q}$ tomará el valor complementario, llevando, en cualquier caso, a una conmutación a la salida y siendo entonces, el factor de conmutación para cualquier situación $\alpha = 1$.

En segundo lugar, una arquitectura completamente simétrica nos permitiría asegurar que el almacenamiento de carga en los distintos nodos que presente la celda sea siempre el mismo, y por tanto, obtener un parámetro C constante en cualquier tipo de situación, sin embargo, y como veremos más adelante, se presentan ciertas dificultades al tener que considerar que la estructura no puede ser, evidentemente, exactamente la misma a la hora de tener que implementar diferentes funciones lógicas en cada una de las celdas.

En el marco del estilo lógico de las celdas SABL podemos diferenciar dos tipos de bloques.

- **Bloque controlador de fases de precarga y evaluación:** Este bloque consta de transistores PMOS y NMOS, que vienen controlados por una señal de reloj “clk” que hacen que el valor de Q y $\bar{Q}$ sea el mismo en la fase de precarga, mientras que conectan cada una de las ramas diferenciales a la salida en la fase de evaluación. En nuestro trabajo, vamos a diferenciar los bloques en función de los valores de tensión a los que se encuentren. En este TFG, el bloque de control puede ser identificado como

el bloque DPUN (Differential Pull Up Network), véase la figura 2.1, en cuyo caso las salidas Q y $\bar{Q}$ se fijarán a Gnd en la fase de precarga debido a que el bloque DPUN consta de inversores a la salida.

- **Bloque combinacional:** En la figura 2.1 se corresponde con el bloque DPDN, que está implementado en nuestro caso particular con transistores NMOS. Dicho bloque realiza la operación lógica diferencial de la celda en cuestión que estemos tratando.

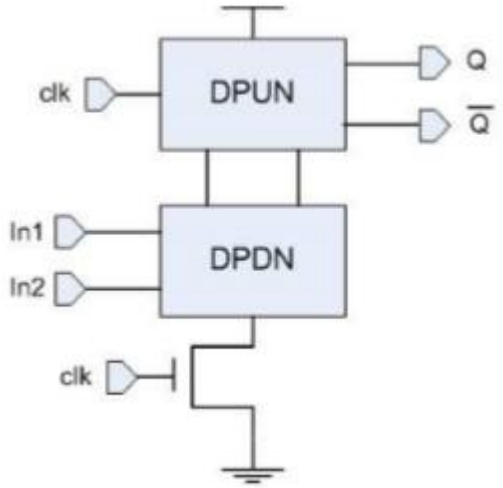


Fig 2.1. Estructura de un estilo lógico dinámico y diferencial con precarga a V_{dd} y evaluación por medio de transistores NMOS. Si clk=1 estamos en fase de evaluación, clk=0 corresponde a la fase de precarga.

2.1. Construcción del bloque DPUN en la estructura SABL.

Para que la arquitectura lógica de los circuitos criptográficos sea segura, además de tener en cuenta un estilo diferencial y dinámico, los nodos de salida deben estar balanceados. Este requisito se hace necesario puesto que los nodos diferenciales que unen la red DPUN con la red DPDN van directamente conectados a los nodos de salida provocando que la capacidad de carga dependa del bloque DPDN. Si las capacidades de los nodos de las ramas diferenciales son iguales el consumo de potencia siempre sería el mismo independientemente de cuáles sean los datos dada la ecuación (1) y lo ya discutido acerca del parámetro de conmutación α. Sin embargo, como también comentábamos antes, el bloque DPDN no podrá ser siempre totalmente simétrico debido a la función implementada.

En el diseño SABL propuesto [11], [12], [14], se introduce un transistor NMOS que siempre está conduciendo entre los nodos n_1 y n_2 que conectan la red DPDN con la DPUN, de tal manera que n_1 y n_2 van al valor lógico “0” tras la fase de evaluación, mientras que irán a un valor intermedio quedando flotantes en la fase de precarga. De este modo, se pretende

eliminar la información sobre la conmutación que tuvo lugar en la anterior evaluación evitando el efecto memoria que podría ser utilizado por un atacante.

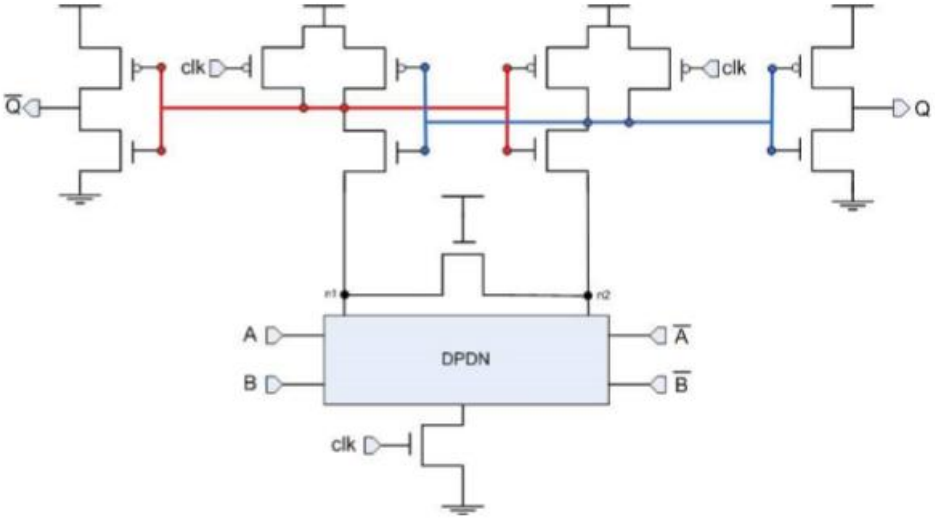
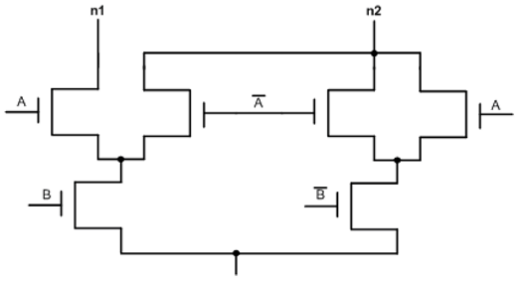


Fig 2.2. Bloque DPUN desarrollado para la celda SABL con transistor NMOS para evitar el efecto memoria.

2.2. Construcción del bloque DPDN en la estructura SABL.

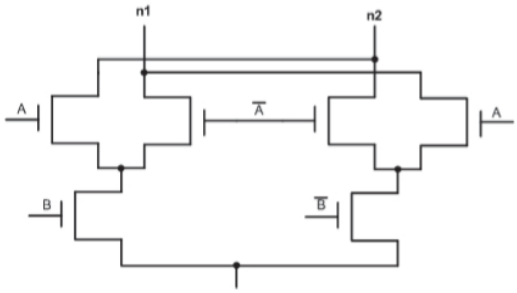
Para la construcción del bloque DPDN, vamos a implementar las puertas lógicas AND/NAND, XOR/XNOR y OR/NOR. A lo largo de este apartado vamos a mostrar la arquitectura que va a tener el bloque DPDN en cada caso, dependiendo de la función lógica que implemente así como la casuística de dicha función. Debemos recordar que el bloque DPDN, al estar trabajando en una lógica de doble raíl, va a tener dos ramas: una que genera el valor dado por la función implementada, y su complementario. Además, por los motivos expuestos anteriormente, la simetría entre ambas ramas debe ser la máxima posible, esto implica que el bloque deberá tener el mismo número de transistores en serie por rama, y que la salida de cada rama deberá estar cargada por el mismo número de transistores.

Con estas condiciones, los diferentes bloques DPDN quedarán diseñados tal y como se muestra en las siguientes figuras e implementando la operación lógica que se muestra en las siguientes tablas[4], [14]. Siendo A y B las entradas al bloque, $\bar{A}$ y $\bar{B}$ sus complementarias mientras que n_1 y n_2 serán los nudos de salida. En el caso de las celdas AND/NAND y XOR/XNOR, el nudo n_1 será el que represente la salida de la función lógica implementada (esto es, AND y XOR, respectivamente) mientras que n_2 representará el complementario de la función lógica (NAND y XNOR). Sin embargo, en el caso de la celda OR/NOR, el nudo n_2 representará la salida de la función lógica implementada (OR) y n_1 será su complementario.



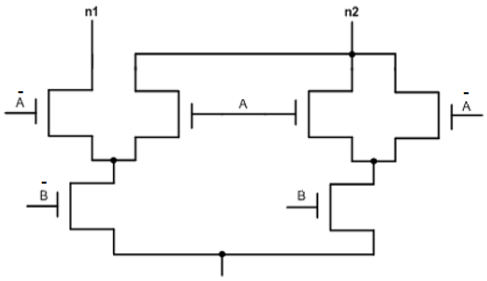
ENTRADA		SALIDA	
A	B	A AND B (n_1)	A NAND B (n_2)
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

Fig 2.3. a) Bloque DPDN de una celda AND/NAND y b) Casuística para las operaciones lógicas AND/NAND en fase de evaluación.



ENTRADA		SALIDA	
A	B	A XOR B (n_1)	A XNOR B (n_2)
0	0	0	1
1	0	1	0
0	1	1	0
1	1	0	1

Fig 2.4. a) Bloque DPDN de una celda XOR/XNOR y b) Casuística para las operaciones lógicas XOR/XNOR en fase de evaluación.



ENTRADA		SALIDA	
A	B	A OR B (n_2)	A NOR B (n_1)
0	0	0	1
1	0	1	0
0	1	1	0
1	1	1	0

Fig 2.5. a) Bloque DPDN de una celda OR/NOR y b) Casuística para las operaciones lógicas OR/NOR en fase de evaluación.

Obsérvese que las celdas OR/NOR y AND/NAND pueden parecer idénticas, pero se han intercambiado las entradas por sus complementarias, este hecho provoca que el resultado a la salida corresponda a una función lógica distinta para ambos casos.

2.3. Diseño en Cadence de celdas SABL.

Una vez que hemos presentado tanto el diseño del bloque DPUN, común a todas las celdas que implementan funciones lógicas en la arquitectura SABL, como el del bloque DPDN para cada una de las celdas según la función que queramos implementar, estamos en disposición de presentar el diseño completo de las diferentes celdas.

Según lo visto, si conectamos el bloque DPUN con cada bloque DPDN, en función de la puerta lógica que pretendamos implementar, vamos a tener el diseño del esquemático completo de cada celda que mostramos en las siguientes figuras. Nosotros vamos a centrarnos, especialmente, en el diseño y el correcto funcionamiento de la celda OR/NOR, que ha sido aquella diseñada específicamente en este proyecto y que tendrá utilidad en aplicaciones futuras.

En primer lugar, se diseñó la puerta OR/NOR a nivel de transistor siguiendo la estructura SABL de las puertas lógicas AND/NAND y XOR/XNOR que ya estaban diseñadas según [13]. Posteriormente, procedemos a generar un bloque que engloba al nivel de jerarquía de transistor, que es el más bajo de todos, de tal manera que se facilite la tarea para poder combinar dichos bloques, cada uno de los cuales implementa una función lógica diferente.

En todos los casos, se utilizó para el diseño de estas celdas el entorno "Virtuoso Schematic Editor" de Cadence. Asimismo, utilizando otras herramientas Cadence como su "Analog Design Environment" fueron testeadas para comprobar su correcto funcionamiento. El diseño de las celdas se realizó en tecnología TSMC 90 nm.

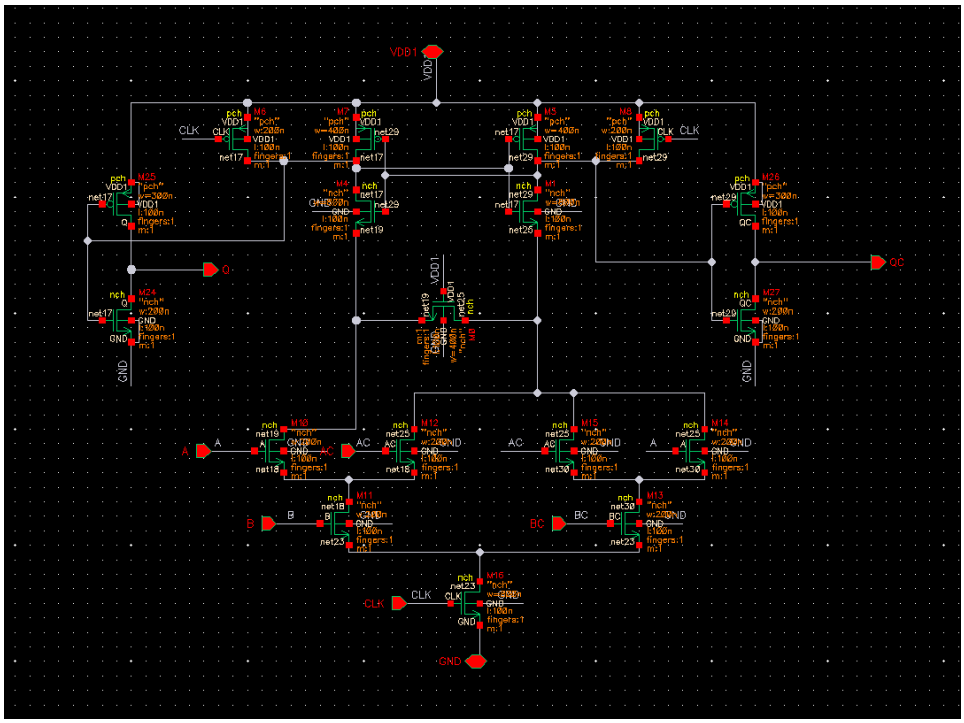


Fig 2.6. Esquemático de la celda AND/NAND en arquitectura SABL diseñada en Cadence.

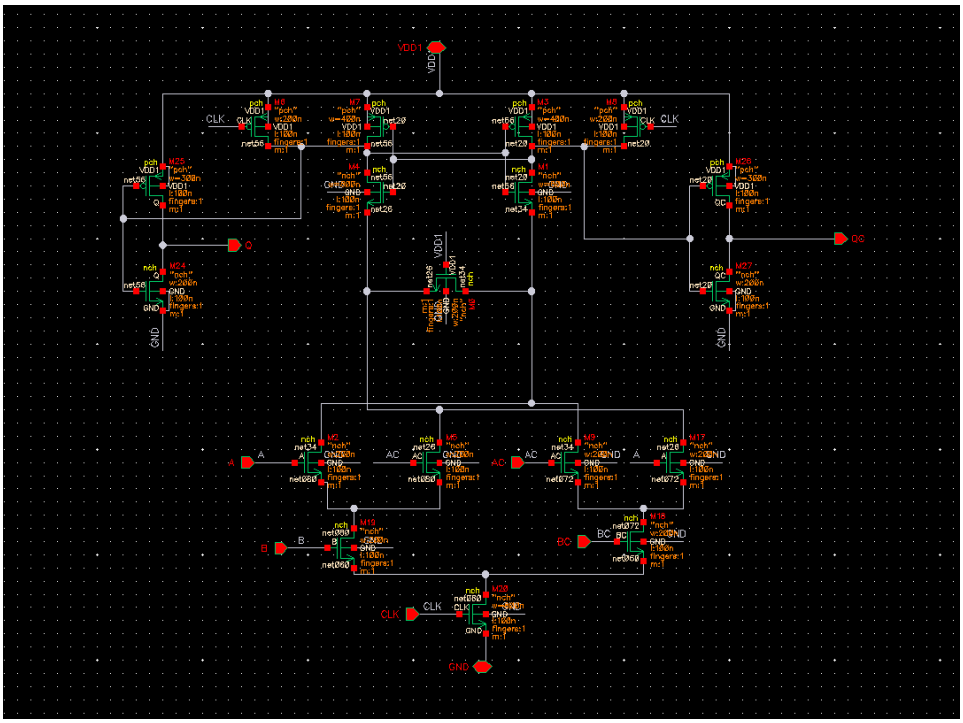


Fig 2.7. Esquemático de la celda XOR/XNOR en arquitectura SABL diseñada en Cadence.

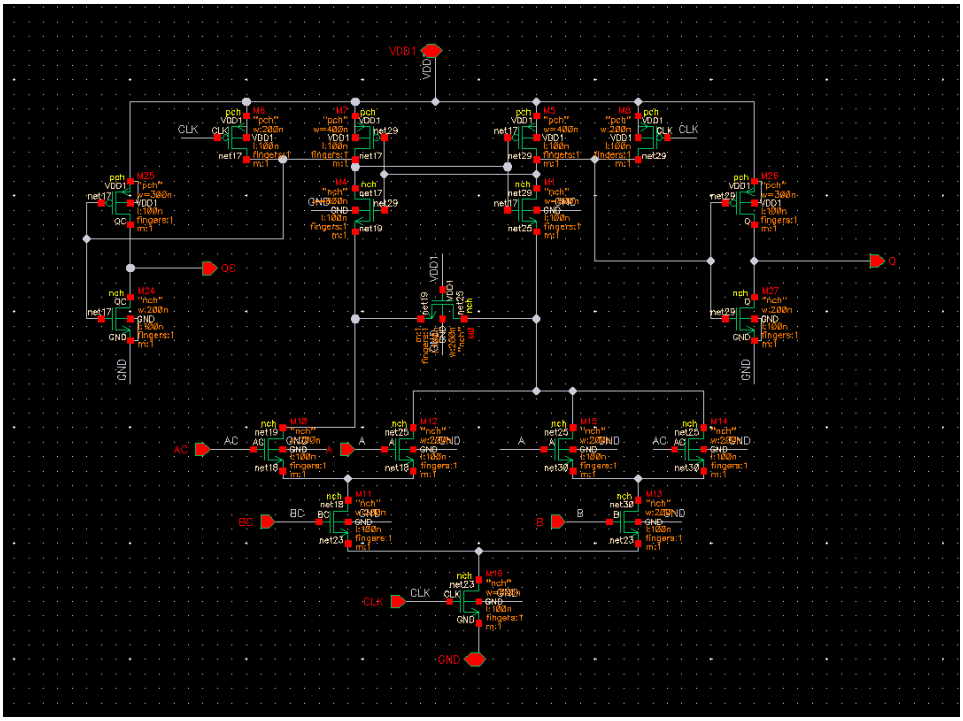


Fig 2.8. Esquemático de la celda OR/NOR en arquitectura SABL diseñada en Cadence.

Englobando este esquemático en un nivel de jerarquía superior, podemos generar un bloque que realice las operaciones dadas por la tabla de verdad de una puerta lógica OR/NOR. Tendremos, por tanto, utilizando la lógica de doble raíl, un bloque con dos entradas “A” y “B” y sus respectivas complementarias, otro pin dedicado a la señal de reloj que marca las fases de precarga y evaluación y dos pines de salida que proporcionarán la salida y su

complementaria, respetando una vez más la lógica de doble raíl. El bloque también consta de otros dos pines que determinan el valor de la tensión de polarización y la tierra. Este bloque en cuestión se presenta en la Fig 2.9, bloques análogos fueron diseñados en [13] para las celdas AND/NAND y XOR/XNOR, cada una de estas tres celdas van a ser utilizadas a lo largo de este trabajo.

La geometría de los transistores seleccionada es la mínima que permite la tecnología, salvo en aquellos transitores cuya geometría fue optimizada en [13] para obtener las mejores prestaciones de consumo y velocidad.

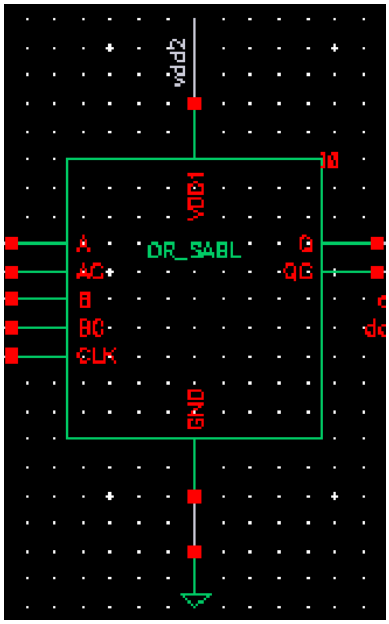


Fig 2.9. Nivel de jerarquía superior de la celda OR/NOR en arquitectura SABL diseñada en Cadence.

Ahora que hemos presentado el diseño completo de la celda OR/NOR que nos será necesaria más adelante, vamos a precisar de un esquemático que nos sirva de test para comprobar la funcionalidad de dicha celda. Lo que vamos a hacer es generar señales cuadradas con distintas frecuencias donde el valor alto va a encontrarse a 1.2 V, mientras que el valor bajo será de 0 V (con estas tensiones trabajaremos durante todo este proyecto), a continuación, sirviéndonos de un circuito auxiliar conseguiremos generar las entradas que van a atacar a la celda OR/NOR que vamos a testear, previo paso por un conjunto de otras dos celdas OR/NOR que colocamos para aislar la celda que queremos testear, y evitar ruidos y señales defectuosas. Del mismo modo, y por un argumento similar, a la salida de la celda que buscamos testear hemos situado otra puerta OR. Presentamos un esquema ilustrativo de este procedimiento y el esquemático de test en las Fig 2.10 y Fig 2.11 respectivamente.

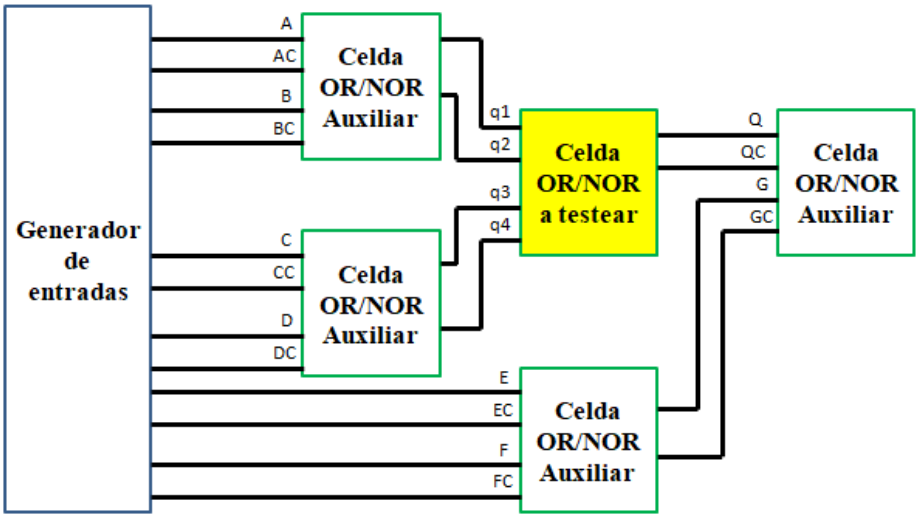


Fig 2.10. Esquema ilustrativo para el esquemático de test de la celda OR.

Donde las letras mayúsculas que van desde la A hasta la G representan las entradas y salidas de las celdas auxiliares. Dado que tenemos que respetar la lógica de doble raíl, las señales complementarias a éstas van designadas por la letra correspondiente acompañada de C. Por su parte, q1 y q3 son las entradas a la celda de testeo y q2 y q4 sus respectivas complementarias. La salida de la celda de testeo se representa por Q y su complementaria por QC. Hemos elegido esta notación para estar en consonancia con la Fig 2.12.

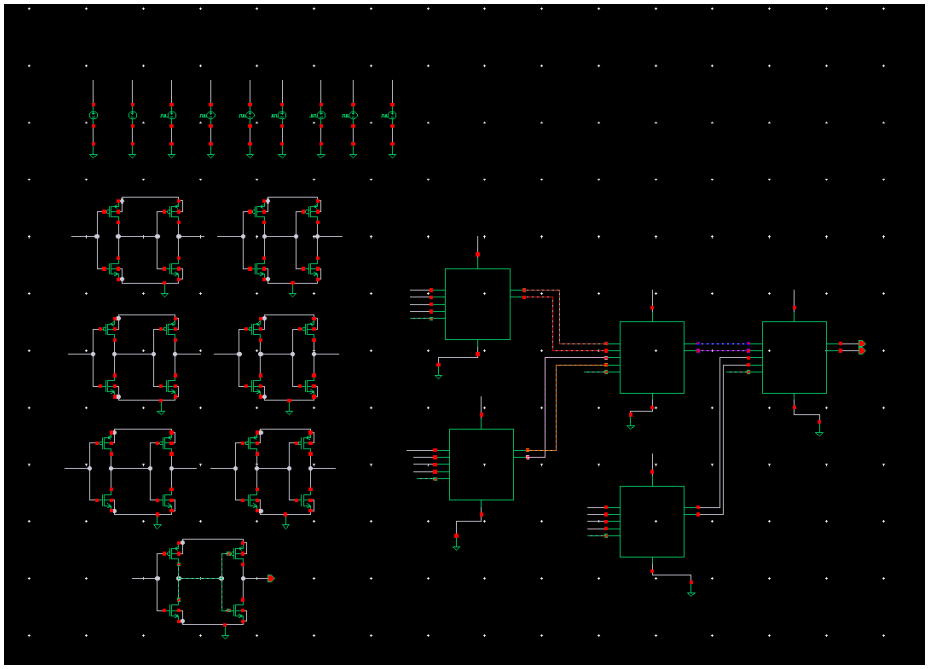


Fig 2.11. Esquemático de test para la celda OR.

Corriendo esta simulación con un análisis transitorio de 100 ns, con una señal de reloj de periodo 10 ns, obtenemos los resultados que pueden observarse en la siguiente figura, donde /q1 y /q3 representan las entradas a la celda OR testeada, /q2 y /q4 son sus respectivas señales complementarias, por su parte, clk representa la señal de reloj y Q y QC son la salida y su complementaria.

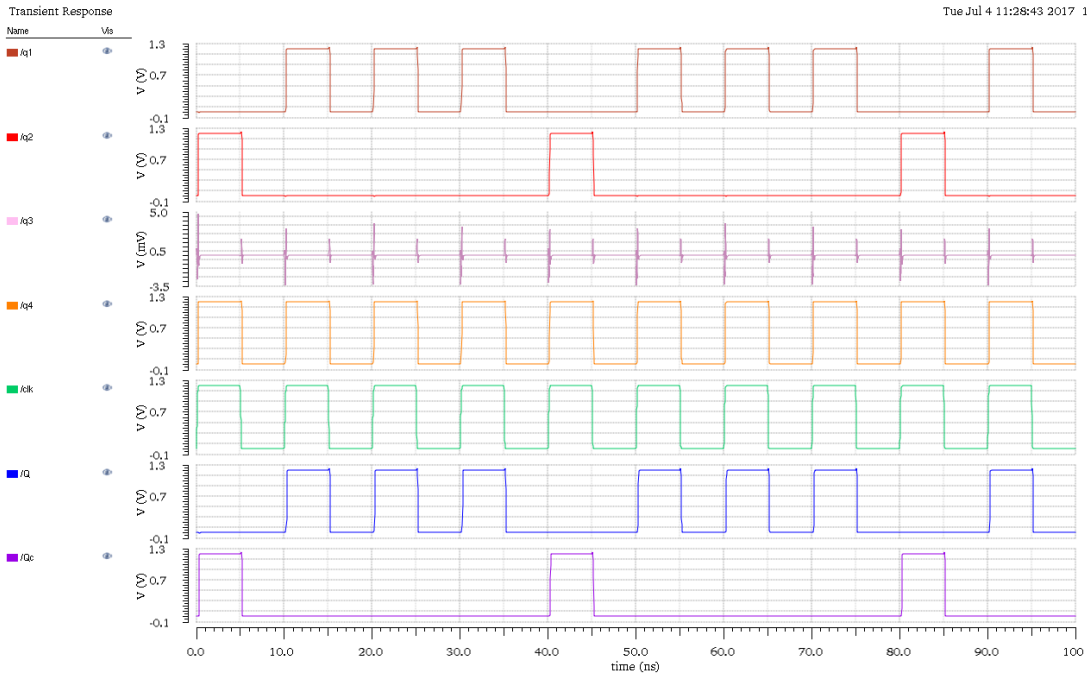


Fig 2.12. Resultado del análisis transitorio para el test de la celda OR/NOR.

Viendo esta figura podemos comprobar que la implementación de la función lógica OR/NOR en esta arquitectura ha sido correcta. En primer lugar, podemos observar cómo se está respetando la lógica de doble raíl, observándose como las señales y sus complementarias van a valores opuestos durante la fase de evaluación, yendo al mismo en la fase de precarga: cuando el "clk" se encuentra al valor lógico "0" todas las señales van a "0" (estamos en fase de precarga) mientras que cuando el "clk" se encuentra al valor lógico "1", las señales van al valor que les corresponda en cada caso, y sus complementarias van al valor contrario, tal y como corresponde en la fase de evaluación. Además, puede verse como la señal a la salida cumple con la casuística presentada para la celda OR para todos los patrones introducidos. Al haberse fijado una de las entradas constantemente a valor lógico "0" a la hora de realizar el test, no puede comprobarse mediante la anterior figura el caso "A=1, B=1", pero con un simple cambio en las condiciones de las fuentes de tensión que generan las señales de entrada podríamos obtener dicho caso obteniendo un valor "Q=1" a la salida. Estos cambios fueron realizados, y se comprobó la funcionalidad por completo de la celda OR. Además, a partir del

valor de la señal complementaria con “A=0, B=0” dando lugar a “QC=1” podemos deducir que el funcionamiento de la celda es correcto.

3. Diseño de bloques criptográficos Sbox..

El objetivo fundamental de este trabajo es comparar distintas implementaciones que se utilicen en aplicaciones criptográficas, a partir de medidas indirectas sobre el consumo de potencia. En este contexto, las estrategias de encriptación pasan siempre por la implementación de cifradores de bloque. Estos cifradores de bloque son dispositivos diseñados con la finalidad de extraviar la relación existente entre el texto que se quiere cifrar y el cifrado, con lo cual se hacen necesarios en cualquier tipo de aplicación criptográfica donde se quiera preservar secretamente cierta información. Generalmente, estos cifradores de bloque ejecutan el algoritmo criptográfico en distintas rondas de operación para un mayor camuflaje del texto cifrado. Como ejemplo, en la Fig 3.1 se muestra un diagrama en el que se detalla el proceso de encriptación seguido por el conocido cifrador de bloque AES-Rijndael [15]. Este cifrador de bloque fue elegido por el gobierno de los Estados Unidos, a través de su Instituto Nacional de Estándares y Tecnología (NIST), en noviembre de 2001 como un estándar para la encriptación y clasificación de información secreta por parte de la Agencia Nacional de Seguridad de los Estados Unidos (NSA), empezándose a utilizar a partir de junio de 2003.

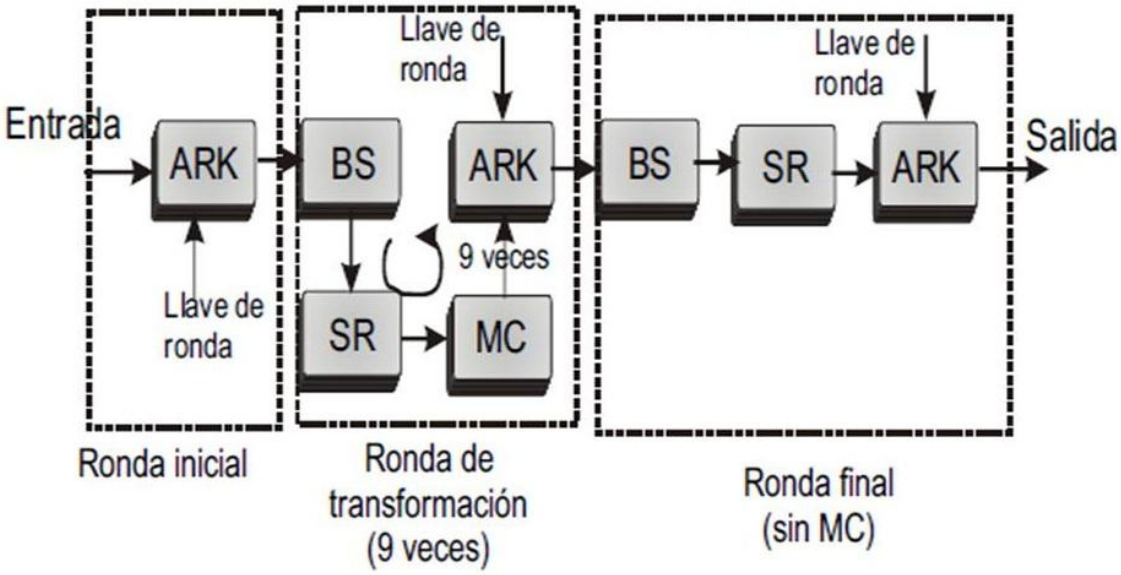


Fig 3.1. Diagrama ilustrativo de las rondas de operación del cifrador de bloque AES-Rijndael.

En este diagrama de flujo podemos ver como el texto plano a la entrada se intenta ocultar a través de diferentes mecanismos: Los bloques ARK (*Add Round Key)* aplican una operación lógica XOR entre clave y texto a cifrar, los SR (*ShiftRow)* producen un desplazamiento de filas en el texto, mientras que los MC (*MixColumns)* producen multiplicaciones entre las

columnas del texto y los BS (*SubByte)* realizan una sustitución no lineal donde cada byte es reemplazado con otro de acuerdo a una tabla de verdad. AES trabaja con bloques de 128 bits, y con claves de 128, 192 y 256 bits, dependiendo del tamaño de la clave el algoritmo puede ejecutarse en 10, 12 o 14 rondas de operación respectivamente [15]. El ejemplo ha sido desarrollado para el cifrador de bloque AES por ser uno de los más reconocidos y utilizados en la actualidad, pero el procedimiento es análogo en cualquier cifrador de bloque.

Sin embargo, dentro del bloque BS se encuentra un punto fundamental para la seguridad de cualquier dispositivo criptográfico: son las S-box (*Substitution box*). Este componente es clave en los algoritmos de cifrado que pretenden perder la relación existente entre texto y clave, por lo que el diseño de estas S-box deberá ser resistente a los diferentes ataques que puedan ejecutarse con objeto de revelar la clave secreta. Por tanto, vamos a centrar nuestras comparaciones en las diferentes S-boxes que implementan distintos cifradores de bloque tratando de hallar diferencias en nuestras medidas que nos permita determinar cuál de ellos es más seguro [2].

Existen muchas referencias en la literatura sobre cifradores de bloque [15], [16]. La tendencia actual a reducir el coste y el consumo de los circuitos criptográficos para su inclusión en dispositivos portables (smartphones, smartcards, etc) está desembocando en lo que se denomina “lightweight cryptography”, es decir, algoritmos que sean seguros pero con bajo coste. En concreto, nosotros vamos a centrarnos en aquellos algoritmos que incluyen S-boxes de 4 bits (Sbox-4) por ser aquellas que últimamente implementan algunos cifradores de bloque lightweight como TWINE [17], Midori [7], Piccolo [18] o PRIDE [19], al tener un diseño optimizado en recursos como el consumo de potencia o el área.

En este trabajo, vamos a comparar las S-boxes que implementan los cifradores Piccolo y PRIDE con sus diferentes ecuaciones lógicas y tablas de verdad. Para ello, diseñaremos las S-boxes en el entorno de Cadence, empleando las celdas SABL diseñadas; comprobaremos, en primer lugar, a partir de un análisis transitorio corto, la funcionalidad de estas S-boxes, y posteriormente a través de un código “checker” desarrollado en MatLab automatizaremos el proceso de comprobación de la funcionalidad para trazas más largas, que procesaremos, tal y como se muestra en el capítulo siguiente, para hacer medidas sobre su seguridad. Por último, generaremos una S-box de 8 bits (Sbox-8) formada por dos Sbox-4 diseñadas según las

ecuaciones lógicas de Piccolo, repetiremos el proceso y compararemos con las anteriores Sbox-4.

3.1. Sbox-4 Piccolo.

Piccolo [18] es un cifrador de bloque que produce una salida de 64 bits con una entrada de 64 bits, soportando claves de 80 bits y 128 bits. El algoritmo se ejecuta en 25 rondas de operación en el primer caso, y en 31 en el segundo. Sin embargo, como hemos introducido, nosotros nos centraremos únicamente en las Sbox que implementa. En concreto, este cifrador de bloque utiliza S-boxes de 4 bits de entrada y salida, dadas por las siguientes ecuaciones lógicas, que implementaremos con las celdas SABL que en el anterior capítulo presentábamos.

$$\begin{aligned} A &= d\ xor\ (a\ nor\ b) \\ B &= a\ xor\ (b\ nor\ c) \\ C &= b\ xnor\ (A\ nor\ c) \\ D &= c\ xor\ (B\ nor\ A) \end{aligned} \qquad (2)$$

Donde utilizamos las letras minúsculas para definir los bits de entrada, y letras mayúsculas para los bits de salida. En ambos casos, la letra "a" representa el bit más significativo, y "d" el menos significativo.

Desde el punto de vista criptográfico, esta S-box será representada por un bloque que reemplaza un cierto texto plano de entrada por un texto cifrado a la salida conforme a una tabla de verdad. La tabla de verdad para las ecuaciones lógicas que implementa Piccolo, en formato hexadecimal, es la siguiente:

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
y	e	4	b	2	3	8	0	9	1	a	7	f	6	c	5	d

Tabla 3.1. Tabla de verdad para Sbox-4 Piccolo.

Es decir que usando las ecuaciones lógicas que hemos utilizado, si tenemos por ejemplo el valor "d" a la entrada, vamos a tener:

$$d_{16} = 13_{10} = 1101_2 \rightarrow a = 1, b = 1, c = 0, d = 1.$$

Sustituyendo los valores de "x", por los de "y" según la lógica que hemos presentado unas líneas más arriba, la salida será:

$$A = 1\ xor\ (1\ nor\ 1) = 1$$
$$B = 1\ xor\ (1\ nor\ 0) = 1$$
$$C = 1\ xnor\ (1\ nor\ 0) = 0$$
$$D = 0\ xor\ (1\ nor\ 1) = 0$$

$$A = 1, B = 1, C = 0, D = 0 \rightarrow 1100_2 = 12_{10} = c_{16}$$

En este caso, como podemos comprobar solo vamos a necesitar 4 puertas lógicas XOR/XNOR y 4 OR/NOR, sin tener que implementar puertas AND/NAND. En la figura 3.2, mostramos un esquema a nivel de puertas lógicas de la Sbox4-Piccolo.

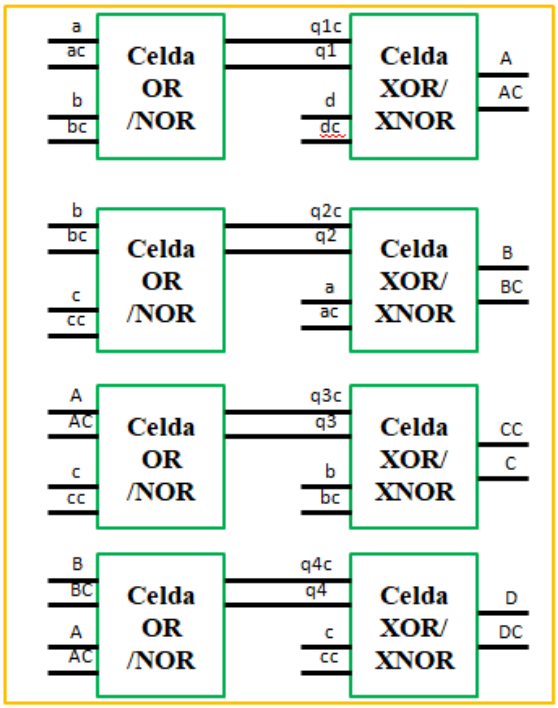


Fig 3.2. Esquema a nivel de puertas lógicas para la Sbox-4 Piccolo.

Obsérvese que al estar utilizando una lógica de doble raíl, introducimos las entradas por el raíl superior y sus complementarias por el raíl inferior. Cuando las operaciones lógicas sean XOR u OR, las salidas se situarán igualmente por el raíl superior y sus complementarias, por el inferior. Por el contrario, cuando las operaciones lógicas sean XNOR o NOR, las salidas se sitúan por el raíl inferior y sus complementarias por el superior, dado que el resultado de estas operaciones es equivalentes a realizar una XOR u OR en cada caso, y después invertir el resultado. De este modo, por ejemplo, la salida C que se encuentra tras una puerta lógica que realiza la operación XNOR irá por el raíl inferior mientras que la complementaria irá por el superior.

Un aspecto importante a la hora de evaluar tecnológicamente estas Sboxes es el número de transistores que incluyen cada una de ellas, ya que un mayor número de transistores generalmente implicará un mayor consumo de potencia. Como puede comprobarse a partir de las Figs 2.6, 2.7 y 2.8, tanto las celdas AND/NAND como las XOR/XNOR y las OR/NOR en tecnología SABL constan de 18 transistores cada una. Dado que la Sbox-4 de Piccolo cuenta con 4 puertas XOR/XNOR y otras 4 AND/NAND, dicha Sbox-4 tendrá un total de 144 transistores.

El bloque generado en Cadence para la Sbox-4 puede verse en la figura 3.3, en él se pueden ver los pines dispuestos para sus entradas y salidas, así como sus complementarias, tal y como corresponde a la lógica de doble raíl, y la señal de reloj. En la figura 3.4, hemos descendido en la jerarquía y podemos ver los bloques de las celdas lógicas que corresponden a las ecuaciones que presenta en sus S-box este cifrador de bloque, todos ellos con estilo lógico SABL.

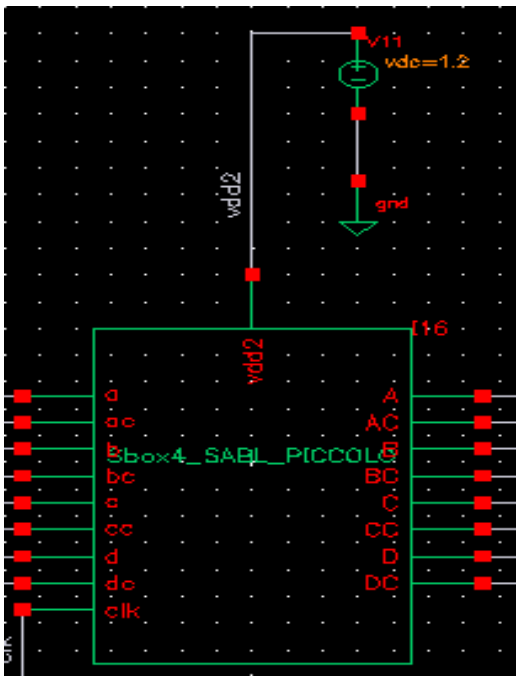


Fig 3.3. Bloque de la celda Sbox-4 Piccolo-SABL en Cadence.

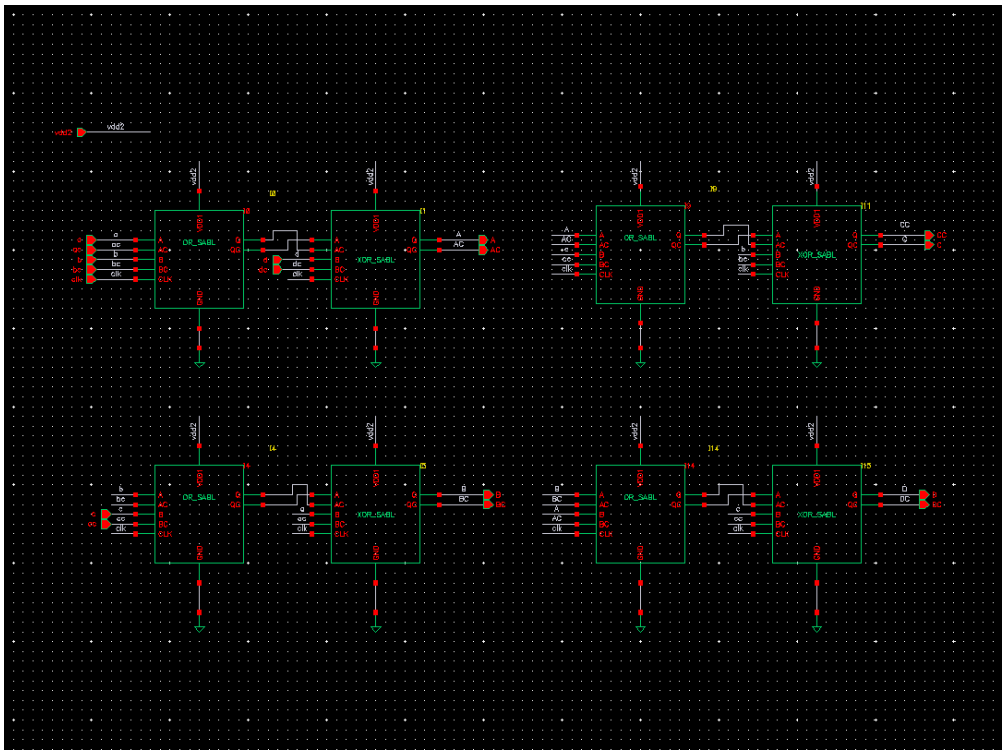


Fig 3.4. Esquemático a nivel de puertas lógicas para la Sbox-4 Piccolo-SABL en Cadence.

Una vez tenemos el diseño del bloque de la Sbox-4 habiendo utilizado las celdas lógicas que mostramos en el capítulo anterior y combinándolas según las ecuaciones que nos proporcionan los desarrolladores de Piccolo en [18], tenemos que comprobar la funcionalidad de dicha Sbox, generando un esquemático de test.

Los 4 patrones de entrada se producen aleatoriamente con un "random generator" de señales cuadradas que fue desarrollado por parte del grupo de investigación que dirige este proyecto. Dichos patrones de entrada, a los que llamaremos "D" se cruzan a través de un conjunto de 4 puertas XOR con la clave "K", dando como resultado "X" que será la entrada a la Sbox, por último, a la salida de la Sbox tendremos "Y". Cabe recalcar que la alimentación de la Sbox es independiente del resto de este esquemático de test, ya que a la hora de hacer medidas sobre la potencia sólo nos interesa el consumo de potencia producido por la Sbox en cuestión. El gráfico siguiente muestra de manera ilustrativa el proceso a seguir en este esquemático de test.

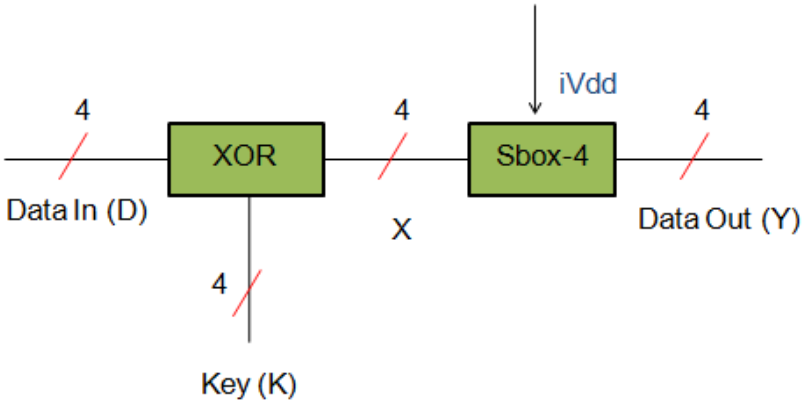


Fig 3.5. Diagrama de bloques del esquemático de test.

El esquemático completo se muestra en la figura 3.6. En la parte izquierda de la imagen, se muestra la generación de los bits "D" (recuadro azul) a través de un generador aleatorio desarrollado en Verilog-A por parte del grupo de investigación que dirige este proyecto, de forma que se puedan aplicar los mismos patrones de forma externa a las Sbox y cuidando que las simulaciones puedan ser reproducibles, siendo las mismas cada vez que se corre la simulación. A su derecha, observamos la generación de los bits de la clave K (recuadro rojo) cuyos valores se fijan en el mismo entorno de simulación. En el centro de la imagen tenemos el cruce a través de una puerta XOR de cada bit de entrada generado con cada bit de la clave (recuadro amarillo), tal y como indicábamos en el gráfico y por último, a su derecha encontramos la Sbox-4 Piccolo (recuadro naranja).

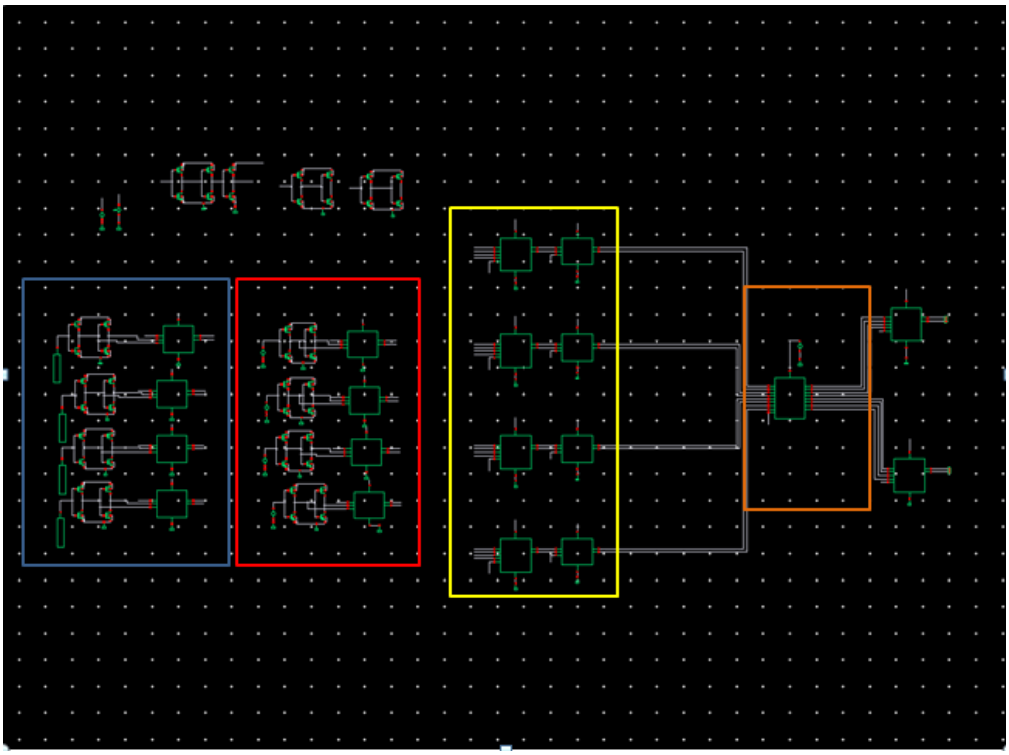


Fig 3.6. Esquemático de Test completo.

A continuación, realizamos un análisis transitorio corto utilizando Cadence (40 ns) que nos permita comprobar, a simple vista, si el bloque que acabamos de diseñar está operando correctamente. Además, la visualización de estos resultados en forma de gráfica nos permitirá notar algunos aspectos importantes que deben cumplirse en la caracterización de nuestro bloque. Las trazas obtenidas en este análisis han sido las siguientes:

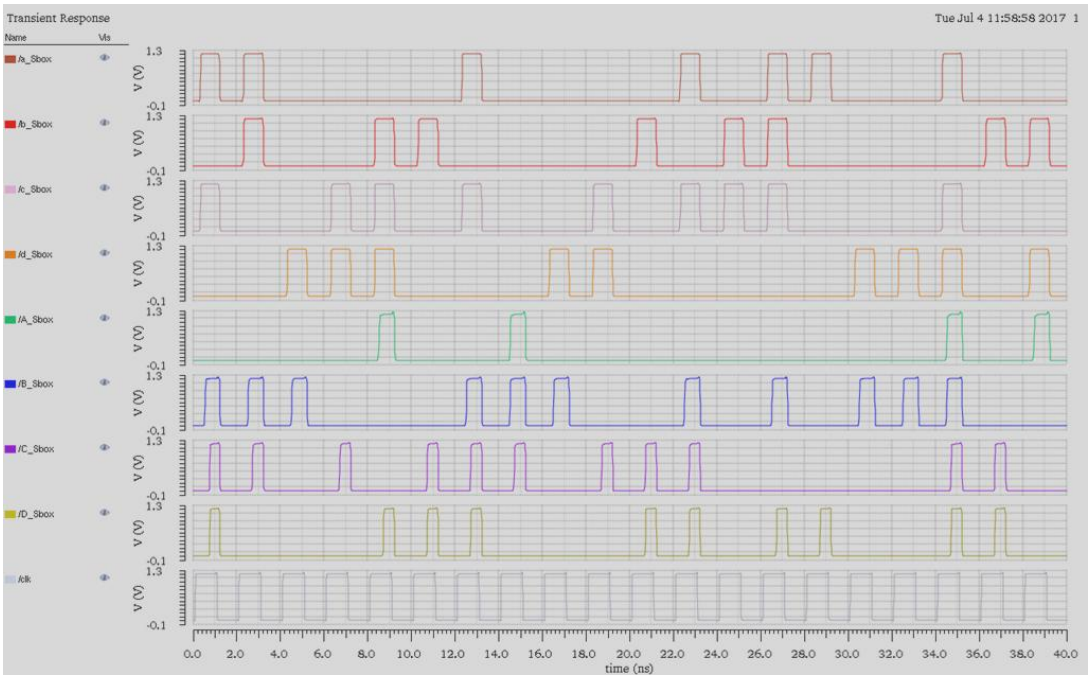


Fig 3.7. Resultado del análisis transitorio de 40 ns para la Sbox-4 Piccolo.

Donde las cuatros primeras formas de onda representan las entradas a la S-box, que representábamos en el gráfico anterior como X (a, b, c y d); las cuatro siguientes, las salidas de la S-box (A,B,C,D), que representábamos como Y, y por último la señal de reloj que marca las fases de precarga y evaluación yendo todas las señales al valor lógico "0" cuando el clk se encuentra a "0" y evaluándose cuando "clk=1". También, podemos observar como la salida conlleva un retraso con respecto a la señal de reloj, que se hace especialmente notable en las salidas "C" y "D" ya que éstas dependen a su vez de las salidas "A" y "B". Esta gráfica sirvió para comprobar si el diseño implementado a través de las ecuaciones lógicas proporcionadas coincidía con la tabla de verdad que proporciona Piccolo, o si por el contrario, existía algún fallo en el diseño de nuestros bloques. Así, tomando por ejemplo el segundo y el séptimo patrón podemos comprobar este aspecto. Para el segundo patrón:

$$a = 1, b = 1, c = 0, d = 0 \rightarrow X = 1100_2 = 12_{10} = C_{16}$$

Utilizando las ecuaciones lógicas:

$$A = 0\ xor\ (1\ nor\ 1) = 0$$
$$B = 1\ xor\ (1\ nor\ 0) = 1$$
$$C = 1\ xnor\ (0\ nor\ 0) = 1$$
$$D = 0\ xor\ (1\ nor\ 0) = 0$$

$$A = 0, B = 1, C = 1, D = 0 \rightarrow Y = 0110_2 = 6_{10} = 6_{16}$$

Resultado coincidente con lo que establece la gráfica así como con la tabla de verdad. Para el séptimo patrón:

$$a = 1, b = 0, c = 1, d = 0 \rightarrow X = 1010_2 = 10_{10} = A_{16}$$

$$A = 0\ xor\ (1\ nor\ 0) = 0$$
$$B = 1\ xor\ (0\ nor\ 1) = 1$$
$$C = 0\ xnor\ (0\ nor\ 1) = 1$$
$$D = 1\ xor\ (1\ nor\ 0) = 1$$

$$A = 0, B = 1, C = 1, D = 1 \rightarrow Y = 0111_2 = 7_{10} = 7_{16}$$

Coincidiendo una vez más lo observado en la gráfica con lo previsto por la tabla de verdad y las ecuaciones lógicas. Esto puede comprobarse para cada uno de los patrones presentes en la gráfica obteniéndose un resultado correcto. No obstante, como queremos analizar un gran

número de patrones, del orden de varios miles, la tarea de comprobación de la funcionalidad de la S-box debe ser automatizada. Para ello, hemos generado un código en MATLAB que realizará esta tarea previa a la realización del análisis estadístico que realizamos para evaluar la seguridad, ya que es importante asegurarse del funcionamiento del bloque antes de hacer una comparación entre diferentes Sboxes, pues si estos bloques no están operando correctamente, no tiene ningún sentido intentar hacer una comparación de bloques con un funcionamiento erróneo. El código en cuestión se muestra en el anexo. En él, lo que hacemos fundamentalmente es cargar las trazas de entrada a la Sbox para una determinada clave, cortarlas por transiciones, hacer lo propio con las salidas y comparar por un lado la salida que tendría que resultar procesando las trazas de entrada a partir de sus ecuaciones lógicas, obteniendo el valor teórico al que llamamos OUT_Soft, con el valor que hemos obtenido de la traza en Cadence. Ambas "salidas" las almacenamos en matrices 4 x n (en binario), donde n es el número de patrones, que pasamos a sistema hexadecimal, quedando sendos vectores columna que comparamos. Si el resultado de esa comparación es verdadero = 1, el programa mostrará un mensaje corroborando que la funcionalidad es correcta. Si por el contrario, los valores experimentales y de "software" no se corresponden, el programa mostrará un mensaje de error. Este procedimiento queda detallado en el organigrama que presentamos en la Fig 3.8.

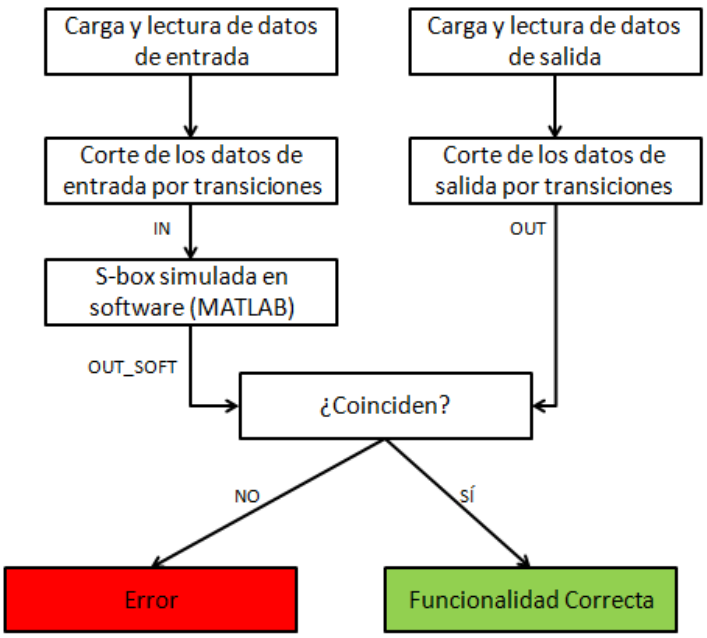


Fig 3.8. Diagrama de flujo del código "checker" utilizado para comprobar la funcionalidad de las S-Box.

Una vez hayamos conseguido que el código "checker" corrobore que la funcionalidad de la Sbox-4 Piccolo es correcta, podemos decir que el diseño de dicha Sbox ha sido terminado. El procedimiento de diseño que hemos seguido para desarrollar la Sbox de Piccolo es análogo en todos los casos, independientemente de las ecuaciones lógicas que cada Sbox implemente o de los bits de entrada y salida que tenga cada una de ellas, como veremos más adelante.

3.2. Sbox-4 PRIDE.

Otro de los diseños que hemos implementado para este trabajo ha sido aquel que se rige según las especificaciones y las ecuaciones lógicas que nos proporciona PRIDE [19]. PRIDE es otro cifrador de bloque de 64x64 bits implementando una clave de 128 bits, en este caso el algoritmo se ejecuta en 20 rondas de operación montando Sboxes-4 que se rigen por las siguientes ecuaciones lógicas:

$$\begin{aligned} A &= c\ xor\ (a\ and\ b) \\ B &= d\ xor\ (b\ and\ c) \\ C &= a\ xor\ (A\ and\ B) \\ D &= b\ xor\ (B\ and\ C) \end{aligned} \qquad (3)$$

Utilizando el mismo criterio que en el caso de Piccolo para letras mayúsculas y minúsculas y siendo, una vez más, "a" el bit más significativo y "d" el menos significativo dando lugar a la siguiente tabla de verdad.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
y	0	4	8	f	1	5	e	9	2	7	a	c	b	d	6	3

Tabla 3.2. Tabla de verdad para Sbox-4 PRIDE.

Por tanto, y con el mismo ejemplo que en el caso anterior, si tenemos como valor de entrada "d" y utilizando las ecuaciones (3), se habrá de tener a la salida:

$$d = 13_{10} = 1101_2 \rightarrow a = 1, b = 1, c = 0, d = 1.$$

$$\begin{aligned} A &= 0\ xor\ (1\ and\ 1) = 1 \\ B &= 1\ xor\ (1\ and\ 0) = 1 \\ C &= 1\ xor\ (1\ and\ 1) = 0 \\ D &= 1\ xor\ (1\ and\ 0) = 1 \end{aligned}$$

$$A = 1, B = 1, C = 0, D = 1 \rightarrow 1101_2 = 13_{10} = d_{16}$$

Habiendo utilizado en esta ocasión únicamente una puerta XOR/XNOR y una AND/NAND para realizar cada una de las dos operaciones que cifran cada bit. Por tanto, la Sbox-4 de PRIDE constará, al igual que la presentada por Piccolo, 144 transistores ya que cada celda independientemente de la función que implemente consta de 18 transistores en tecnología SABL.

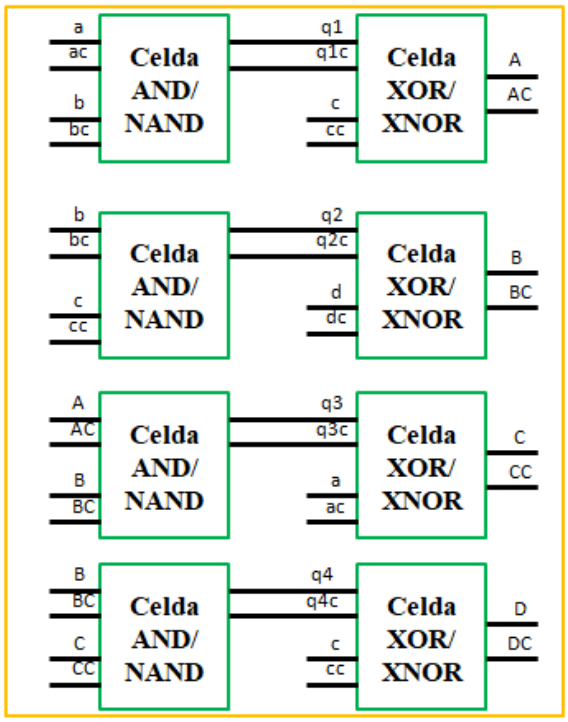


Fig 3.9. Esquema a nivel de puertas lógicas para la Sbox-4 PRIDE.

Análogamente al caso de Piccolo, hemos implementado en Cadence un bloque que realiza las funciones de la S-box con las ecuaciones lógicas proporcionadas por PRIDE, del mismo modo, descendiendo en la jerarquía, podemos ver los bloques que conforman la S-box formados por las celdas estilo SABL que se han definido anteriormente.

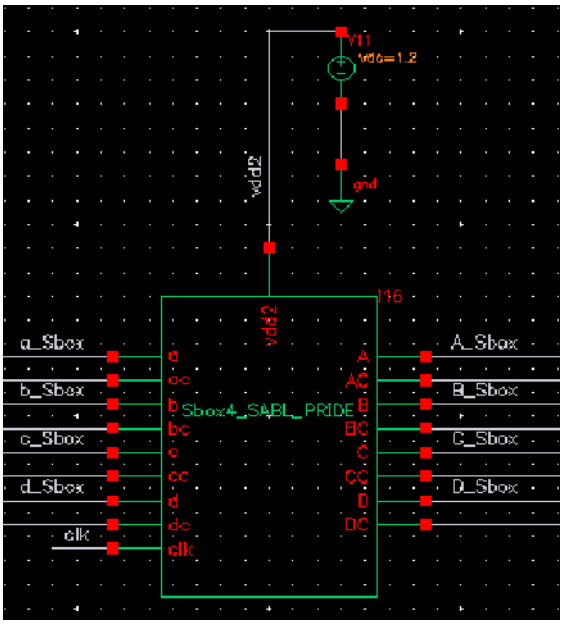


Fig 3.10. Bloque de la celda Sbox-4 PRIDE en Cadence.

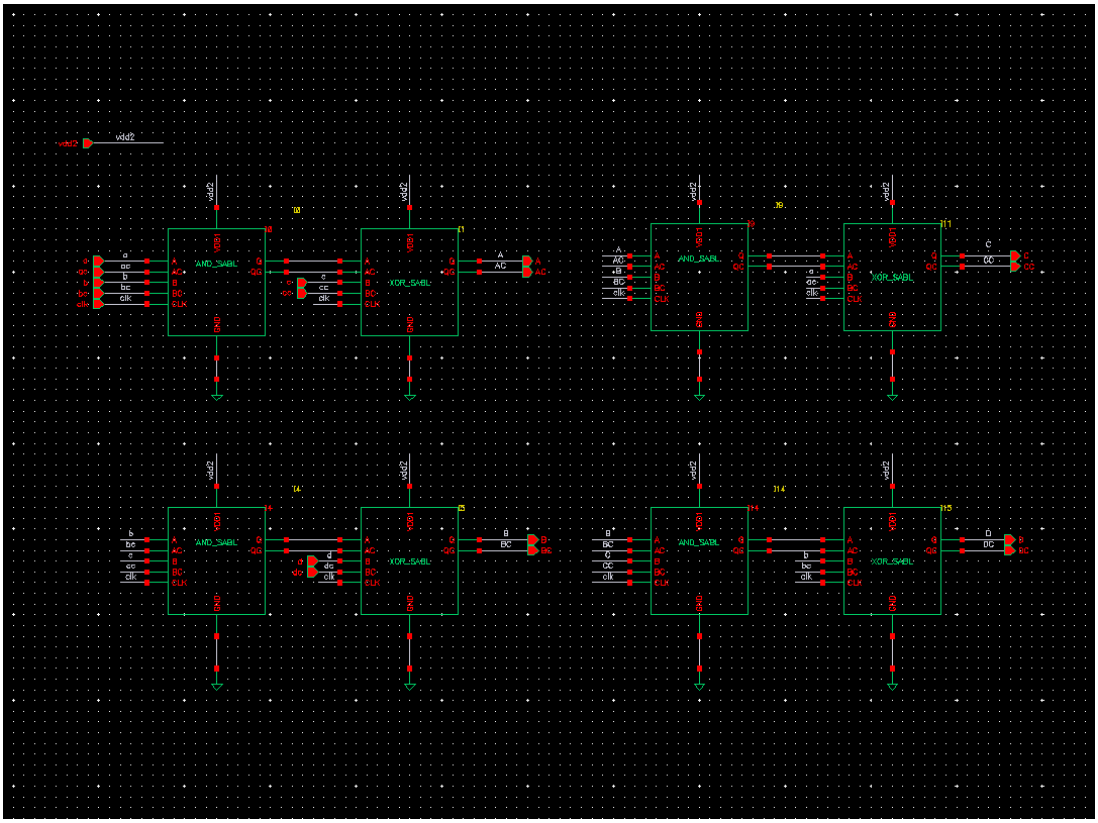


Fig 3.11. Esquemático a nivel de puertas lógicas para la Sbox-4 PRIDE-SABL en Cadence.

El siguiente paso en nuestro procedimiento, como ya hicimos en el caso de Piccolo es generar un esquemático de test a través del cual podamos comprobar la funcionalidad de la S-Box, en este caso según las especificaciones de PRIDE. El esquemático de test para este caso es completamente análogo al caso de Piccolo, y tanto el esquemático presentado en la Fig 3.6. como el diagrama de bloques que presentábamos en la Fig 3.5, así como todo lo comentado

en referencia al esquemático de test puede ser extrapolado a este nuevo caso. Por tanto, volvemos a realizar un análisis corto de 40 ns, en el que se van a observar 20 patrones distintos ya que la frecuencia con la que estamos trabajando a nivel de S-boxes es 500 MHz, luego tendremos un periodo de 2 ns que nos permitirá visualizar esos 20 patrones. La figura siguiente muestra el resultado de dicho análisis transitorio que nos permite comprobar la funcionalidad de la S-Box.

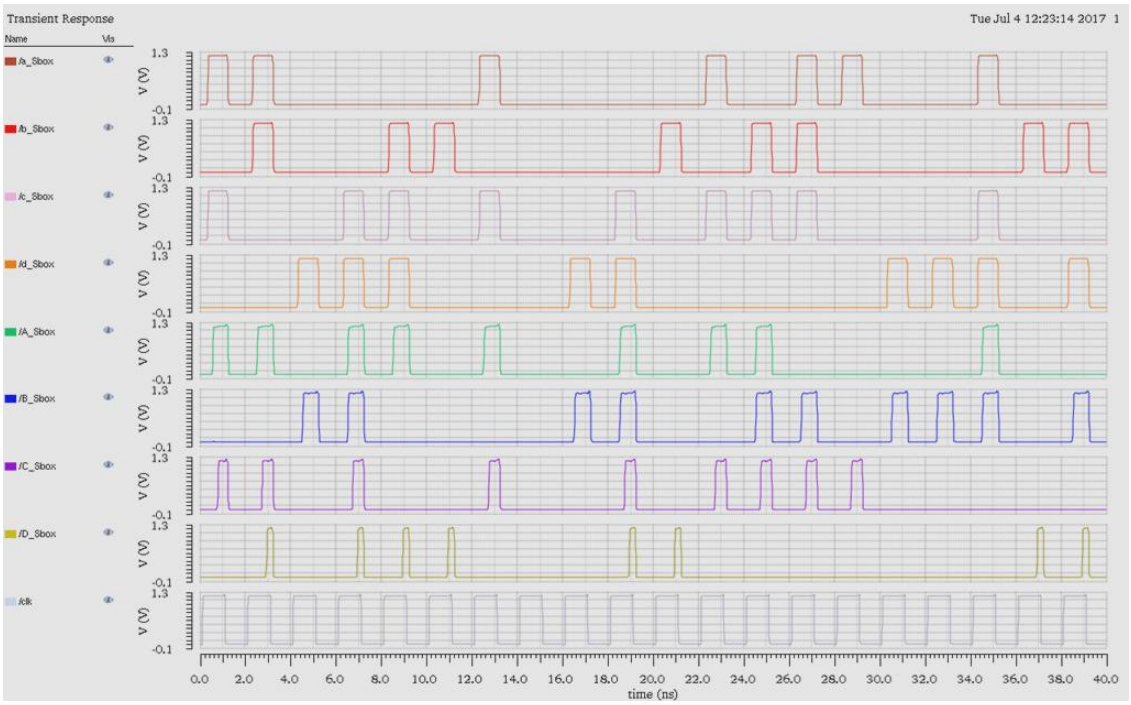


Fig 3.12. Resultado del análisis transitorio de 40 ns para la Sbox4-PRIDE.

Volviéndose a comprobar el estilo de precarga y evaluación, el retraso producido por la configuración en bucle de las dos últimas salidas, teniendo ahora incluso un retraso mayor las salidas “D” y “C”, debido a que en este caso la salida “D” depende también de la “C” y, a su vez, la “C” depende de las dos salidas anteriores (“A” y “B”) y no solo de una como en el caso anterior. Cabe señalar que las formas de onda están ordenadas del mismo modo que en el caso de Piccolo. Si, del mismo modo que unas páginas atrás, tomamos, por ejemplo, el segundo y el séptimo patrón podemos comprobar la funcionalidad de la S-Box. Los resultados pueden extrapolarse y probarse con los otros 18 patrones, obteniéndose un resultado favorable. No obstante, haciendo ciertos cambios en el código “checker” que implementábamos en MatLab y que presentamos con anterioridad, podremos comprobar la funcionalidad para un número indefinido y generalizado de patrones. Asimismo, se observa en la figura que los patrones de entrada son exactamente iguales que para la anterior S-Box. Esto se debe al generador aleatorio que hemos utilizado para producir los patrones de entrada,

dado que, en función de una semilla dada, cada bit se generará de forma distinta para cada semilla pero los patrones de dicho bit serán los mismos si se repite el análisis con la misma semilla un número indefinido de veces. Como el esquemático de test es el mismo, lo único que hemos cambiado ha sido el bloque de la S-box sin modificar la semilla de los generadores ni cualquier otro elemento del esquemático, los patrones de entrada observados van a ser los mismos. Por tanto, observando el segundo patrón para comprobar la funcionalidad tendremos:

$$a = 1, b = 1, c = 0, d = 0 \rightarrow X = 1100_2 = 12_{10} = C_{16}$$

Y a la salida tenemos:

$$A = 1, B = 0, C = 1, D = 1 \rightarrow Y = 1011_2 = 11_{10} = B_{16}$$

Y además siguiendo las ecuaciones lógicas definidas en (3), se corrobora la relación entre tabla de verdad, ecuaciones y lo implementado en Cadence ya que:

$$A = 0\ xor\ (1\ and\ 1) = 1$$
$$B = 0\ xor\ (1\ and\ 0) = 0$$
$$C = 1\ xor\ (1\ and\ 0) = 1$$
$$D = 1\ xor\ (0\ and\ 1) = 1$$

Para el séptimo patrón obtendremos:

$$a = 1, b = 0, c = 1, d = 0 \rightarrow X = 1010_2 = 10_{10} = A_{16}$$

$$A = 1\ xor\ (1\ and\ 0) = 1$$
$$B = 0\ xor\ (0\ and\ 1) = 0$$
$$C = 1\ xor\ (1\ and\ 0) = 1$$
$$D = 0\ xor\ (0\ and\ 1) = 0$$

$$A = 1, B = 0, C = 1, D = 0 \rightarrow Y = 1010_2 = 10_{10} = A_{16}$$

Y para cerrar esta sección, tendremos que comprobar la funcionalidad de manera automatizada de cara a la evaluación de la seguridad que haremos en el próximo capítulo. A partir del código checker que hemos presentado anteriormente, será, por tanto sencillo presentar otro similar para las especificaciones de PRIDE. Manteniendo el código anterior tal y como fue explicado y cambiando las líneas de código que conciernen a las ecuaciones lógicas, podremos automatizar el proceso de comprobación de la funcionalidad.

3.3. Sbox-8 Piccolo.

Por último, hemos implementado una Sbox de 8 bits formada por dos Sbox de 4 bits según las especificaciones de Piccolo, de manera que a partir de los resultados obtenidos por medio de la posterior comparación podremos esbozar si un mayor número de bits implica mayor seguridad, o si por el contrario, no existe una relación clara entre el número de bits y la seguridad, desechando, entonces, la inversión en consumo de potencia y área que implicaría el hacer una Sbox con el doble de bits. Las ecuaciones lógicas de esta Sbox serán, por tanto, equivalentes a situar dos Sbox-4 paralelamente contenidas en el mismo bloque, tal y como se indica en la Fig 3.13. En este caso, al tener dos Sbox-4 colocadas en paralelo vamos a tener 8 celdas OR/NOR y 8 XOR/XNOR, y por tanto, esta Sbox constará de 288 transistores en total, el doble que su análoga de 4 bits.

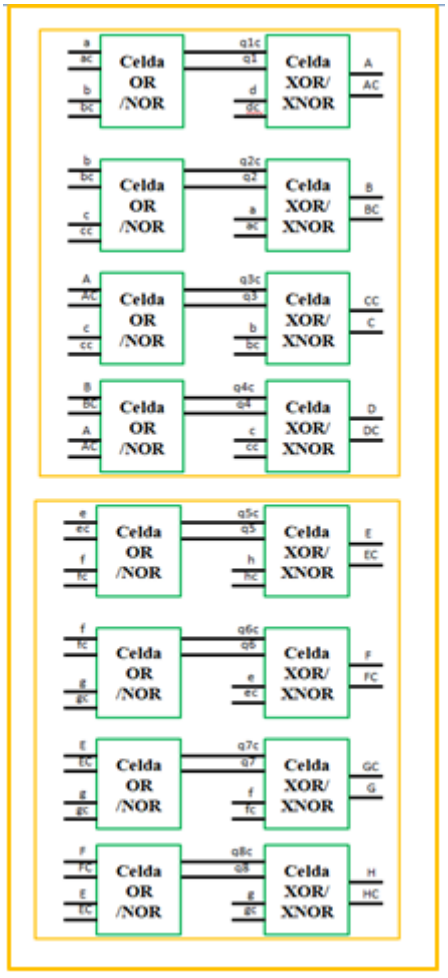


Fig 3.13. Esquema a nivel de puertas lógicas para la Sbox-8 Piccolo.

Luego:

$$
\begin{aligned}
A &= d\ xor\ (a\ nor\ b)\\
B &= a\ xor\ (b\ nor\ c)\\
C &= b\ xnor\ (A\ nor\ c)\\
D &= c\ xor\ (B\ nor\ A)\\
E &= h\ xor\ (e\ nor\ f)\\
F &= e\ xor\ (f\ nor\ g)\\
G &= f\ xnor\ (E\ nor\ g)\\
H &= g\ xor\ (F\ nor\ E)
\end{aligned}
\qquad (4)
$$

En esta ocasión, la tabla de verdad será algo más complicada dada la multitud de posibilidades que ahora se presentan. Hemos aumentado de 4 bits a 8, lo cual hace aumentar las distintas posibles configuraciones de 16 a 256. Por ello, en esta ocasión vamos a utilizar una tabla decimal de doble entrada que, mediante la fórmula que se presenta más abajo, nos permitirá encontrar de una manera sencilla la equivalencia entre el texto a la entrada y a la salida. Hemos utilizado el sistema decimal en este caso, porque es mucho más eficiente a la hora de hacer operaciones aritméticas que nos permiten la descomposición de los datos de entrada según la tabla de verdad para obtener los datos de salida, pero igualmente estos valores podríamos pasarlos a sistema hexadecimal si lo que se busca es encontrar una coherencia con el sistema utilizado en el caso de las Sbox-4. La fórmula a seguir es:

$$X = 16 \cdot nº\ fila + nº\ columna.$$

Descomponiendo el valor, en decimal, que tenemos a la entrada de esta manera y buscando el valor que se le atribuye en la siguiente tabla, es posible conocer el valor Y que le corresponde a la salida. La tabla de verdad es la siguiente:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	238	228	235	226	227	232	224	233	225	234	231	239	230	236	229	237
1	78	68	75	66	67	72	64	73	65	74	71	79	70	76	69	77
2	190	180	187	178	179	184	176	185	177	186	183	191	182	188	181	189
3	46	36	43	34	35	40	32	41	33	42	39	47	38	44	37	45
4	62	52	59	50	51	56	48	57	49	58	55	63	54	60	53	61
5	142	132	139	130	131	136	128	137	129	138	135	143	134	140	133	141
6	14	4	11	2	3	8	0	9	1	10	7	15	6	12	5	13
7	158	148	155	146	147	152	144	153	145	154	151	159	150	156	149	157
8	30	20	27	18	19	24	16	25	17	26	23	31	22	28	21	29
9	174	164	171	162	163	168	160	169	161	170	167	175	166	172	165	173
10	126	116	123	114	115	120	112	121	113	122	119	127	118	124	117	125
11	254	244	251	242	243	248	240	249	241	250	247	255	246	252	245	253
12	110	100	107	98	99	104	96	105	97	106	103	111	102	108	101	109
13	206	196	203	194	195	200	192	201	193	202	199	207	198	204	197	205
14	94	84	91	82	83	88	80	89	81	90	87	95	86	92	85	93
15	222	212	219	210	211	216	208	217	209	218	215	223	214	220	213	221

Tabla 3.3. Tabla de Verdad para Sbox-8 Piccolo.

Por ejemplo, si tenemos el valor X= 92 a la entrada, usando la tabla decimal vamos a tener a la salida que Y = 134, es decir, tenemos la posición que vemos remarcada.

Y utilizando las ecuaciones lógicas anteriormente detalladas en (4) podemos, una vez más, comprobar la coherencia entre lo establecido por la tabla de verdad y lo predicho por las ecuaciones lógicas para el ejemplo detallado:

$$92_{10} = 01011100_2 \rightarrow a = 0, b = 1, c = 0, d = 1, e = 1, f = 1, g = 0, h = 0$$

$$A = 1\ xor\ (0\ nor\ 1) = 1$$
$$B = 0\ xor\ (1\ nor\ 0) = 0$$
$$C = 1\ xnor\ (1\ nor\ 0) = 0$$
$$D = 0\ xor\ (0\ nor\ 1) = 0$$
$$E = 0\ xor\ (1\ nor\ 1) = 0$$
$$F = 1\ xor\ (1\ nor\ 0) = 1$$
$$G = 1\ xnor\ (0\ nor\ 0) = 1$$
$$H = 0\ xor\ (1\ nor\ 0) = 0$$

$$A = 1, B = 0, C = 0, D = 0, E = 0, F = 1, G = 1, H = 0 \rightarrow\ 10000110_2 = 134_{10}$$

Una vez hechas estas comprobaciones, nos encontramos en disposición de presentar el bloque que hemos diseñado en Cadence. Para la Sbox-8, nos encontraremos con un bloque con 16 pines de entrada y 16 pines de salida correspondiente a cada uno de los bits implementados más su complementario, manteniendo así la lógica de doble raíl, más el pin dedicado a la señal de reloj, y el dedicado a la alimentación.

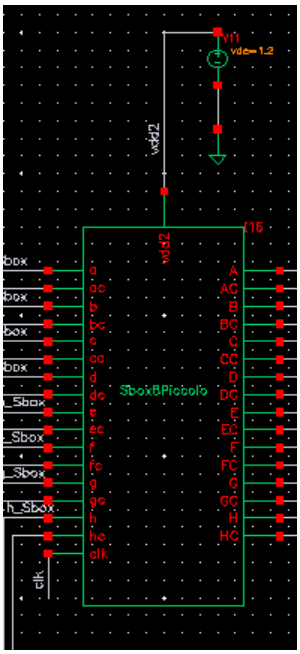


Fig 3.14. Bloque de la celda Sbox-8 Piccolo-SABL.

Descendiendo en la jerarquía, se puede observar que el diseño realizado está formado únicamente por dos Sbox-4 colocadas en paralelo. En esta ocasión, por tanto, no seguiremos descendiendo en la jerarquía para mostrar el esquemático a nivel de puertas lógicas dado que este vendrá dado según las especificaciones de Piccolo, de manera completamente análoga a lo mostrado en el caso de la Sbox-4.

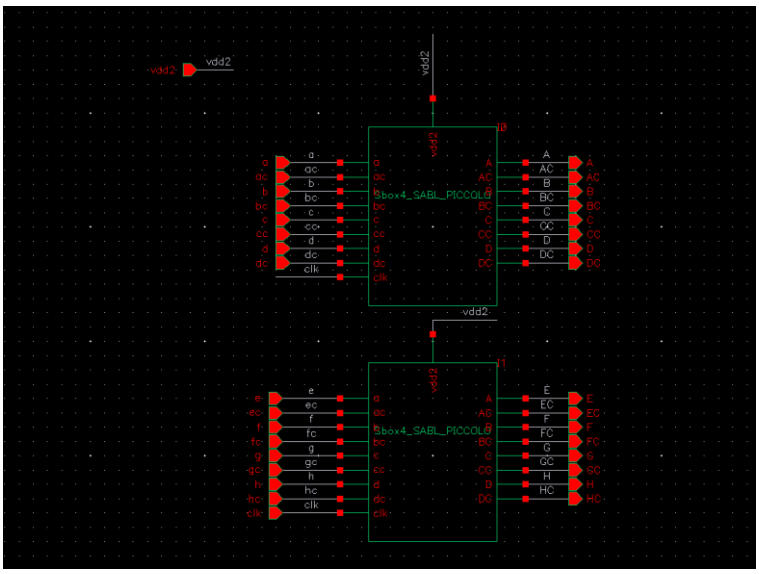


Fig 3.15. Segundo nivel de jerarquía del bloque Sbox-8 Piccolo-SABL.

Al estar tratando en este caso con 8 bits en lugar de 4, lo establecido para el esquemático de test que nos permite obtener las trazas para las posteriores comprobaciones de la funcionalidad y medidas de seguridad tendremos que extrapolarlo a este nuevo caso, de tal

modo que el esquemático sea completamente análogo al utilizado con 4 bits a fin de utilizar metódicamente un mismo procedimiento en los tres casos mostrados, pero permitiéndonos, a su vez, hacer los análisis buscados en el desarrollo de nuestro trabajo. La siguiente figura muestra, entonces, el diseño de un esquemático de test exactamente igual a lo explicado para las Sbox-4, al que le hemos añadido 4 bits tanto en la clave secreta, como en el texto plano de entrada. Asimismo, habremos de aumentar el número de operaciones lógicas necesarias para aislar la S-box a testear del resto de elementos del circuito, tales como las puertas XOR que introducíamos a la salida de dicho bloque. Volvemos a utilizar el recuadro azul para señalar el apartado donde se generan los textos planos aleatoriamente, el recuadro rojo para la generación de las claves que introducimos manualmente desde el Analog Design Environment de Cadence, siendo el recuadro amarillo el lugar donde texto plano y clave se cruzan y el recuadro naranja señalando la S-box a testear.

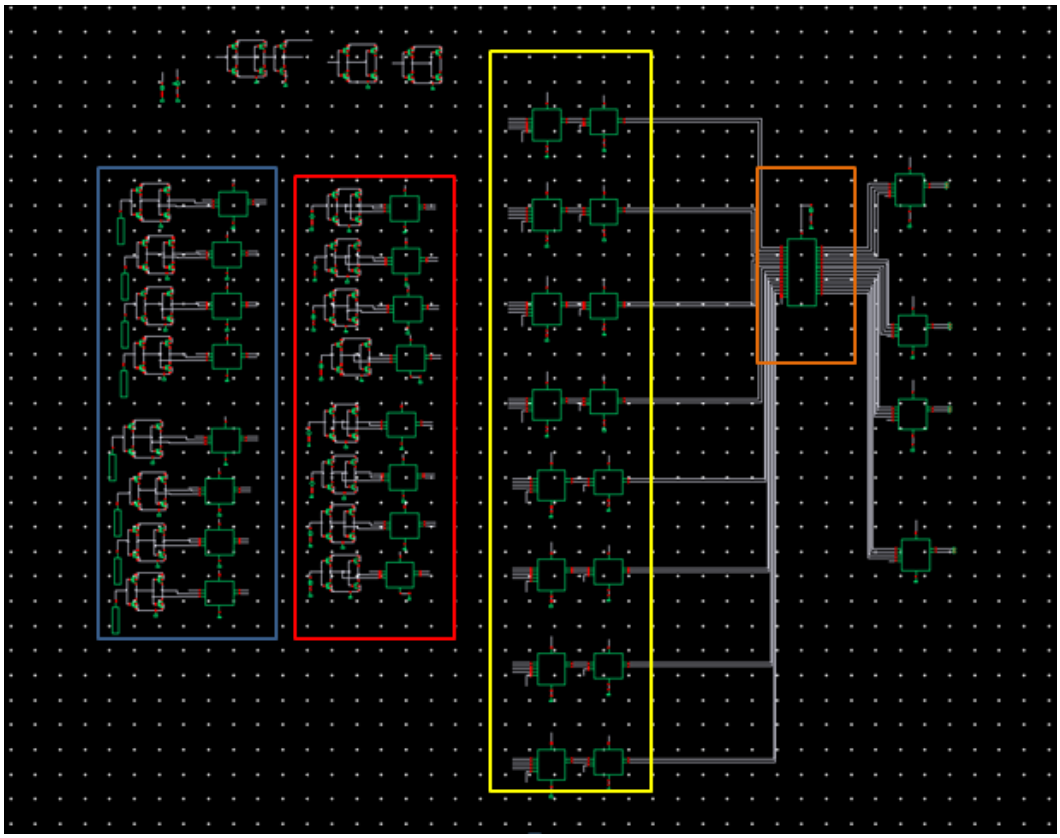


Fig 3.16. Esquemático de Test Completo para Sbox-8.

Y, teniendo listo este esquemático, al igual que hemos hecho para las Sbox de 4 bits, lanzamos un análisis transitorio corto con la finalidad de comprobar la funcionalidad de una manera visual. En esta ocasión el código “checker” tendrá considerables cambios, viéndose afectado por el hecho del aumento de bits. Mostramos, por tanto, los resultados del análisis con sus correspondientes comprobaciones de ejemplo para la Sbox-8 Piccolo.

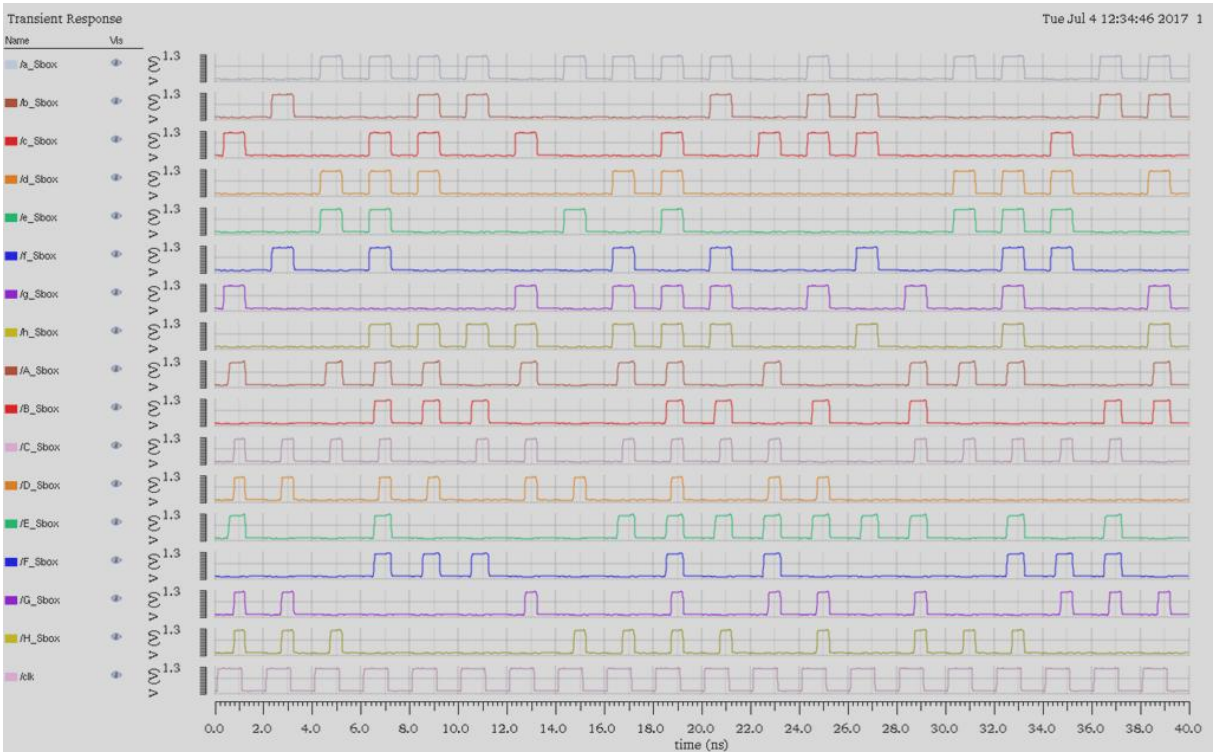


Fig 3.17. Resultado del análisis transitorio de 40 ns para la SBox-8 Piccolo.

Siendo las 8 primeras formas de onda correspondientes a las entradas ordenadas por orden alfabético (primero la "a", el bit más significativo, y por último la "h", el bit menos significativo), las 8 siguientes corresponden a la salida utilizando el mismo criterio y siendo la última la señal de reloj que marca las fases de precarga y evaluación. Comprobando para el segundo patrón, se tiene:

$$a = 0, b = 1, c = 0, d = 0, e = 0, f = 1, g = 0, h = 0 \rightarrow X = 01000100_2 = 68_{10}$$

Utilizando las ecuaciones lógicas:

$$A = 0\ xor\ (0\ nor\ 1) = 0$$
$$B = 0\ xor\ (1\ nor\ 0) = 0$$
$$C = 1\ xnor\ (0\ nor\ 0) = 1$$
$$D = 0\ xor\ (0\ nor\ 0) = 1$$
$$E = 0\ xor\ (0\ nor\ 1) = 0$$
$$F = 0\ xor\ (1\ nor\ 0) = 0$$
$$G = 1\ xnor\ (0\ nor\ 0) = 1$$
$$H = 0\ xor\ (0\ nor\ 0) = 1$$

$$A = 0, B = 0, C = 1, D = 1, E = 0, F = 0, G = 1, H = 1 \rightarrow\ 00110011_2 = 51_{10}$$

Para el séptimo patrón será:

$$a = 0, b = 0, c = 1, d = 0, e = 0, f = 0, g = 1, h = 1 \rightarrow X = 00100011_2 = 35_{10}$$

$$A = 0\ xor\ (0\ nor\ 0) = 1$$
$$B = 0\ xor\ (0\ nor\ 1) = 0$$
$$C = 0\ xnor\ (1\ nor\ 1) = 1$$
$$D = 1\ xor\ (0\ nor\ 1) = 1$$
$$E = 1\ xor\ (0\ nor\ 0) = 0$$
$$F = 0\ xor\ (0\ nor\ 1) = 0$$
$$G = 0\ xnor\ (0\ nor\ 1) = 1$$
$$H = 1\ xor\ (0\ nor\ 0) = 0$$

$$A = 1, B = 0, C = 1, D = 1, E = 0, F = 0, G = 1, H = 0 \rightarrow\ 10110010_2 = 178_{10}$$

Resultados que coinciden tanto con la tabla de verdad como con el resultado de las correspondientes salidas que pueden observarse en la figura donde presentábamos nuestras formas de onda.

Para cerrar este capítulo, tendremos que hacer cambios en el código "checker" que permite la automatización de la comprobación de la funcionalidad con los cambios para hacerlo aplicable a una S-box de 8 bits. En concreto, como ahora las posibilidades de claves aumentan desde 16 hasta 256 por el hecho de tener 8 bits, mostramos el código "checker" correspondiente a la que hemos llamado "key10" en el anexo. Esta clave en realidad corresponde al valor 11101111_2 y dicho código, como se puede comprobar, es extrapolable a cualquier otra clave que podamos imaginar.

4. Medidas para la evaluación de seguridad de las Sboxes.

En este capítulo, vamos a tratar de obtener información sobre la vulnerabilidad de las Sboxes diseñadas. Para ello, las medidas que se mostrarán a continuación nos permiten, mediante una simulación eléctrica corta, y con un procesado de datos relativamente sencillo determinar la robustez de las celdas que hemos diseñado así como comparar, a partir de diferentes parámetros, las diferencias existentes a nivel de seguridad de dichas celdas.

En concreto, el test consiste en recoger y almacenar las trazas de intensidad de polarización de cada una de las S-boxes, que mostrará variaciones a lo largo del tiempo, procesar esas trazas por transiciones, diferenciando entre precarga y evaluación, y una vez tengamos los datos procesados, analizaremos la seguridad de cada Sbox en función de los parámetros estadísticos que se presentan más abajo.

Para cada simulación eléctrica que le apliquemos a cada una de las Sboxes, vamos a medir los siguientes parámetros a partir de las trazas de la corriente de polarización:

- Mín: Valor mínimo de energía
- Máx: Valor máximo de energía.
- μ: Valor medio de energía
- σ: Desviación estándar de la energía.

Las medidas más utilizadas en la literatura que pueden dar una medida de la seguridad de una celda criptográfica son [4]:

- NED (Normalized Energy Deviation):

$$NED = \frac{Máx - Mín}{Máx} \qquad (5)$$

- NSD (Normalized Standard Deviation):

$$NSD = \frac{\sigma}{\mu} \qquad (6)$$

Cuanto menor sea el resultado de estas medidas, mayor será la seguridad de las celdas a las que se refieran. De este modo, una celda ideal completamente segura deberá tener NED = 0 y NSD = 0.

Además, también se mostrarán medidas de la potencia y energía consumida en cada fase de precarga y evaluación, como una medida importante a la hora de buscar un compromiso entre los recursos que consume cada Sbox y la seguridad que proporciona. Estas medidas van a venir dadas por las siguientes ecuaciones:

$$E = V_{DD} \cdot I_{AV} \cdot T_{CLK} \quad (7)$$

$$P = V_{DD} \cdot I_{AV} \qquad (8)$$

$$I_{AV} = \frac{1}{T_{CLK}} \int_{-T_{CLK}/2}^{T_{CLK}/2} i_{DD}(t)dt \qquad (9)$$

Donde V_{DD} será la tensión de alimentación de la S-Box que, como en las simulaciones anteriores para la comprobación de la funcionalidad, fijamos en 1,2 V. Mientras que el T_{CLK} será en todos los casos de 1 ns, el razonamiento es el siguiente: La frecuencia de las señales cuadradas que implementan los patrones de entrada es de 500 MHz, por tanto, con una simple cuenta podremos deducir que el período es de 2 ns, sin embargo, esos 2 ns están compuestos de una fase de precarga y otra de evaluación de igual duración; como nosotros estamos haciendo medidas del consumo de potencia y energía de forma separada para precarga y evaluación, el T_{CLK} será de 1 ns en cada caso.

Para hacer estas simulaciones, y poder analizar luego los resultados que se extraen de sus trazas, se han creado dos scripts, uno en Ocean Script y otro en MatLab. El primero de ellos nos permite lanzar en cadena cada simulación de forma automática, recogiendo los datos después de cada simulación almacenándolos en ficheros de texto plano que luego son interpretables en MatLab. Por su parte, el segundo script generado en MatLab carga el archivo previamente generado en Ocean y calcula automáticamente el NED, NSD y el consumo de potencia y energía de cada celda en precarga y evaluación. Esto nos permite obtener los resultados de simulación de una manera manejable, sin requerir demasiado esfuerzo y permitiéndonos cambiar las condiciones de simulación.

4.1. Resultados.

A continuación, se presentan los resultados obtenidos de las medidas que anteriormente comentábamos para las 3 celdas que hemos diseñado: Sbox-4 Piccolo, Sbox-4 PRIDE y Sbox-8 Piccolo. Para las dos primeras vamos a presentar los resultados correspondientes a la clave $F_{16} = 15_{10} = 1111_2$, de tal manera que podamos presentar una comparación fehaciente de

ambas celdas testeadas en las mismas condiciones. Los resultados son extrapolables a cualquier otra clave que se pueda proponer, siendo las diferencias obtenidas desdeñables. Para el caso de la Sbox-8, es difícil presentar una clave en concreto que pueda equipararse a las Sbox-4 a la hora de compararlas, por ello, elegiremos una clave de las 256 posibles de manera arbitraria para obtener estos resultados y representarlos en nuestras próximas tablas sin que, una vez más, muestre diferencias reseñables con respecto al resto de claves. La clave elegida en este caso ha sido $EF_{16} = 239_{10} = 11101111_2$.

Las simulaciones se han realizado en todos los casos proporcionando una entrada de 6000 patrones aleatorios con un tiempo total de simulación de 12 μs. En las Fig 4.1, 4.2 y 4.3 mostramos las trazas de corriente de polarización de cada una de las Sboxes para los primeros 200 ns, de tal manera que puedan apreciarse las variaciones comentadas que nos permiten obtener los parámetros estadísticos que dan cuenta de la seguridad de cada Sbox.

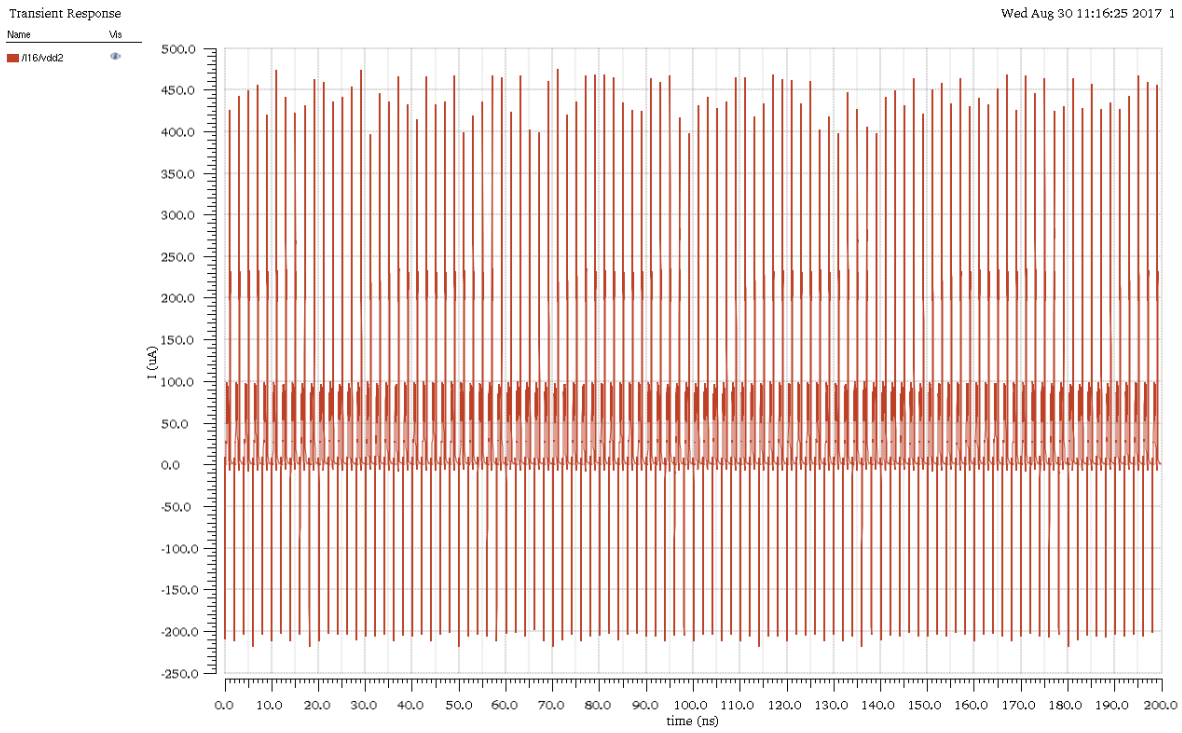


Fig 4.1. Traza de corriente de polarización para la Sbox-4 Piccolo en los primeros 200 ns de análisis.

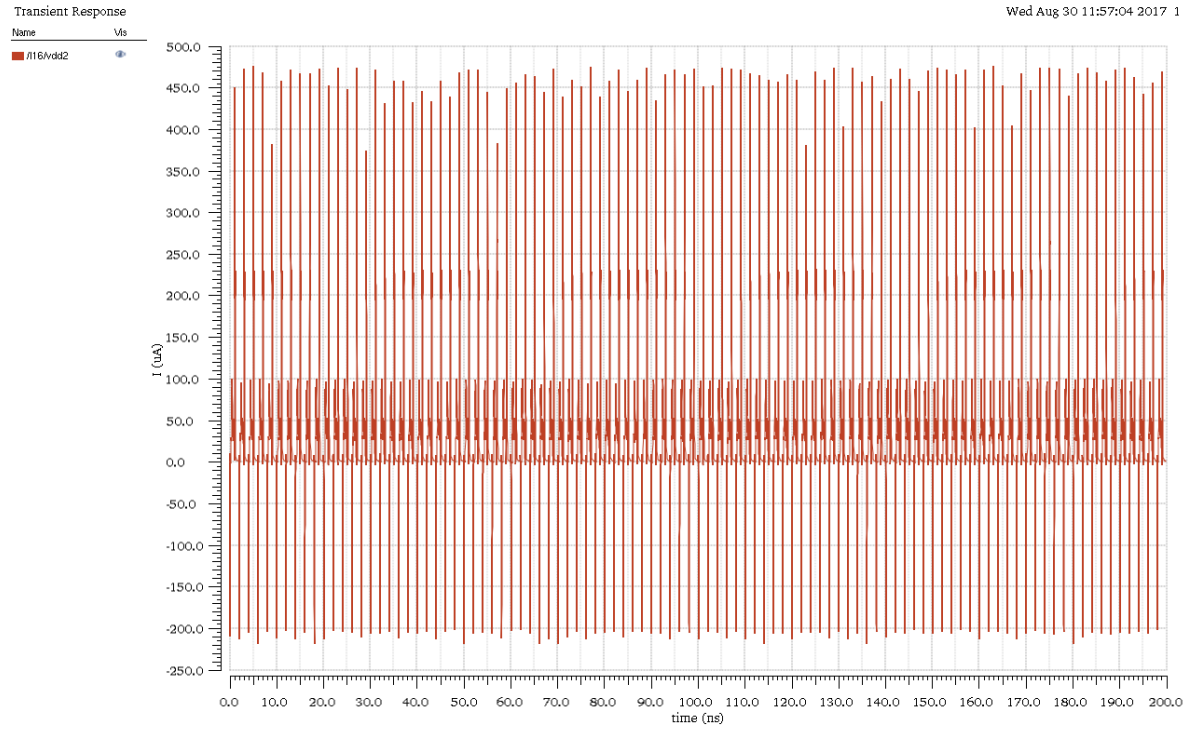


Fig 4.2. Traza de corriente de polarización para la Sbox-4 PRIDE en los primeros 200 ns de análisis.

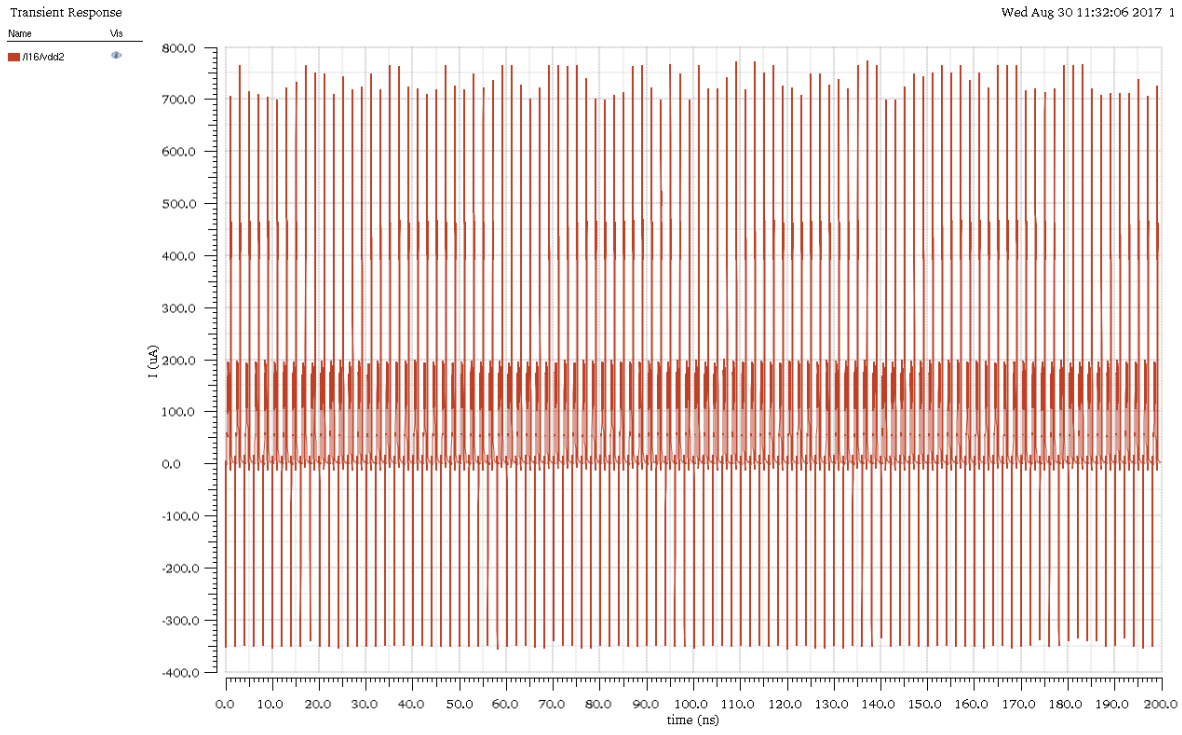


Fig 4.3. Traza de corriente de polarización para la Sbox-8 Piccolo en los primeros 200 ns de análisis.

Viendo estas gráficas y, en virtud de lo establecido por la ecuación (7), al ser V_{DD} y T_{CLK} valores constantes para todas las S-boxes podemos predecir un mayor consumo energético por parte de la Sbox-8 Piccolo con respecto a las Sboxes de 4 bits, como es de esperar al tener el doble de bits que sus competidoras y teniendo en cuenta que la Sbox-8 Piccolo alcanza

intensidades de más de 700 µA mientras que las Sboxes de 4 bits no superan los 500 µA. Además, como ya comentamos en los apartados correspondientes al diseño, un mayor número de celdas y, por tanto, de transistores repercutirá en un mayor consumo energético. La comparación en celdas lógicas y transistores se presenta detalladamente en la Tabla 4.1.

	Celdas AND/NAND	Celdas XOR/XNOR	Celdas OR/NOR	Nº de transistores
Sbox-4 Piccolo	-	4	4	144
Sbox-4 PRIDE	4	4	-	144
Sbox-8 Piccolo	-	8	8	288

Tabla 4.1. Comparación en número de celdas y transistores de las distintas Sboxes.

Una vez hecho el análisis completo de 12 µs para las trazas que anteriormente presentábamos, y hechos los análisis estadísticos oportunos mediante MatLab, hemos obtenido los siguientes resultados:

	Evaluación		Precarga	
	NED	NSD	NED	NSD
Sbox-4 Piccolo	2,14E-02	1,15E-03	4,66E-03	8,62E-04
Sbox-4 PRIDE	1,53E-02	2,06E-03	6,62E-03	9,46E-04
Sbox-8 Piccolo	1,89E-02	1,14E-03	3,73E-03	6,96E-04

Tabla 4.2. Resultados de NED y NSD para cada una de las celdas diseñadas.

	Potencia evaluación (µW)	Potencia Precarga (µW)	Potencia total (µW)
Sbox-4 Piccolo	37,14	43,11	80,25
Sbox-4 PRIDE	38,39	43,65	82,04
Sbox-8 Piccolo	72,32	86,28	158,60

Tabla 4.3. Resultados del consumo de potencia de las Sbox implementadas.

Y según lo previsto por las ecuaciones (7) y (8), para hallar el consumo total de energía solo tendremos que multiplicar por el T_{CLK} que, como hemos dividido las transiciones en fase de precarga y de evaluación, y al ser el período de 2 ns, quedará en T_{CLK} = 1 ns. Por tanto, vamos a obtener la siguiente tabla, que será análoga a la anteriormente presentada pero con dimensiones de energía y multiplicada por un factor de 10^{-9} para el consumo de energía, fruto de la operación entre potencia y T_{CLK}.

	Energía evaluación (fJ)	Energía Precarga (fJ)	Energía Total (fJ)
Sbox-4 Piccolo	37,14	43,11	80,25
Sbox-4 PRIDE	38,39	43,65	82,04
Sbox-8 Piccolo	72,32	86,28	158,60

Tabla 4.4. Resultado del consumo de energía de las Sbox implementadas.

En las tablas se pueden observar los siguientes datos: en la primera de ellas, obtenemos el valor del NED en evaluación y precarga, y el NSD en la evaluación y precarga. En la segunda

se tiene el consumo de potencia en cada una de las fases y en un cómputo global. Mientras que en la tercera, como ya hemos comentado antes, a través de la multiplicación por el tiempo de cada una de las fases obtenemos la energía consumida en cada fase, y en la última columna tenemos la energía total.

Los datos más significativos en cuanto a seguridad van a venir dados por el NED y el NSD en la fase de evaluación puesto que en la fase de precarga el consumo de energía no depende del dato procesado, por tanto, y teniendo en cuenta que tanto NED como NSD dependen de dicho consumo de energía, no tiene sentido valorar la seguridad en base a una medida que no depende de los datos procesados que se pretenden encriptar.

Sacando un gráfico de barras para el NED y NSD en fase de evaluación intentaremos establecer un compromiso entre la seguridad de las diferentes Sbox y su gasto energético.

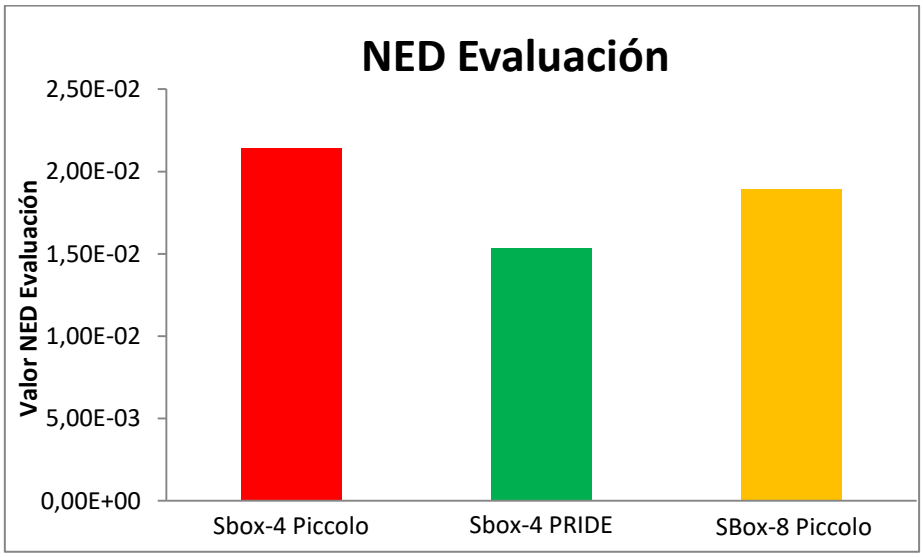


Fig 4.4. Comparativa para el NED en fase de evaluación entre las tres Sbox diseñadas.

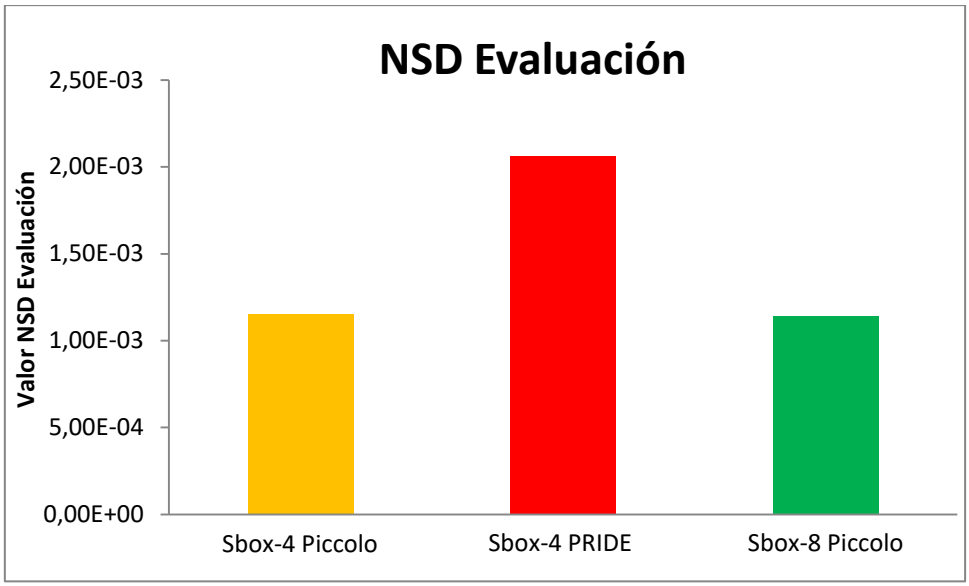


Fig 4.5. Comparativa para el NSD en fase de evaluación entre las tres Sbox diseñadas.

El primer análisis que podemos hacer es que la superioridad en número de bits no siempre implica una clara superioridad a nivel de seguridad en contra de lo que pudiera parecer, dado el mayor número de posibilidades de patrones diferentes a la entrada de la S-Box. De este modo, si medimos el rendimiento de la Sbox-8 por el NED, obtiene un valor de 0,88 veces el que resulta de la Sbox-4, mostrando solo alrededor de un 12% de mejora con respecto a su equivalente en 4 bits y siendo 1,24 veces el obtenido por la Sbox-4 PRIDE. Si nos regimos, ahora, por el parámetro NSD, la Sbox-8 de Piccolo consigue casi los mismos resultados que su análoga de 4 bits con un valor de 0,99 veces el obtenido por la Sbox-4 Piccolo, aunque mejor ampliamente el resultado obtenido por la Sbox-4 PRIDE en un factor de 0,55.Como puede comprobarse por los resultados anteriormente presentados, el aumento de bits, pese a las múltiples posibilidades en cuanto a claves que ofrece comparado con una Sbox-4; no resulta en una mejora clara en cuanto a seguridad. Casualmente, la Sbox-8 Piccolo sí muestra unos resultados considerablemente mejores para el NED y el NSD en la fase de precarga con respecto a sus competidores, pero como ya dijimos anteriormente, no estamos interesados en medir estos parámetros en la fase de precarga dado que no será una medida de la seguridad del bloque por no depender la energía consumida del dato procesado.

Por otro lado, no es fácil establecer cuál de las dos Sbox de 4 bits presenta mayor seguridad ya que la Sbox-4 Piccolo presenta peores resultados en el NED, mientras que la Sbox-4 PRIDE presenta peores resultados en el NSD. Empecemos por analizar su consumo energético. El consumo generado por PRIDE es 1,02 veces el que necesita Piccolo, luego, en este aspecto podemos determinar que ambas celdas están prácticamente igualadas mostrando

una ligera superioridad las especificaciones dadas por Piccolo. En cuanto al NED, PRIDE muestra una clara mejoría siendo este parámetro 0,71 veces el de Piccolo. No obstante, las diferencias en el NSD son superiores a favor de Piccolo, este parámetro ha resultado ser 1,79 veces más grande en el caso de PRIDE que en el de Piccolo. Por tanto, PRIDE solo muestra una mejora del 29% en el NED, mientras que es un 79% peor que la celda implementada por Piccolo según el NSD teniendo un consumo ligerísimamente superior. Luego, si tenemos que evaluar según estas medidas podemos concluir que la Sbox-4 Piccolo presenta un comportamiento algo superior que la Sbox-4 de PRIDE, teniendo en cuenta además la igualdad en el número de transistores que es necesario implementar para montar cada una de las dos Sboxes, tal y como se vio en la Tabla 4.1.

5. Conclusiones.

5.1. Trabajo realizado.

La finalidad fundamental de este trabajo ha sido el diseño microelectrónico en Cadence de Sboxes seguras ante ataques laterales, para su posterior comparación a través de medidas estadísticas indirectas que nos proporcionan una estimación de la seguridad de cada una de ellas. Así, el cuerpo del trabajo ha estado dividido en tres capítulos que dan cuenta de cada uno de los pasos que hemos seguido en nuestra metodología para el diseño y testeo de estos elementos claves en cualquier dispositivo criptográfico.

En el primero de ellos, buscamos un estilo lógico que sea compatible con las aplicaciones criptográficas. Para ello, basándonos en trabajos previos presentes en la bibliografía, hemos hecho un estudio sobre los estilos lógicos que muestran una mayor simetría en sus configuraciones, presentando asimismo un consumo de potencia con menores variaciones que sean resistentes ante ataques laterales, en concreto los DPA, habiendo seleccionado el estilo lógico SABL. Asimismo, se presenta el diseño a nivel de transistor de celdas lógicas tipo SABL en una tecnología TSMC de 90 nm.

En el segundo de ellos, presentamos la descripción y el diseño a nivel de transistor de las propias Sboxes, estableciendo un proceso automatizado y metódico para comprobar la funcionalidad a través del código checker que programamos en MatLab, y a partir de un esquemático de test que aísla los bloques de las Sbox a testear facilitándonos el buen funcionamiento de la misma y que se repite para cada una de las tres implementaciones que hemos hecho según las ecuaciones lógicas de cada cifrador de bloque que contienen estas Sboxes. Los resultados de las simulaciones han pasado el checker sin ninguna dificultad en todos los casos mostrando un buen acuerdo entre lo previsto por la tabla de verdad proporcionadas por los desarrolladores del cifrador de bloque en cada caso, las ecuaciones lógicas que ellos mismos establecen para diseñar estas S-Box y lo observado en nuestros análisis transitorios, tanto en los cortos que nos permitían hacer una comprobación “visual” del funcionamiento de cada bloque, como las trazas que recogíamos para su posterior procesado automatizado en nuestro código de MatLab. Además, utilizando dos Sbox-4 hemos diseñado un bloque Sbox de 8 bits que realiza la misma función que las de 4 bits, mostrando una relación biyectiva entre entrada y salida en función de la clave personal establecida, tal y como se pretendía. Este bloque nos sirvió para hacer una comparación entre las Sbox de 4 bits

y la de 8, analizando la rentabilidad que ofrece implementar una Sbox con mayor número de bits.

Para finalizar, en el último capítulo se da una definición detallada de los parámetros estadísticos que vamos a utilizar para comparar cada una de las Sboxes entre sí. El trabajo en este capítulo se ha centrado en automatizar el proceso de almacenamiento y procesado de datos con los dos scripts generados para tener una medida de esos parámetros generalizada, sin importar el número de bits que implemente cada bloque, las ecuaciones lógicas de los mismos o el número de patrones con los que se quiere hacer el análisis. Los resultados han mostrado que, pese a mostrar cierta mejora con respecto a las Sboxes de 4 bits, un aumento en el número de bits que implementen las SBox no resulta en una mayor seguridad. Por otro lado, no ha sido sencillo establecer cuál de las 2 Sboxes de 4 bits muestra una mayor seguridad puesto que los parámetros utilizados ofrecen resultados dispares, aunque si otorgamos el mismo nivel de fiabilidad a los dos, la Sbox-4 Piccolo presenta resultados ligeramente mejores.

5.2. Conclusiones y aportaciones al alumno.

Las aportaciones y conclusiones fundamentales que pueden deducirse de la realización de este Trabajo de Fin de Grado son:

1. Se ha producido una introducción y primer aproximamiento al mundo de la criptografía, el cual es un territorio inexplorado para un estudiante de Física medio. A través de este acercamiento, se han utilizado por primera vez herramientas útiles en el ámbito de la Microelectrónica como son todas aquellas relacionadas con el software Cadence-Virtuoso.
2. Se han diseñado puertas lógicas y Sboxes en el entorno de Cadence, que después han sido testeadas demostrando que la implementación de dichos diseños estaba de acuerdo con lo previsto por los desarrolladores de los cifradores de bloque que implementan estas Sboxes y que se encuentran en el estado del arte de la criptografía.
3. Se ha comparado las diferentes Sboxes diseñadas entre sí, observándose que el hecho de construir una SBox con un mayor número de bits no tiene por qué llevar a una mayor seguridad del bloque criptográfico.
4. La realización de un ataque DPA es, en ocasiones, la manera más acertada de comparar dos Sboxes entre sí dado que los parámetros estadísticos presentados

muestran resultados contradictorios que no nos llevan a una conclusión clara sobre la seguridad de los bloques.

Algunas de las ideas que se han quedado en el tintero, y que pueden ser de gran interés como tarea futura siguiendo la línea de este Trabajo de Fin de Grado pueden ser la realización de ataques DPA satisfactorios para una comparación más exacta y concreta de las celdas diseñadas. Además, otro de los análisis futuros que podría llevarse a cabo es aquel que establezca una relación fiable entre número de bits, consumo energético y seguridad criptográfica teniendo como objetivo el encontrar la máxima optimización para las tres variables anteriormente detalladas.

Bibliografía.

[1] P. C. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems,” in *Proc.Int.Cryptol.Conf.*, 1996, pp. 104–113.

[2] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks : revealing the secrets of smart cards*. Springer, 2007.

[3] P. Kocher, J. Jaffe, and B. Jun, “Differential Power Analysis,” in *Proc.Int.Cryptol.Conf.*, 1999, pp. 388–397.

[4] E. Tena-Sanchez, J. Castro, and A. J. Acosta, “A Methodology for Optimized Design of Secure Differential Logic Gates for DPA Resistant Circuits,” *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 4, no. 2, pp. 203–215, Jun. 2014.

[5] E. Tena-Sanchez and A. J. Acosta, “DPA vulnerability analysis on Trivium stream cipher using an optimized power model,” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 1846–1849.

[6] Y. Ishai, A. Sahai, and D. Wagner, “Private Circuits: Securing Hardware against Probing Attacks,” in *Proc.Int.Cryptol.Conf.*, Springer, Berlin, Heidelberg, 2003, pp. 463–481.

[7] S. Banik *et al.*, “Midori: A Block Cipher for Low Energy,” in *Advances in Cryptology - ASIACRYPT 2015*, Springer, Berlin, Heidelberg, 2015, pp. 411–436.

[8] S. Guilley, F. Flament, P. Hoogvorst, R. Pacalet, and Y. Mathieu, “Secured CAD Back-End Flow for Power-Analysis-Resistant Cryptoprocessors,” *IEEE Des. Test Comput.*, vol. 24, no. 6, pp. 546–555, Nov. 2007.

[9] M. W. Allam and M. I. Elmasry, “Dynamic current mode logic (DyCML): a new low-power high-performance logic style,” *IEEE J. Solid-State Circuits*, vol. 36, no. 3, pp. 550–558, Mar. 2001.

[10] I. Hassoune, F. Mace, D. Flandre, and J.-D. Legat, “Low-swing current mode logic (LSCML): A new logic style for secure and robust smart cards against power analysis attacks,” *Microelectronics J.*, vol. 37, no. 9, pp. 997–1006, Sep. 2006.

[11] K. Tiri, M. Akmal, and I. Verbauwhede, “A Dynamic and Differential CMOS Logic

with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards," *Eur. Solid-State Circuits Conf.*, pp. 403–406, 2002.

[12] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, pp. 246–251.

[13] E. Tena-Sanchez, "Desarrollo de celdas lógicas digitales resistentes a ataques DPA," University of Seville, 2013.

[14] K. Tiri and I. Verbauwhede, "Design Method for Constant Power Consumption of Differential Logic Circuits," in *Design, Automation and Test in Europe*, pp. 628–633.

[15] "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," *FIPS PUB 197*, 2001.

[16] "Data Encryption Standard (DES)," *FIPS PUB 46*, 1977.

[17] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "TWINE : A Lightweight Block Cipher for Multiple Platforms," in *Selected Areas in Cryptography*, Springer, Berlin, Heidelberg, 2013, pp. 339–354.

[18] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: An Ultra-Lightweight Blockcipher," in *Cryptographic Hardware and Embedded Systems - CHES 2011*, Springer, Berlin, Heidelberg, 2011, pp. 342–357.

[19] M. R. Albrecht, B. Driessen, E. B. Kavun, G. Leander, C. Paar, and T. Yalçın, "Block Ciphers – Focus on the Linear Layer (feat. PRIDE)," in *Advances in Cryptology - CRYPTO 2014*, Springer, Berlin, Heidelberg, 2014, pp. 57–76.

Anexo.

Código checker utilizado para la Sbox-4 Piccolo.

```
%% Programa checker comprobacion Sbox
% Cargamos valores de entrada
ina= load('ina_16');
ina= ina(:,2);
inb= load('inb_16');
inb= inb(:,2);
inc= load('inc_16');
inc= inc(:,2);
ind= load('ind_16');
ind= ind(:,2);
clk= load('clk_16');
clk= clk(:,2);

n=2000;
m=200;
for j=1:n
 i=(j-1)*m+1:(j-1)*m+m;
    IN(j,1)= sum(ina(i)>0.6)>0;
    IN(j,2)= sum(inb(i)>0.6)>0;
    IN(j,3)= sum(inc(i)>0.6)>0;
    IN(j,4)= sum(ind(i)>0.6)>0;
end
% Cargamos valores de salida
outa= load('outA_16');
outa= outa(:,2);
outb= load('outB_16');
outb= outb(:,2);
outc= load('outC_16');
outc= outc(:,2);
outd= load('outD_16');
outd= outd(:,2);

for j=1:n
 i=(j-1)*200+1:(j-1)*200+200;
    OUT(j,1)= sum(outa(i)>0.6)>0;
    OUT(j,2)= sum(outb(i)>0.6)>0;
    OUT(j,3)= sum(outc(i)>0.6)>0;
    OUT(j,4)= sum(outd(i)>0.6)>0;
end
OUTHEX=dec2hex(bin2dec(num2str(OUT)));
% Pasamos checker
for j=1:n
    OUT_Soft(j,1)= xor(IN(j,4),~or(IN(j,1),IN(j,2)));
    OUT_Soft(j,2)= xor(IN(j,1),~or(IN(j,2),IN(j,3)));
    OUT_Soft(j,3)= ~xor(IN(j,2),~or(OUT_Soft(j,1),IN(j,3)));
    OUT_Soft(j,4)= xor(IN(j,3),~or(OUT_Soft(j,2),OUT_Soft(j,1)));
end
OUT_Soft;
OUTHEX_Soft=dec2hex(bin2dec(num2str(OUT_Soft)));
check=isempty(find((OUTHEX_Soft==OUTHEX)==0));
if check==1
    disp('Functionality OK')
else
    disp('ERROR!!!!')
end
```

Modificaciones del código checker para utilizarlo en la Sbox-4 PRIDE.

```
for j=1:n
    OUT_Soft(j,1)= xor(IN(j,3),and(IN(j,1),IN(j,2)));
    OUT_Soft(j,2)= xor(IN(j,4),and(IN(j,2),IN(j,3)));
    OUT_Soft(j,3)= xor(IN(j,1),and(OUT_Soft(j,1),OUT_Soft(j,2)));
    OUT_Soft(j,4)= xor(IN(j,2),and(OUT_Soft(j,2),OUT_Soft(j,3)));
end
```

Código checker utilizado para la Sbox-8 Piccolo.

22/08/17 19:28 C:\Users\Nacho\Doc...\CodigoAtaqueSbox8.m 1 of 2

```
%% Programa checker comprobacion Sbox

% Cargamos valores de entrada
ina= load('ina_10');
ina= ina(:,2);
inb= load('inb_10');
inb= inb(:,2);
inc= load('inc_10');
inc= inc(:,2);
ind= load('ind_10');
ind= ind(:,2);
ine= load('ine_10');
ine= ine(:,2);
inf= load('inf_10');
inf= inf(:,2);
ing= load('ing_10');
ing= ing(:,2);
inh= load('inh_10');
inh= inh(:,2);
clk= load('clk_10');
clk= clk(:,2);
%n tiene que ser igual al n1¿½mero de transiciones que se ven, es decir,
%puntos totales/puntos por transiciÏ¿½n. m es el valor de puntos por
%transiciÏ¿½n
n=2000;
m=200;
for j=1:n
 i=(j-1)*m+1:(j-1)*m+m;
    IN(j,1)= sum(ina(i)>0.6)>0;
    IN(j,2)= sum(inb(i)>0.6)>0;
    IN(j,3)= sum(inc(i)>0.6)>0;
    IN(j,4)= sum(ind(i)>0.6)>0;
    IN(j,5)= sum(ine(i)>0.6)>0;
    IN(j,6)= sum(inf(i)>0.6)>0;
    IN(j,7)= sum(ing(i)>0.6)>0;
    IN(j,8)= sum(inh(i)>0.6)>0;
end

% Cargamos valores de salida
outa= load('OutA_10');
outa= outa(:,2);
outb= load('OutB_10');
outb= outb(:,2);
outc= load('OutC_10');
outc= outc(:,2);
outd= load('OutD_10');
outd= outd(:,2);
oute= load('OutE_10');
oute= oute(:,2);
outf= load('OutF_10');
outf= outf(:,2);
outg= load('OutG_10');
outg= outg(:,2);
outh= load('OutH_10');
```

```
%n tiene que ser igual al nÍ¿½mero de transiciones que se ven, es decir,
%puntos totales/puntos por transiciÍ¿½n. m es el valor de puntos por
%transiciÍ¿½n

for j=1:n
 i=(j-1)*m+1:(j-1)*m+m;
    OUT(j,1)= sum(outa(i)>0.6)>0;
    OUT(j,2)= sum(outb(i)>0.6)>0;
    OUT(j,3)= sum(outc(i)>0.6)>0;
    OUT(j,4)= sum(outd(i)>0.6)>0;
    OUT(j,5)= sum(oute(i)>0.6)>0;
    OUT(j,6)= sum(outf(i)>0.6)>0;
    OUT(j,7)= sum(outg(i)>0.6)>0;
    OUT(j,8)= sum(outh(i)>0.6)>0;
end
OUTHEX=dec2hex(bin2dec(num2str(OUT)));
% Pasamos checker

for j=1:n
    OUT_Soft(j,1)= xor(IN(j,4),~or(IN(j,1),IN(j,2)));
    OUT_Soft(j,2)= xor(IN(j,1),~or(IN(j,2),IN(j,3)));
    OUT_Soft(j,3)= ~xor(IN(j,2),~or(OUT_Soft(j,1),IN(j,3)));
    OUT_Soft(j,4)= xor(IN(j,3),~or(OUT_Soft(j,2),OUT_Soft(j,1)));
    OUT_Soft(j,5)= xor(IN(j,8),~or(IN(j,5),IN(j,6)));
    OUT_Soft(j,6)= xor(IN(j,5),~or(IN(j,6),IN(j,7)));
    OUT_Soft(j,7)= ~xor(IN(j,6),~or(OUT_Soft(j,5),IN(j,7)));
    OUT_Soft(j,8)= xor(IN(j,7),~or(OUT_Soft(j,6),OUT_Soft(j,5)));
end
OUT_Soft;
OUTHEX_Soft=dec2hex(bin2dec(num2str(OUT_Soft)));
check=isempty(find((OUTHEX_Soft==OUTHEX)==0));
if check==1
    disp('Functionality OK, continue with attack')
else
    disp('ERROR!!!!')
end
```

Código para el cálculo automatizado de los parámetros estadísticos y el consumo de potencia y energía.

```
%% Programa checker comprobacion Sbox
clear all;close all;clc;
key=15;
% Cargamos valores de entrada
file=strcat('ivdd_',num2str(key));
ivdd= load(file);
ivdd= ivdd(:,2);
%n va a ser el numero de patrones menos uno, porque sino al haber quitado
%las primeras componentes el numero de puntos que va a indexar va a ser
%mayor que el numero de componentes del vector.
n=11999;
%Saco a partir de la componente 13 en adelante que es cuando empieza la
%evaluacion y a partir de ahi corto de 100 en 100 puntos(voy a tener una
%componente de evaluacion, la siguiente de precarga y asi
%sucesivamente).Calculo la media de intensidad en cada corte, es decir, en
%cada evaluacion y cada precarga y las almaceno en un vector
for j= 1:n
    i=(j-1)*100+13:(j-1)*100+112;
    avg (1,j)= mean (ivdd(i));
end
%Ahora separo las componentes pares de las impares, a fin de separar las
%evaluaciones de las precargas. Y calculo todo lo demas con las formulas
%que tengo.
avg_evaluation= avg (1:2:length(avg));
avg_precharge= avg (2:2:length (avg));
Eavg_evaluation= 1.2*1e-9*mean(avg_evaluation)
Eavg_precharge= 1.2*1e-9*mean(avg_precharge)
Pavg_evaluation= 1.2*mean(avg_evaluation)
Pavg_precharge= 1.2*mean(avg_precharge)
NED_evaluation=(max(avg_evaluation)-min(avg_evaluation))/max(avg_evaluation)
NED_precharge=(max(avg_precharge)-min(avg_precharge))/max(avg_precharge)
NSD_evaluation= std(avg_evaluation)/mean(avg_evaluation)
NSD_precharge= std(avg_precharge)/mean(avg_precharge)
Eavg_Total= Eavg_evaluation + Eavg_precharge
Pavg_Total=Pavg_evaluation + Pavg_precharge
```