PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

EVALUACIÓN ELÉCTRICA Y FÍSICA DE MÉTODOS DE GENERACIÓN DE REDES LÓGICAS PARA COMPUERTAS ESTÁTICAS CMOS COMPLEMENTARIAS (SCCG)

Tesis para obtener el título profesional de Ingeniero Electrónico

AUTOR:

Jair Moises Perez Ramirez

ASESOR:

Carlos Bernardino Silva Cárdenas

Lima, Diciembre, 2022

Informe de Similitud

Yo, ……CARLOS SILVA CARDENAS……,

docente de la Facultad de …CIENCIAS E INGENIERIA de la Pontificia Universidad Católica del Perú, asesor(a) de la tesis/el trabajo de investigación titulado

EVALUACIÓN EL ÉCTRICA Y FÍSICA DE MÉTODOS DE GENERACIÓN DE REDES LÓGICAS PARA COMPUERTAS ESTÁTICAS CMOS COMPLEMENTARIAS (SCCG)

del/del autor(a)/ de los(as) autores(as) JAIR MOISES PEREZ RAMIREZ, dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 9%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 18/01/2023.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: …LIMA, 23 ENERO 2023…

Apellidos y nombres del asesor / de la asesora: SILVA CARDENAS, CARLOS BERNARDINO	
DNI:08014721	Firma
ORCID: 0000-0003-4653-0915	

Dedicado a mis padres Anita y Beto quienes me enseñaron que la perseverancia, el respeto y la humildad son los pilares fundamentales para alcanzar nuestros sueños.
A mi hermano Pierre por ser mi ejemplo cuando más lo necesitaba.
A mis tías Miriam y Carmen por acompañarme desde mi niñez y apoyarme incondicionalmente.
A mis gatos Koko y Lolo por ser los perfectos compañeros de estudio, sacarme siempre una sonrisa y alegrarme el día. Son mi motor.
A Loreley por enseñarme a valorar nuevos aspectos de la vida, y quien con su amor y apoyo fortalece mi deseo de crecer.

A Octavio por los bellos momentos que vivimos en nuestra adolescencia que forjaron nuestra fuerte amistad.
A Adrián por enseñarme la virtud del autoaprendizaje, ser la voz consejera y mi compañero de carrera durante mi paso por la USB.
A Julio, Kendhal y Eduardo por los ser los fieles compañeros de proyectos y salidas.

A mi tío Jorge por apoyarme con presencia y almuerzos en mi llegada al Perú. Un abrazo al cielo.
A mi tía Marta por ayudarme en mi deseo de retormar mis estudios. Por los almuerzos domingueros, comida y hospedaje. Un beso al cielo.

Agradezo a mi maestro y amigo Mg. Ing. Mario Raffo por ser el profesor que depositó su confianza en mi y guiarme en mi camino a conventirme en el profesional que deseo ser.

Agradezco a mi asesor Dr. Ing. Carlos Silva por sus enseñanzas, comprensión, paciencia y aptitud para guiarme en cada paso del desarrollo de este trabajo.

Agradezco también al Dr. Ing. Julio Saldaña, al Ing. Kelvin Yllahuamán y al Ing. Jesús Salazar por apoyarme con su disposición, herramientas y asesorías.

Agradezco al Grupo de Microelectrónica GμE-PUCP por todo el apoyo brindado.

Agradezco a Henrique Kessler, Adriel Ziesemer, Gabriel Ammes y Renato Ribas por su disponibilidad a mis consultas y brindarme las herramientas en las que se basaron este trabajo.

"Si queremos un mundo de paz y de justicia
hay que poner decididamente la inteligencia
al servicio del amor."

Antoine de Saint-Exupéry

Resumen

Recientemente la evolución de la industria de la microelectrónica ha permitido el desarrollo de herramientas de diseño electrónico automático (EDA), las cuales tienen por objetivo optimizar el proceso de diseño de circuitos integrados (IC). Tradicionalmente en la creación de un IC se suele utilizar el enfoque de diseño de celdas estándar; no obstante, este tipo de flujo de diseño se encuentra limitado por la cantidad de compuertas lógicas que estén definidas en la libreria utilizada. Es por ello que diversos estudios han realizado investigaciones respecto a la optimización de circuitos por Compuertas CMOS Estáticas Complemetarias (SCCG). En la literatura podemos encontrar diversas estrategias de diseño de compuertas SCCG; sin embargo, la métrica que se usa para definir el mejor arreglo es la cantidad de transistores, la cual carece de otros análisis concernientes a los parámetros eléctricos y físicos. Es por ello que en este trabajo de tesis se plantea evaluar las redes de transistores SCCG generadas por el framework SwitchCraft mediante un análisis elétrico realizado con el software CADENCE y un análisis físico de los layouts generados por medio de la herramienta ASTRAN.

Palabras claves–SCCG, Síntesis Lógica, Síntesis Física, SwitchCraft, ASTRAN.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Scripts

Introducción

El desarrollo de la tecnología en diversas área de la microelectrónica se ha dado de manera significativa en los últimos cincuenta años. En gran parte las aplicaciones derivadas de este desarrollo han permitido alcanzar una revolución digital en la que los principales protagonistas de este acontecimiento son los circuitos integrados. Es por este motivo que la industria de la microelectrónica ha evolucionado con respecto a las estrategias de implementación de circuitos integrados digitales, siempre teniendo en cuenta el aspecto económico. Recientes desarrollos en las herramientas EDA (Electronic Design Automation) presentan un nuevo enfoque en el que se emplean compuertas SCCG con la finalidad de reducir la cantidad de transistores y la longitud de las conexiones en un circuito integrado. De esta manera se obtiene una solución económica, ya que se utilizan menos transistores y por ende una reducción en los costos por adquisición del material silicio. Las más recientes investigaciones enfocadas a comparaciones entre diferentes métodos de generación para compuertas SCCG emplean como única métrica la cantidad de transistores; por consiguiente, el proceso puede que no guíe a un resultado óptimo. Es por este motivo, que en la presente tesis se tiene por objetivo realizar una evaluación eléctrica y física de métodos de generación para compuertas SCCG, ya que se ha observado en la literatura que no necesariamente una cantidad menor de transistores representa un circuito más óptimo.

Este documento está estructurado en cinco partes: cuatro capítulos y conclusiones. El primer capítulo, aborda el estado del arte el cual presenta estudios que son la base para el análisis propuesto; asimismo, se definen los objetivos para el desarrollo del estudio. El segundo capítulo, explora los fundamentos teóricos que se requieren para el entender como una función Booleana es implemetada por redes lógicas. De la misma manera, se explican los conceptos de compuertas CMOS y SCCG. El tercer capítulo, es el núcleo de este trabajo y se enfoca en plantear la propuesta de metodología para estudiar un conjunto de funciones lógicas de 4 entradas de equivalencia P. Las redes lógicas serán generadas por tres métodos FAC, FAC-PD y BDD, mediante un framework desarrollado por investigadores de la Universidade Federal do Rio Grande do Sul denominado SwitchCraft. El análisis eléctrico se llevará a cabo mediante el

software CADENCE y el análisis físico se podrá realizar con la ayuda de la herramienta de generación de capas (layout) denominada ASTRAN, también desarrollada por investigadores de la Universidade Federal do Rio Grande do Sul. El cuarto capítulo, se exponen y discuten los resultados obtenidos de las simulaciones y evaluaciones realizadas en el capítulo previo. Finalmente, se realizan conclusiones en base a los resultados obtenidos.

Capítulo 1

Marco problemático

1.1. Entorno general

El mundo ha experimentado un incremento en el uso de circuitos integrados en los últimos años. De esta manera, la industria ha venido desarrollando avances con respecto al proceso de diseño de estos circuitos con la finalidad de satisfacer la demanda del mercado.

1.2. Motivación

El uso de los circuitos integrados han evolucionando hacia funciones más complejas como: la ejecución de software, la manipulación de datos y el control de funciones de dispositivos electrónicos; por consiguiente, con la necesidad de cumplir con tales funcionalidades, cada vez un mayor número de transistores están siendo empleados en los circuitos integrados (ICs). Este hecho se refleja al realizar un contraste entre la cantidad de transistores que poseía el primer microprocesador de la empresa Intel y los dispositivos más actuales como la GPU GeForce RTX 3090 de la empresa NVIDIA, el primero contenía tan solo 2300 transistores a diferencia de la tarjeta gráfica que posee 28,3 mil millones de transistores [2].

Es por ello que, desde un punto de vista económico, el uso de una mayor cantidad de transistores implica costos mayores debido a que el área de silicio que se emplea para su construcción aumenta. De esta manera, con la finalidad de reducir gastos económicos, se debe pensar en soluciones a diversos niveles, siendo uno de ellos a nivel de diseño, en el cual se deben buscar optimizaciones referentes a la generación de las redes de transistores.

1.3. Estado del arte

En la literatura más reciente, respecto al diseño digital VLSI (Very Large-Scale Integration), se ha podido observar que los estudios se han enfocado en la optimización de métodos de generación de compuertas SCCG (Static CMOS Complex Gates). Uno de los procesos más importantes en el proceso de optimización corresponde a la generación de la red de transistores que representa la función lógica Booleana que se desea implementar. Es por ello que las metodologías que se emplean, con miras de optimización en el proceso de diseño, apuntan hacia un menor uso de transistores.

En ese sentido, en el estudio titulado "Area-Aware Design of Static CMOS Complex Gates" [1], analizan el impacto en la capa de área de circuitos no-planos que son generados por un método que denominan Kernel Finder [3]. Esta metodología genera de redes de transistores con enfoque SCCG; asimismo, emplea un menor número de transistores en comparación a otras metodologías de generación basado en gráficos (Graph-Based) o métodos de factorización Booleana, ya que el algoritmo de generación permite conexiones no-serial-paralela. Sin embargo, esta técnica produce irregularidades en la topología de la red, ya que surgen arreglos no-planos y no-duales [1]. Es importante tener en cuenta que con el término no-plano los autores hacen referencia a redes que poseen una configuración no-serial-paralela. Con respecto al término no-dual este alude al hecho de que la red de pull-down no puede se puede obtener a partir de la red de pull-up (o viceversa), debido a que esto es únicamente posible para redes que presentan la propiedad de dualidad. En la Figura 1.1 se visualiza un ejemplo de lo mencionado anteriormente.

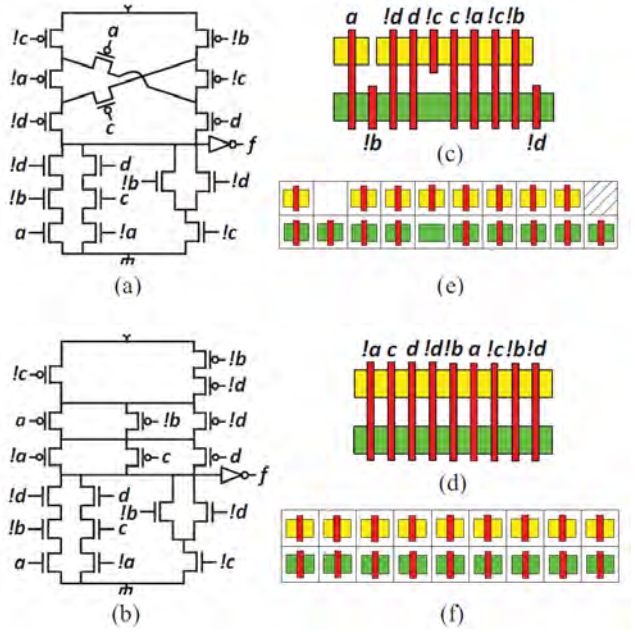


Figura 1.1: Compuertas SCCG y pseudo capas de una una función lógica Booleana [1].

El circuito (a) y (b) representan la misma función lógica. No obstante, el circuito (a), a pesar de emplear un transistor menos, resulta ser no-plano y no-dual en contraste con el circuito (b) que sí es plano y dual. Al contrastar las pseudo capas, ya que no se consideran los procedimientos de compactación y enrutamiento, podemos destacar que la capa (c) presenta lo siguiente: una ruptura de difusión en el plano pull-up, el cual está representado por la barra de color amarillo; rupturas de polisilicios, graficados como las barras verticales de color rojo; una divergencia en el tamaño de áreas activas, lo cual es previsto considerando que el plano pull-up (barra amarilla) posee un transistor más que el plano pull-down (barra verde); y un espaciado no regular entre las compuertas de polisilicios en el plano de pull-down.

Por consiguiente, los investigadores proponen un nuevo enfoque para el diseño lógico de estas celdas en el que se puedan minimizar estas irregularidades. La metodología propuesta consiste en obtener una capa regular de una compuerta SCCG a través de la generación de redes duales. Es por ello que el algoritmo debe ser capaz de producir una solución divergente solo cuando a una única red pull-up (PU) o pull-down (PD) es no-plano, ya que es posible obtener una red dual cuando ambas redes PU y PD son no-planas. El algoritmo propuesto primero analiza la topología de las redes pull-up y pull-down antes de generar la compuerta SCCG. Esto se logra evaluando la planicidad después de cada plan lógico generado, de manera que en caso se detecte una red no-plana se generen planos lógicos duales incluso si la solución implica emplear más transistores.

Con la finalidad de evaluar la capa de área hay que considerar aspectos como el enrutamiento de celdas, su complejidad y la longitud de los caminos. Es por ello que se emplea una herramienta de código abierto denominada ASTRAN. Los resultados de estos estudios demuestran que, a pesar de emplear mayores transistores, en comparación a la metodología Kernel Finder, se consigue una optimización geométrica. Debido a que se obtienen celdas con un decremento de 5,84 % en columnas. Asimismo, se obtuvo optimizaciones en las capas con un decremento de 15,81 % en área y 3,18 % en la longitud en los caminos del enrutamiento[1].

Otro estudio titulado “Electrical Evaluation of Logic Network Generation Methods for On-the-Fly Supergate Design” [4], mencionan que a pesar de las diversas estrategias de diseño enfocada en las compuertas SCCG, las comparaciones están limitadas a métricas como el número de transistores, careciendo de una evaluación eléctrica. Por consiguiente, cuando se optimizan técnicas dirigidas a métricas que no son el propósito final de diseño el proceso puede que no conduzca a un resultado más óptimo [4]. De esta manera, el estudio presenta una comparación eléctrica de tres diferentes técnicas de diseño: FAC, FAC-PD y BDD.

El primer método FAC, tiene por objetivo reducir la cantidad de transistores en la compuerta lógica. Para lograr esto las redes de pull-up y pull-down son optimizadas por separadas

resultando en una compuerta lógicamente complementaria pero no necesariamente topológicamente complementaria. El segundo método FAC-PD, no reduce la cantidad de literales en la función lógica. La expresión es utilizada para construir la red de pull-down y la red de pull-up es construida para ser topológicamente y lógicamente complementaria. Por último, el tercer enfoque BDD (Binary Decision Diagram) emplea un método gráfico para hallar arreglos serial-paralelo y no-serial-paralelo.

El estudio planteó evaluar funciones de 4 entradas; sin embargo, como es de conocer 4 entradas pueden generar 65,536 funciones lógicas. En consecuencia, se optó por obtener un grupo reducido de funciones considerando la equivalencia P, la cual se puede entender como funciones que posee una misma respuesta al permutar el orden de sus entrada. Por ejemplo, la red de la función $x.!y$ posee el mismo comportamiento que la red de la función $!x.y$ [5]. Por ello, finalmente se obtuvo un conjunto reducido de 3,982 funciones lógicas de equivalencia P de 4 entradas. Para la generación de las redes de transistores se empleó el framework SwitchCraft y el software *Spice* fue utilizado para la creación del ambiente de simulación y la evaluación eléctrica. Los resultados demuestran que, respecto a características eléctricas, la técnica FAC obtiene mejores resultados en todas las métricas [4].

Tabla 1.1: Resultados normalizados al método FAC para un conjunto de funciones de equivalencia P y 4 entradas [4].

	FAC	**FAC-PD**	**BDD**
Retraso promedio	1.000	1.249	1.055
Retraso máximo	1.000	1.224	1.027
Potencia estática minima	1.000	1.291	1.289
Potencia estática promedia	1.000	1.239	1.115
Potencia promedio	1.000	0.999	1.061
No. de transistores	1.000	1.094	1.012
Estimación de área	1.000	1.021	1.029
Pila Pull-Up	1.000	1.021	0.980
Pila Pull-Down	1.000	1.073	0.980

Por consiguiente, se sugiere que dicha técnica es la más apropiadas para el diseño cuando se emplea el framework SwitchCraft. Asimismo, los resultados demuestran que tener los transistores cerca de las líneas de la fuente de alimentación mejoran sus características eléctricas. En añadidura, los investigadores mencionan que los resultados muestran que las comparaciones entre redes lógicas respecto a la cantidad de transistores pueden no ser válidas para el rendimiento del circuito [4].

En la actualidad existen diversas herramientas que permiten evaluar diversas soluciones de redes de transistores para una misma función lógica. Entre ellas están herramientas a nivel industrial como CADENCE y también herramientas realizadas por investigadores tales como ABC Tool y SwitchCraft. Esta última herramienta mencionada, fue desarrollada en lenguaje Java con la finalidad de ser una plataforma independiente del sistema operativo [6]. Tiene por objetivo principal proveer un conjunto de algoritmos y métodos que ayuden a los diseñadores a generar redes de transistores y compuertas lógicas, y a su vez estimar y evaluar sus características. Los autores mencionan que el concepto detrás de su desarrollo esta la posibilidad de hacer de SwitchCraft un proyecto de código abierto.

Posee cuatro módulos principales: Datos de entrada, generación de red de transistores, perfil y estimación de la red de transistores, y visualización de la red de transistores. El primer módulo, acepta formatos de entrada como: Expresión Booleana, BDD, Tabla de la verdad, BLIF y Spice netlist. Esta información es empleada por el segundo módulo, el cual emplea algoritmos para generar la red de transistores en los siguientes formatos:

- Directo de la ecuación.
- Directo de la expresión factorizada.
- Basado en BDD.
- Basado en BDD con optimización inactiva.
- Basado en BDD respecto a un mínimo de transistores.
- Basado en MUX.
- Doble carril.
- Red complementaria.

Cada uno de estos métodos generan redes de transistores con diferentes arreglos y características. Es por ello, que el tercer módulo de la herramienta provee un perfil y estimaciones que nos indican la siguiente información:

- Número total de transistores.
- Máximo número de transistores en serie.
- Número de nodos en la red.

- Número de conexiones en nodos internos.
- Número de caminos entre nodos terminales.
- Camino más corto entre nodos terminales.
- Camino más largo entre nodos terminales.
- Modelo de retraso de Elmore.
- Disipación de potencia dinámica.
- Disipación de potencia estática.
- Estimación de área.

Consecuentemente, en el año 2017, los investigadores Ammes, Gabriel et al. [7]. implementaron dos mejoras al framework SwitchCraft. La primera, realizada en el primer módulo, consiste en la expansión de los formatos lógicos de entrada y salida mediante la integración de una herramienta llamada Logic2Logic, la cual nos permite representar la función de entrada en formatos AIG (And Inverter Diagram) y TLF (Threshold Logic Functions). El primer formato AIG consiste en un DAG (Directed Acyclic Graph) donde los nodos están representados por puertas AND2. Tales nodos están conectados por aristas que tienen su polaridad definida por inversores representados por burbujas al extremo de las mismas. El segundo formato TLF, son un subconjunto de funciones booleanas, que pueden ser implementadas por una estructura llamada TLG (Threshold Logic Gate). Dicha estructura está compuesta por un conjunto de entradas, donde cada entrada tiene un valor de peso y un valor umbral asociado a la función. El comportamiento lógico del TLF se define por la suma de los pesos de las entradas [7]. Con respecto a la segunda mejora, implementada en el tercer módulo, corresponde a la inserción de un método de colocación de transistores aplicando el algoritmo convencional de Caminos Eulerianos, este algoritmo permite reducir el número de rupturas de difusión durante la generación de la capa. El objetivo es conseguir el mismo camino ininterrumpido en los planos pull-up y pull-down de una compuerta lógica. Caso contrario, cuando no es posible conseguir un único camino Euleriano, el algoritmo retorna dos o más sub-caminos separados por una ruptura de difusión [7].

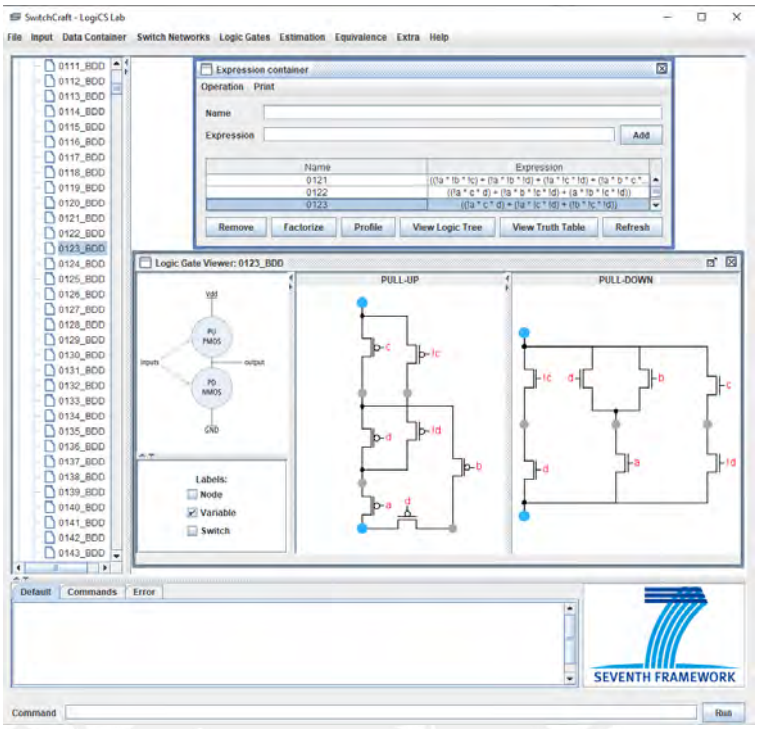


Figura 1.2: Interfaz gráfica del usuario del framework SwitchCraft [6],[7].

1.4. Objetivos

1.4.1. Objetivo General

- Evaluar y analizar las métricas eléctricas y físicas de métodos de generación de redes lógicas para compuertas SCCG, con el propósito de determinar el arreglo circuital que posea mejores características eléctricas y físicas.

1.4.2. Objetivos específicos

1. Definir el método de generación más óptimo para implementar una función lógica, con la finalidad de reducir costos de diseño y la probabilidad de error en el funcionamiento circuital
2. Emplear el framework de SwitchCraft para generar las compuertas lógicas por los métodos FAC, FAC-PD y BDD.
3. Analizar las características eléctricas de las redes generadas mediante un ambiente de simulación en el software Virtuoso Analog Design Enviroment de CADENCE y el simulador Spectre.
4. Analizar las características físicas de las pseudocapas pertenecientes a las redes generadas mediante la herramienta ASTRAN.

Capítulo 2

Fundamentos teóricos

El presente capítulo describe los conceptos necesarios para comprender los términos relacionados a los conmutadores y las redes lógicas. De manera que estas nociones se puedan extrapolar hacia conceptos más elaborados tales como las Compuertas CMOS Estáticas Complementarías (o en inglés Static CMOS Complex Gates).

2.1. Funciones lógicas

Una función lógica Booleana $f(X)$ definida sobre un conjunto de variables $X = \{x_0, x_1, \ldots, x_{n-1}\}$ es una función tal que $f(X) : B_n \rightarrow B$, donde $B = \{0, 1\}$ y $n = |X|$, siendo n el número de variables en X [3]. En estas funciones lógicas se realizan únicamente operaciones con AND, OR y NOT que están denotadas simbólicamente como: "*", "+", "!", respectivamente. Las variables que pertenecen al conjunto X se denominan literales; por ejemplo, dada $f(X) = a * b + !c * d$ los literales de esta función corresponden a $a,\ b,\ !c\ y\ d$. Una función lógica puede representarse como una Suma de Productos (SOP) o Producto de Sumas (POS), conceptos que serán explicado a continuación.

2.1.1. Suma de Productos (SOP) y Producto de Sumas (POS)

2.1.1.1. Mintérminos

Para una función de n variables, un mintérmino es denominado al producto de términos en donde cada una de las n variables aparece al menos una vez [8].

2.1.1.2. Maxtérminos

El complemento de los mintérminos es denominado maxtérminos. Es decir, es la suma de términos en donde cada una de las n variables aparece al menos una vez [8].

2.1.1.3. Suma de Productos

Una función puede ser representada por una expresión compuesta por sumas de mintérminos [8]. A esta forma se le denomina Suma de Productos Estándar. Con respecto a la forma no estándar, basta con que la función lógica posea un producto de términos en el que no aparezcan al menos una vez todas las variables. Para una mejor compresión consideremos las funciones 2.1 y 2.2 que corresponden a una SOP estándar y no estándar, respectivamente.

$$f(a,b,c) = a.b.c + !a.b.c + a.!b.!c \tag{2.1}$$

$$f(a,b,c) = a.b + !a.b.c + a.!b.!c \tag{2.2}$$

2.1.1.4. Producto de Sumas

Una función puede ser representada por una expresión compuesta por producto de maxtérminos [8]. A esta forma se le denomina Suma de Productos Estándar. Con respecto a la forma no estándar, basta con que la función lógica posea una suma de términos en el que no aparezcan al menos una vez todas las variables. Las funciones 2.3 y 2.4 ejemplifican los conceptos de una POS estándar y no estándar, respectivamente.

$$f(a,b,c) = (a+b+c).(!a+!b+!c).(a+!b+!c) \tag{2.3}$$

$$f(a,b,c) = (a+b+c).(!a+!b+!c).(a) \tag{2.4}$$

2.2. Conmutadores lógicos

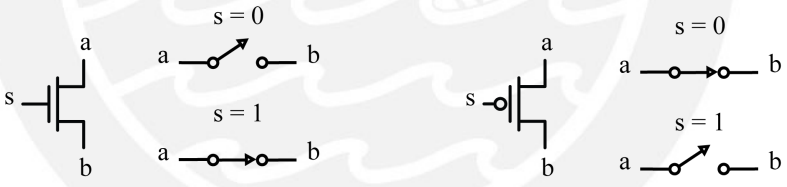
El elemento más básico para implementar redes es el conmutador lógico [9]. Estos elementos son denominados conmutadores directos si conducen cuando se aplica un ‘1’ lógico en su terminal de control, o conmutadores complementarios si conducen cuando su terminal de control está en ‘0’ lógico. Dependiendo de la tecnología utilizada, estos conmutadores pueden implementarse como dispositivos físicos denominados transistores. En la actual tecnología CMOS, están representados por los transistores NMOS y PMOS.

Estos dispositivos con tecnología CMOS poseen tres terminales: Un terminal de control llamado Gate, y dos terminales de contacto Source y Drain. El primer terminal, es el encargado de definir si existe una conexión entre los terminales de contacto Source y Drain. Su respectiva representación simbólica se puede observar en la Figura 2.1.

(a) PMOS. (b) NMOS.

Figura 2.1: Simbología de los transistores PMOS y NMOS.

Ambas configuraciones funcionan con una lógica distinta. En el caso del NMOS, la conexión entre los terminales Drain y Source se da cuando la señal de control está en 1 lógico, es decir es un conmutador directo. Caso contrario ocurre con la configuración PMOS, en donde la conexión de los terminales Drain y Source es dada cuando la señal de control corresponde a un '0' lógico, en otras palabras es un conmutador complementario. Este comportamiento se puede observar en la Figura 2.2.



(a) Lógica del NMOS (b) Lógica del PMOS

Figura 2.2: Conmutación de los transistores NMOS y PMOS.

2.2.1. Características ideales de los conmutadores lógicos

Es importante resaltar unas de las características ideales de estos conmutadores. En el caso del tipo N, cuando se permite la conducción de la corriente por los respectivos terminales, se puede afirmar que si la entrada corresponde a un '0' lógico la salida será un '0' lógico fuerte; caso contrario, cuando la entrada es un '1' lógico la salida será un '1' lógico débil. Análogamente, ocurre con los transistores de tipo P cuando se permite la conducción entre los terminales, si la entrada es un '1' lógico la salida será un '1' lógico fuerte; si la entrada es un '0' lógico, la salida será un '0' lógico débil. Siendo un '0' lógico fuerte el valor de 0 Volts (GND) y un '1' lógico

fuerte el valor de VDD Volts [10]. Este efecto ocurre por el hecho de que la magnitud de la corriente depende de la diferencia de potencial existente entre los terminales Gate y Source. En las Figuras 2.3 y 2.4 se detalla un resumen de lo explicado anteriormente.

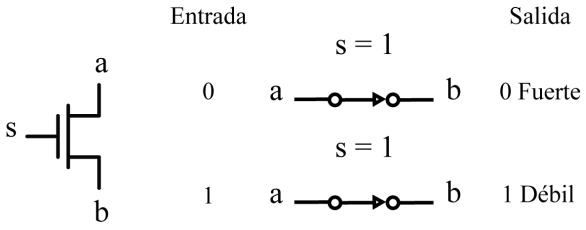


Figura 2.3: Relación entrada y salida del transistor NMOS [10].

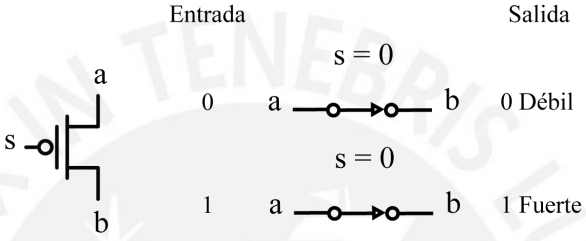


Figura 2.4: Relación entrada y salida del transistor PMOS [10].

2.3. Generación de redes

Con base a lo explicado anteriormente, es posible extrapolar las aplicaciones de estos conmutadores lógicos hacia arreglos para implementar una función lógica Booleana. A manera de comprender este concepto, consideremos los siguientes ejemplos ilustrativos mostrados en la Figura 2.5, los cuales son denominados por la literatura como arreglos basados en ramas (Branch Based).

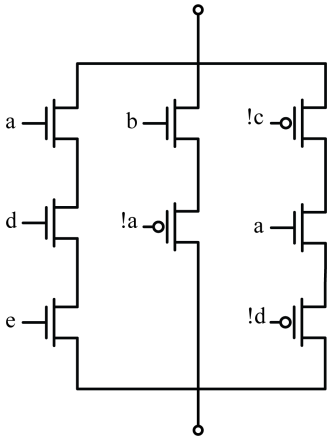


Figura 2.5: Red lógica Branch Based de f=a*d*e+b*!a+!c*a*!d [9].

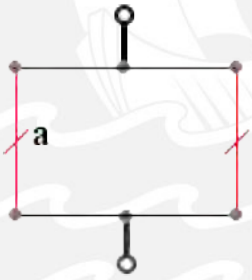
Como es de notar en el circuito ejemplo, una característica de este tipo de arreglos es que los transistores presentan puramente conexiones en serie para realizar una conexión entre dos terminales.

2.3.1. Solución basada en una ecuación

Una función lógica puede implementarse en un arreglo de transistores a partir de los siguientes tres pasos básicos:

- Cada literal de la ecuación representa un conmutador.
- Cada producto de literales, es un arreglo de conmutadores en series. Figura 2.6a.
- Cada suma de literales, es un arreglo de conmutadores en paralelo. Figura 2.6b.

(a) Conmutadores en configuración serie

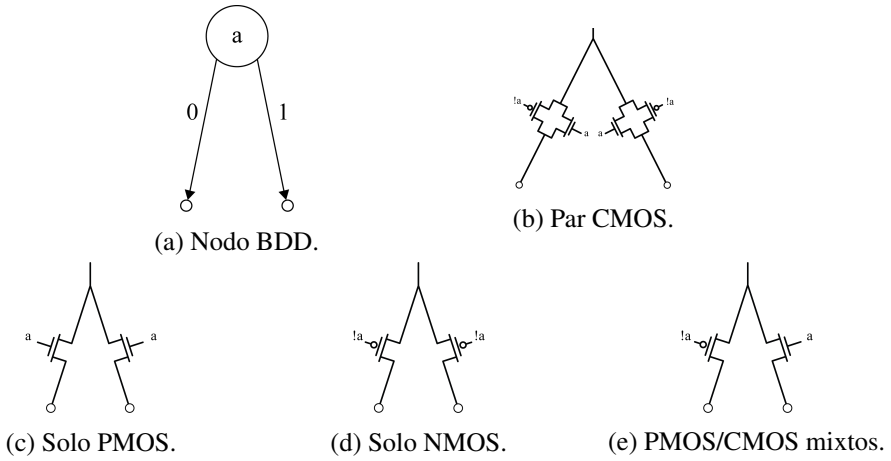


(b) Conmutadores en configuración paralelo

Figura 2.6: Redes lógicas Branch-Based de funciones lógica Booleanas [11].

2.3.2. Solución basada en gráficos

Otra forma de generar una red lógica es a partir de una representación gráfica [11]. A manera de ejemplificar esta solución se considera la representación BDD (Binary Decision Diagram), en la cual se puede obtener una red de transistores por la acción de asignar un conmutador a cada arco de un nodo BDD. Este concepto está ilustrado en la Figura 2.7.



(a) Nodo BDD. (b) Par CMOS.

(c) Solo PMOS. (d) Solo NMOS. (e) PMOS/CMOS mixtos.

Figura 2.7: Soluciones a partir del nodo BDD [12].

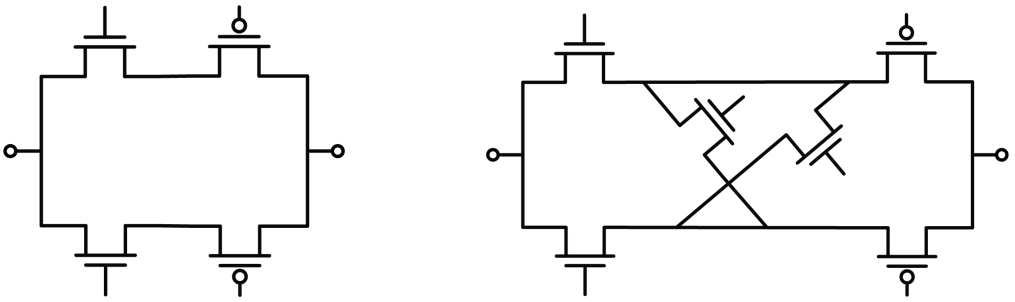
Como se puede visualizar, existen cuatro posibles soluciones con respecto a la asignación de los conmutadores las cuales son: par CMOS, solo NMOS, solo PMOS y PMOS/NMOS mixtos. Una de las características más interesantes de este enfoque es el hecho de que se puede obtener implementaciones serial-paralelo y no-serial-paralelo [12]. Siendo esta última característica utilizada por muchos investigadores, debido a que se pueden obtener redes con una menor cantidad de transistores.

2.4. Propiedades de las redes

Usualmente, cuando se observan redes de dos terminales se pueden destacar las siguientes propiedades: planicidad, serial-paralelo y no-serial-paralelo.

2.4.1. Planicidad y Dualidad

Las redes planas son aquellas que corresponden a un gráfico plano [13]. La representación gráfica de una red de transistores consiste en tomar cada transistor como un borde y cada nodo como un vértice. La gráfica de una red plana se caracteriza por el hecho de que puede ser dibujada en un plano sin que las líneas se crucen. Análogamente, para las redes lógicas, se requiere que los terminales se conecten externamente sin que las conexiones se crucen. En cuanto a la dualidad de una gráfica (y una red lógica), esta una propiedad asociada a la planicidad de la misma, la cual tiene la interesante propiedad de ser lógicamente complementaria. Es decir, considerando que un gráfico G es plano, se puede obtener un gráfico dual $G*$ lógicamente complementario [1]. En la Figura 2.8 se observa un ejemplo de estos conceptos.



(a) (b)

Figura 2.8: Red lógica (a) plana y (b) no plana [9].

2.4.2. Serial-paralelo

Según las asociaciones presentes en una red, se puede clasificar como serial-paralelo si los conmutadores están conectados en serie o en paralelo recursivamente. Como consecuencia, una característica importante de las redes que cumplen con esta propiedad es que también cumplen con la propiedad de planicidad. La Figura 2.9 ejemplifica este concepto.

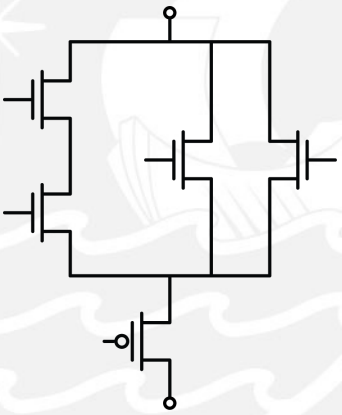


Figura 2.9: Red lógica Serial-paralelo [9].

2.4.3. No-serial-paralelo

Esta es una propiedad que se observa en redes que contienen otra red con una configuración en puente de Wheatstone. Es importante mencionar que aquellas redes que cumplen con esta propiedad pueden ser o no planas. En añadidura, nunca se debe considerar que estas redes poseen un arreglo serial-paralelo [9]. En la literatura, estas redes también son denominadas como Redes Puentes. En la Figura 2.10 se puede visualizar un ejemplo de una red puente.

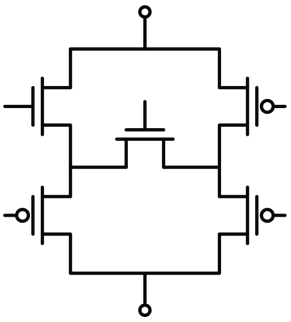


Figura 2.10: Red lógica Puente [9].

Por consiguiente, el investigador Leomar Da Rosa [9] menciona que algunos lemas que pueden derivarse de estas propiedades:

- Lema 1: Todas las redes serial-paralelo son planas.
- Lema 2: Todas las redes planas son duales.
- Lema 3: Todas las redes no planas son redes puentes.
- Lema 4: Las redes no-serial-paralelo pueden o no ser planas.

2.5. Redes complementarias

Los siguientes conceptos aplican para redes lógicas que constan de un par de redes pull-up y pull-down.

2.5.1. Topológicamente complementario

Esta característica se presenta cuando las representaciones gráficas de las redes de pull-up y pull-down son duales [14]. Es importante tener en cuenta que si el gráfico no es plano no es posible obtener un gráfico dual, según lo explicado en la Sección 2.3.1.

2.5.2. Lógicamente complementario

Cuando estrictamente solo una red, ya sea la de pull-up o pull-down, conduce por cada vector de entrada [14].

En la siguiente Figura 2.11 se puede observar un red conformadas por una red de pull-up y pull-down topológicamente y lógicamente complementarios.

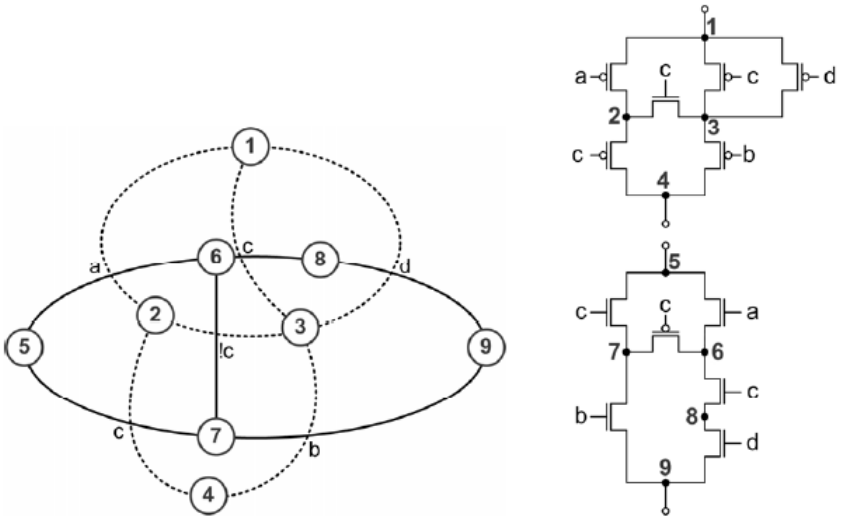


Figura 2.11: Redes duales obtenidas mediante gráficos duales [9].

2.6. Circuitos CMOS Estáticos

Se ha explicado anteriormente que se puede implementar físicamente un función lógica mediante transistores NMOS o PMOS. Un circuito básico corresponde al de un inversor, el cual podemos construir como se muestra en la Figura 2.12.

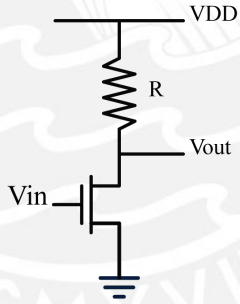
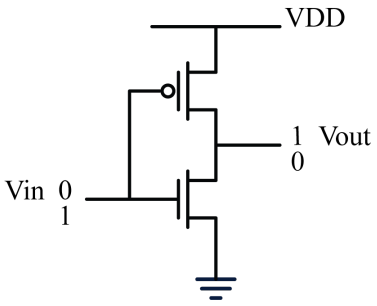


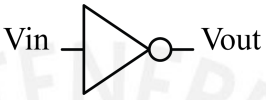
Figura 2.12: Circuito inversor [15].

Si el voltaje V_{in} es menor que el voltaje umbral V_{th} del transistor NMOS, este se encontrará abierto. Por lo tanto, la salida V_{out} es "pulled up" hacia la fuente V_{DD}. Si el voltaje V_{in} es mayor que el voltaje umbral V_{th}, la corriente circula desde la fuente V_{DD} hacia tierra. De esta manera, la salida será el complemento de la entrada [15]. Sin embargo, es de notar que si la resistencia de la linea fuese lo suficientemente grande, para cuando $V_{in} > V_{th}$, el voltaje en la salida correspondería a un valor prácticamente nulo, por lo que el circuito ya no se comporta como un inversor. Es en este punto donde se consideran los circuitos CMOS, los cuales consisten en el uso conjunto de los transistor NMOS y PMOS. Es de conocer que cuando los transistores NMOS y PMOS están conduciendo, la resistencia efectiva entre sus terminales posee valores pequeños. Asimismo,

teniendo en cuenta lo explicado en la Sección 2.2.1, se puede construir un circuito inversor como se demuestra en la Figura 2.13.



(a) Circuito CMOS inversor



(b) Símbolo CMOS inversor

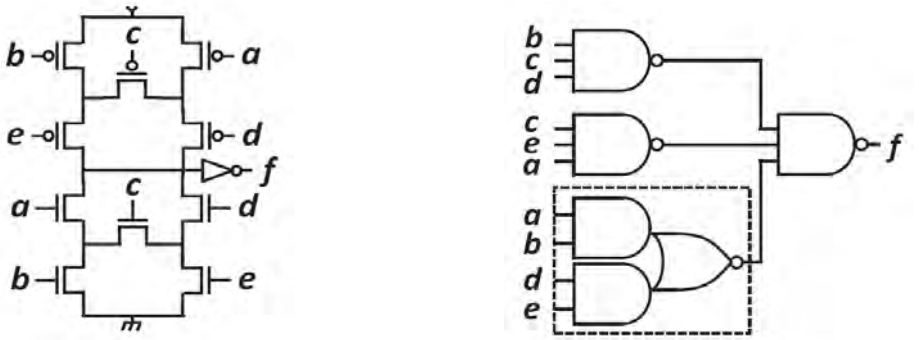
Figura 2.13: CMOS inversor.

Exceptuando el periodo de transición, la salida de una compuerta CMOS estática está ligada a VDD o VSS (GND) a través de un camino de baja resistividad. Asimismo, la salida siempre asumirá el valor de la función Booleana implementada por el circuito. Lo anterior explicado, difiere de la clase de circuito CMOS dinámico, el cual se basa en el almacenamiento temporal de valores de señal en capacitancias de nodo de circuito de alta impedancia [16].

2.7. Compuertas CMOS Estáticas Complementarias

Una compuerta CMOS Estática Complementaria (en inglés Static CMOS Complex Gate) es un dispositivo en el que se puede implementar funciones complejas con diversas entradas utilizando una única compuerta. Uno de los principales objetivos de emplear dicha compuerta es la reducción en la cantidad de transistores que son necesarios para describir una misma función lógica. Para una mejor compresión se puede visualizar la Figura 2.14.

Ambos circuitos representan la misma función lógica $f = a*b+a*c*e+d*e+b*c*d$, el circuito 2.14b se da a través de una solución vía diseño de celdas estándares con 26 transistores y el circuito 2.14a corresponde una supercompuerta que contiene 13 transistores. Básicamente, el circuito 2.14b es diseñado empleando varias compuerta CMOS. Esto se debe a que la mayoría de las librerías, proporcionadas por los proveedores de las herramientas EDA (Electronic design automation), poseen una limitada cantidad de funciones lógicas. Una típica librería de celdas no posee más de 100 diferentes funciones lógicas, al contrastar esta cantidad con las 65,536 posibles



(a) Compuerta SCCG (b) Compuerta CMOS con diseño de celdas estándar

Figura 2.14: Implementación de la función lógica $f = a*b+a*c*e+d*e+b*c*d$ [1]

funciones lógicas, que se pueden obtener para 4 entradas, se puede fácilmente determinar la razón por la cual no se generan compuertas lo más óptimas posible según los requerimientos de diseño. De acuerdo a la literatura, como una alternativa a la metodología de diseño de celdas estándar, se propone un nuevo enfoque de tecnología de mapeo libre de librerías, el cual emplea una gran librería virtual en lugar de una librería de celdas físicas. La optimización de la síntesis lógica se da usando una representación más limitada de las características de la celda, y asumiendo que toda netlist lógica puede ser mapeada sobre el silicio usando una red de transistores en lugar de una red de compuertas [17].

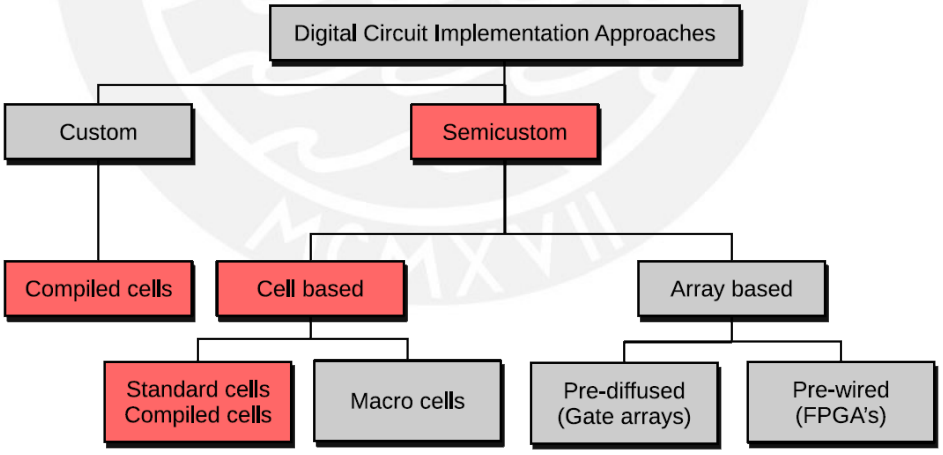


Figura 2.15: Enfoques de implementación para circuitos digitales [17].

Según los investigadores Calebe Oliveira y Ricardo Reis [17], las compuertas SCCG también son denominadas como Celdas Compiladas (Compilled Cells); en añadidura, destacan que desde un punto de vista taxonómico este enfoque de Celdas Compiladas es una mezcla de las mejores características de las metodologías de diseño Semipersonalizadas (Semicustom) y Personalizadas (Custom). La Figura 2.15 ilustra un esquema de los enfoques mencionados.

Capítulo 3

Propuesta de metodología

El presente capítulo tiene como propósito describir las herramientas principales y métodos que se utilizaron para realizar este estudio. SwitchCraft se empleó para generar las compuertas lógicas del conjunto de 3,982 funciones booleanas de clase P de 4 entradas. Con ASTRAN se obtuvo los respectivos layouts de las funciones lógicas considerando las reglas de diseño de la tecnología TSMC 180 nm. El estudio está dirigido a realizar una comparación de las características eléctricas y físicas de las redes generadas a través métodos FAC, FAC-PD y BDD. Estos métodos de generación corresponden a los denominamos por el framework de SwitchCraft como: Direct from expression (FAC-PD), Factorization method (FAC) y BDD - minimum stacks (BDD).

El análisis eléctrico ha sido realizado por simulaciones en el software Virtuoso Analog Design Enviroment de CADENCE con uso del simulador *Spectre* utilizando la tecnología TSMC 180 nm. Con este análisis se ha pretendido medir la potencia promedio disipada para los tres métodos mencionados.

De igual manera, en el análisis de las características físicas se considera la tecnología de transistores TSMC 180 nm. Su objetivo consistió en medir el área total activa de los layout de los circuitos, el número de transistores y las longitudes de las secciones de polisilicio empleadas.

3.1. Diagrama de flujo

Como se puede visualizar en la Figura 3.1, el estudio inicia con el ingreso del conjunto de funciones de clase P a la herramienta SwitchCraft. Para cada una de las funciones lógicas, se crea su respectiva compuerta lógica seleccionando uno de los tres métodos de generación: Direct from expression (FAC), Factorization method (FAC-PD) o BDD-minimum stacks (BDD). Posteriormente, se imprime la descripción del circuito (Netlist) en formato *Spice* utilizando un módulo de la herramienta.

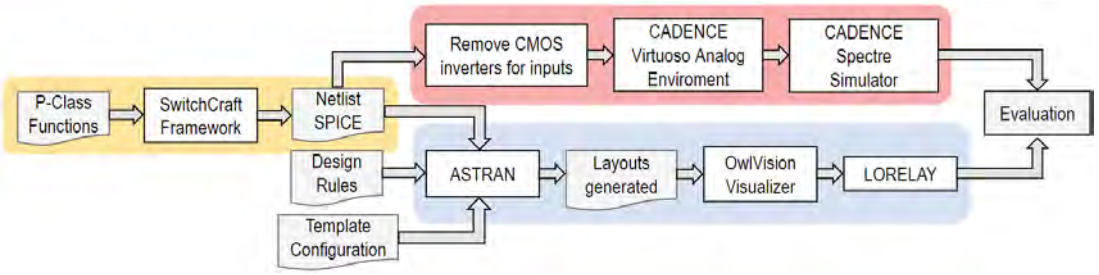


Figura 3.1: Diagrama de flujo propuesto para el estudio.

Para la medición de los parámetros eléctricos es necesario realizar modificaciones en las netlist de las supercompuertas generadas. Esta modificación consiste en retirar de las netlist las lineas que describen los inversores CMOS para las entradas. El motivo específico se detalla en futuras secciones. Posteriormente, una vez realizados los cambios, se inicia un proceso automatizado para la generación de los circuitos de las compuertas SCCG en el software de CADENCE.

Con respecto al estudio físico, la descripción de los circuitos en formato *Spice* se ingresan junto a las reglas de diseño de la tecnología TSMC 180 nm y una configuración de plantilla hacia la herramienta ASTRAN. Estos archivos de entradas pasan por un proceso de síntesis automático para generar los layouts de los circuito en el formato GDSII. Por ultimo, los layouts obtenidos son procesados a un formato ASCII por la herramienta OwlVision. Dicho formato nos permitirá obtener la información precisa del área total y la longitud de los polisilicios a través de la herramienta LORELAY desarrollada en este trabajo.

Este proceso de análisis físico y eléctrico se aplicará para los tres métodos anteriormente aludidos. Finalmente, se realizará una comparativa para determinar el método que posea mejores características físicas y eléctricas.

3.2. Selección del conjunto de funciones lógicas

Un elemento principal que se debe considerar para generar una red lógica es la función booleana que se desea implementar. Para n entradas se puede obtener hasta 2^{2^n} funciones lógicas posibles. Por consiguiente, con la finalidad de realizar un estudio robusto, es necesario considerar un conjunto amplio de funciones. De esta manera, se seleccionó el grupo de 65,536 funciones lógicas posibles que se obtienen para 4 entradas. Ciertamente es posible reducir la cantidad de funciones de este conjunto sin que la robustez del estudio se vea comprometida. Esto se logra por el concepto de funciones de equivalencia P, el cual consiste en la permutación del orden de las señales de entradas que permiten identificar las funciones que poseen una misma respuesta. En consecuencia, se consigue una reducción de 65,536 a 3,982 funciones lógicas [18].

3.3. Generación de las compuertas lógicas

3.3.1. Framework SwitchCraft

SwitchCraft es una herramienta que fue desarrollada por investigadores de la Universidade Federal do Rio Grande do Sul con el objetivo de proporcionar un conjunto de algoritmos y métodos que ayuden a los diseñadores a generar redes lógicas [6]. En el Capítulo 1, se destacó que SwitchCraft cuenta con 4 módulos para la generación de las compuertas SCCG. No obstante, cabe mencionar que para esta sección el enfoque del estudio estará direccionado a los siguientes módulos de la herramienta:

- 1er. módulo: Información de entrada.
- 2do. módulo: Generación de red de transistores.
- 4to. módulo: Visualizador del circuito generado.

El primer módulo, permitirá crear en el software el conjunto de funciones de clase P de 4 entradas, a través del comando ***create_expression [nombre_función] [expresión_booleana]***. En la Figura 3.2 se puede observar la ejecución de este comando.

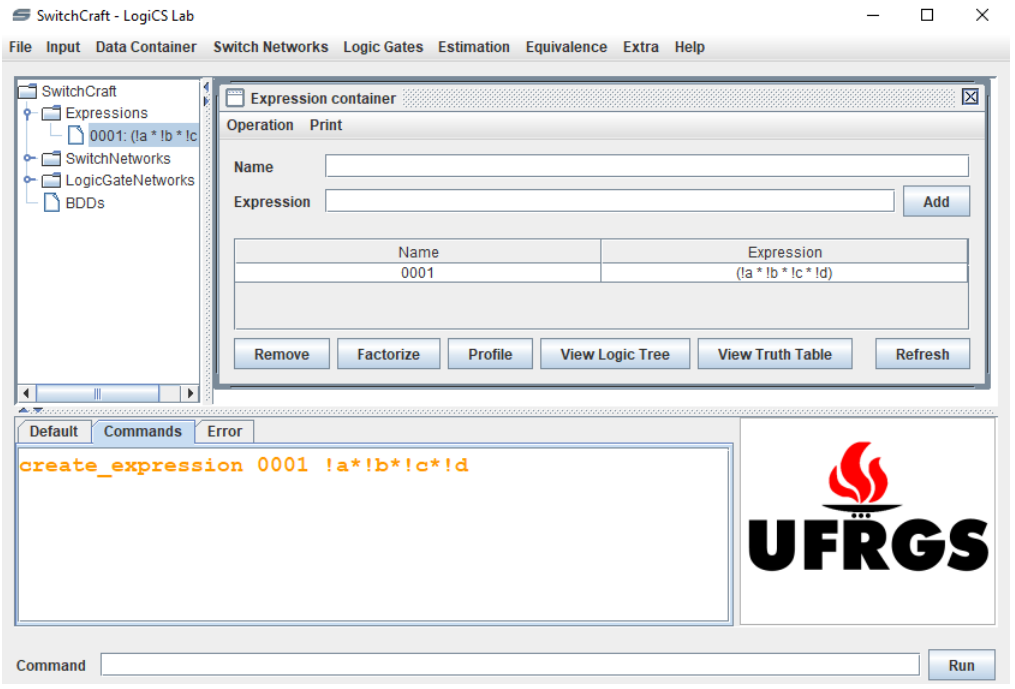


Figura 3.2: Expresión *!a*!b*!c*!d* creada en SwitchCraft.

El segundo módulo, se encarga de generar las compuertas lógicas utilizando la información de las funciones booleanas ingresadas. Según el método que se desee aplicar para generar las redes lógicas el comando a utilizar varía:

- FAC: *create_logic_gate_network_from_factorization [nombre_función] [nombre_compuerta]*
- FAC-PD: *create_logic_gate_network_from_direct_expression [nombre_función] [nombre_compuerta]*
- BDD: *create_logic_gate_network_from_lbbdd [nombre_función] [nombre_compuerta]*

En la Figura 3.3 se contempla la ejecución del comando para generar la compuerta de la primera función por el método FAC-PD.

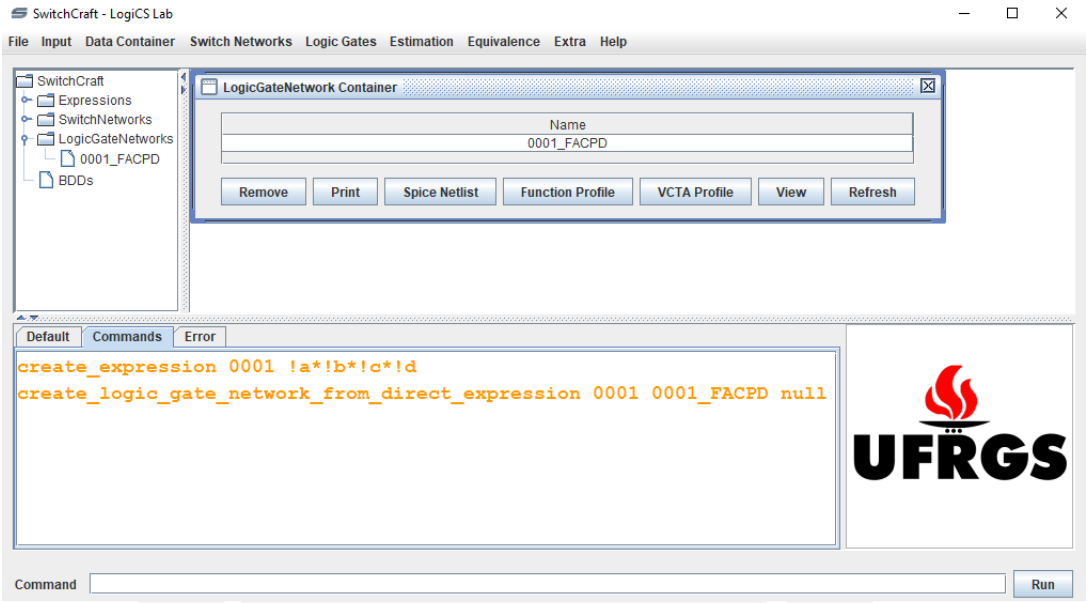


Figura 3.3: Compuerta lógica de *!a*!b*!c*!d* generada por FAC-PD.

El cuarto módulo nos permite obtener una vista del diagrama circuital y la descripción del circuito en formato *Spice*. Para ello se debe ingresar a la ventana *LogicGateNetwork Container* y seleccionar *View* y *Spice Netlist*, respectivamente. Asimismo, también es posible usar el comando *print_flat_spice_from_logic_gate [nombre_compuerta] vdd gnd pu pd*. En la Figura 3.4 se visualiza el circuito y la netlist para la primera función generada por el método FAC-PD.

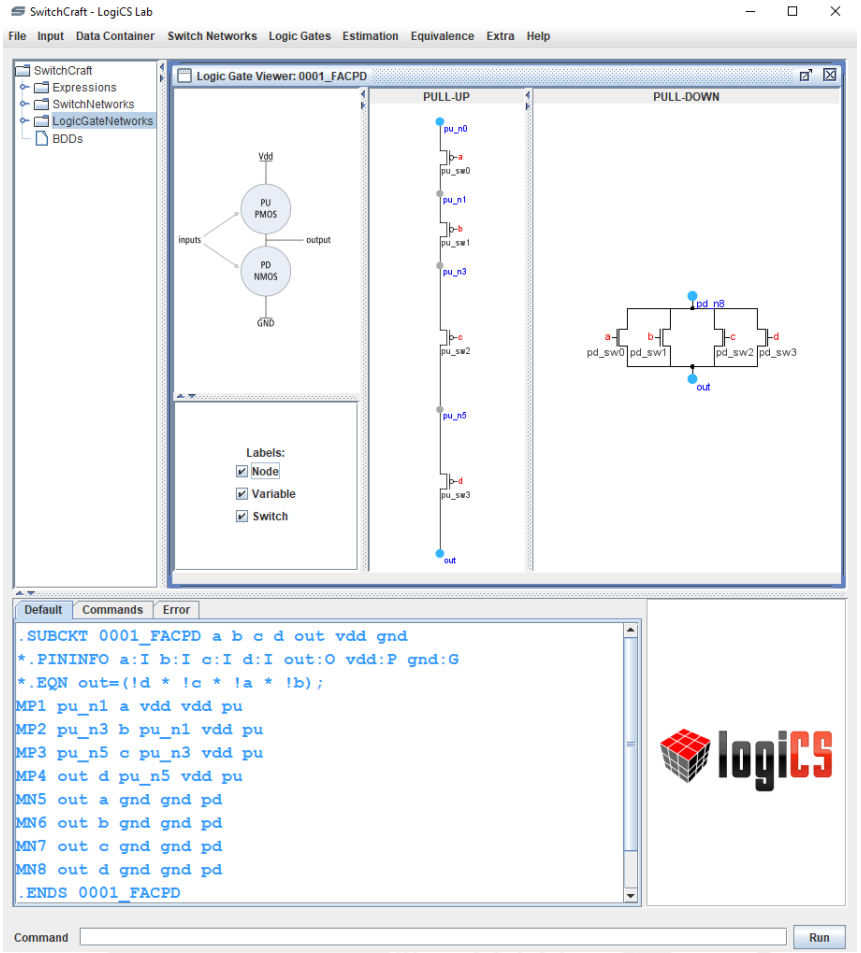


Figura 3.4: Diagrama circuital y *Spice* Netlist de la compuerta lógica de !a*!b*!c*!d generada por FAC-PD.

3.3.2. Automatización del proceso de generación de compuertas lógicas

En la Sección 3.2 se mencionó que se analiza un conjunto de 3,982 funciones para cada método; por consiguiente, resulta necesario plantear un proceso de automatización. SwitchCraft permite scripts que contenga listados los comandos a ejecutar. Por ello, teniendo en cuenta que el proceso es iterativo se desarrollaron scripts en Python que listan los comandos para la creación de las funciones, la generación de las compuertas lógicas y la impresión de las netlists los circuitos.

```
archivo = open('function4pClass.csv','r')
contenido = archivo.read()
archivo.close
lista_contenido = contenido.split("\n")
datos = []

for linea in lista_contenido:
```

```
    lineas_contenido = linea.split(",") #A cada elemento del vector linea de
    lista_contenido lo separamos por "," en otro elementos
    row = []                            #Vector vacio
    i = 0
    for item in lineas_contenido:
      if i == 1:
        row.append(item)              #Anexamos cada elemento del vector lineas
     en el vector row
      else:
        i = i+1
    datos.append(row)
  num = []
  for i in range(3983):
    if i < 10:
      num.append('000' + str(i))
    elif i >= 10 and i < 100:
      num.append('00' + str(i))
    elif i >= 100 and i < 1000:
      num.append('0' + str(i))
    elif i>=1000:
      num.append(str(i))

  textfile = open("script_input_logic_functions.txt", "w")
  j = 0
  for element in datos[1:-1]:
      j = j+1
      textfile.write("create_expression " + num[j] + " " + str(element)[2:-2] +
     '\n')
  textfile.close()
```

Listing 3.1: Script Python para ingresar las funciones booleanas a SwitchCraft.

Una fragmento de lo que imprime el script Listing 3.1 para ingresar las funciones lógicas corresponde a lo mostrado en la Figura 3.5.

```
create_expression 0001 !a*!b*!c*!d
create_expression 0002 !a*!b*!c*d
create_expression 0003 !a*!b*!c
create_expression 0004 !a*!b*!c*d+!a*!b*c*!d
create_expression 0005 !a*!b*!c+!a*!b*!d
create_expression 0006 !a*!b*c*d
create_expression 0007 !a*!b*!c*!d+!a*!b*c*d
create_expression 0008 !a*!b*d
create_expression 0009 !a*!b*!c+!a*!b*d
create_expression 0010 !a*!b*d+!a*!b*c
```

Figura 3.5: Porción de lo impreso por el script Listing 3.1

```
num = []
for i in range(3983):
  if i < 10:
    num.append('000'+str(i))
  elif i >= 10 and i < 100:
    num.append('00'+str(i))
  elif i >= 100 and i < 1000:
    num.append('0'+str(i))
  elif i >= 1000:
    num.append(str(i))

#Listado de los comandos para generar las compuertas SCCG
textfile = open("script_logic_gate_fac_gen.txt", "w")
for i in range(1,3983):
    textfile.write("create_logic_gate_network_from_direct_expression " + num[
  i] + " " + num[i] + "_FAC" + " null" + '\n')
textfile.close()

textfile = open("script_logic_gate_facpd_gen.txt", "w")
for i in range(1,3983):
    textfile.write("create_logic_gate_network_from_factorization " + num[i] +
   " " + num[i] + "_FACPD" + '\n')
textfile.close()

textfile = open("script_logic_gate_bdd_gen.txt", "w")
for i in range(1,3983):
    textfile.write("create_logic_gate_network_from_lbbdd " + num[i] + " " +
  num[i] + "_BDD" + '\n')
textfile.close()

#Listado de los comando para imprimir las netlists
textfile = open("script_netlist_fac_gen.txt", "w")
for i in range(1,3983):
    textfile.write("print_flat_spice_from_logic_gate " + num[i] + "_FAC " + "
  vdd gnd < >" + '\n')
textfile.close()

textfile = open("script_netlist_facpd_gen.txt", "w")
for i in range(1,3983):
    textfile.write("print_flat_spice_from_logic_gate " + num[i] + "_FACPD " +
   "vdd gnd < >" + '\n')
textfile.close()
```

```
38
39  textfile = open("script_netlist_bdd_gen.txt", "w")
40  for i in range(1,3983):
41      textfile.write("print_flat_spice_from_logic_gate " + num[i] + "_BDD " + "
      vdd gnd < >" + '\n')
42  textfile.close()
```

Listing 3.2: Script Python para generar las compuertas SCCG e imprimir sus respectivas netlists.

Asimismo, en la Figura 3.6 y Figura 3.7 se visualiza un fragmento de lo que se obtiene del script desarrollado en Listing 3.2.

```
1   create_logic_gate_network_from_lbbdd 0001 0001_BDD
2   create_logic_gate_network_from_lbbdd 0002 0002_BDD
3   create_logic_gate_network_from_lbbdd 0003 0003_BDD
4   create_logic_gate_network_from_lbbdd 0004 0004_BDD
5   create_logic_gate_network_from_lbbdd 0005 0005_BDD
6   create_logic_gate_network_from_lbbdd 0006 0006_BDD
7   create_logic_gate_network_from_lbbdd 0007 0007_BDD
8   create_logic_gate_network_from_lbbdd 0008 0008_BDD
9   create_logic_gate_network_from_lbbdd 0009 0009_BDD
10  create_logic_gate_network_from_lbbdd 0010 0010_BDD
```

Figura 3.6: Fragmento de lo obtenido por el script Listing 3.2 para generas las compuertas SCCG el método BDD.

```
1   print_flat_spice_from_logic_gate 0001_BDD vdd gnd pu pd
2   print_flat_spice_from_logic_gate 0002_BDD vdd gnd pu pd
3   print_flat_spice_from_logic_gate 0003_BDD vdd gnd pu pd
4   print_flat_spice_from_logic_gate 0004_BDD vdd gnd pu pd
5   print_flat_spice_from_logic_gate 0005_BDD vdd gnd pu pd
6   print_flat_spice_from_logic_gate 0006_BDD vdd gnd pu pd
7   print_flat_spice_from_logic_gate 0007_BDD vdd gnd pu pd
8   print_flat_spice_from_logic_gate 0008_BDD vdd gnd pu pd
9   print_flat_spice_from_logic_gate 0009_BDD vdd gnd pu pd
10  print_flat_spice_from_logic_gate 0010_BDD vdd gnd pu pd
```

Figura 3.7: Fragmento de lo obtenido por el script Listing 3.2 para imprimir las netlist de las compuertas generadas por el método BDD.

3.4. Análisis de las características eléctricas de las compuertas lógicas

3.4.1. Entorno de simulación

Para la simulación del comportamiento eléctrico de los circuitos se utiliza el software *Virtuoso Analog Enviroment* de CADENCE con el simulador *Spectre*. En este estudio se seleccionó la tecnología de transistores TSMC de 180 nm y se definieron los estímulos de las entradas como fuentes pulsos que permiten hacer un barrido de la tabla de la verdad. En la Figura 3.8 se observa el entorno propuesto.

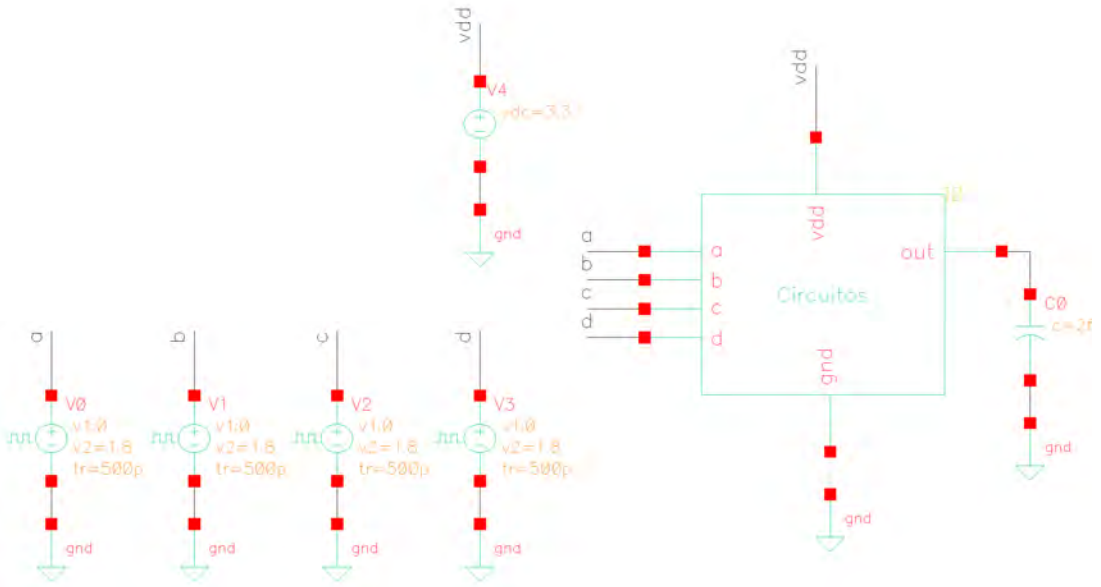


Figura 3.8: Entorno de simulación propuesto para el análisis eléctrico

3.4.2. Medición de parámetros eléctricos

El rendimiento de un circuito CMOS se puede determinar por diversos parámetros como la potencia promedio que consume. En el ámbito del diseño del circuitos VLSI, es importante que los circuitos consuman la menor potencia posible. De esta forma la energía disipada en forma de calor influye menos en el rendimiento. Con la finalidad de realizar un aporte y una comparativa con los estudios manifestados en el estado del arte, la métrica a medir en el análisis eléctrico es la potencia promedio que consumen las compuertas.

La potencia promedio consumida se calcula mediante las siguientes ecuaciones:

$$P_{promedio} = \frac{\int_0^T p(t).dt}{T} \tag{3.1}$$

$$p(t) = V_{DD}.i(t) \tag{3.2}$$

V_{DD}**:** Voltaje de la fuente de alimentación de la compuerta lógica.

$i(t)$**:** Corriente instantánea que provee la fuente de alimentación.

A manera de ejemplificar la medición de este parámetro, se toma en cuenta el entorno de simulación propuesto en la Figura 3.8 y el esquema del circuito de la Figura 3.9.

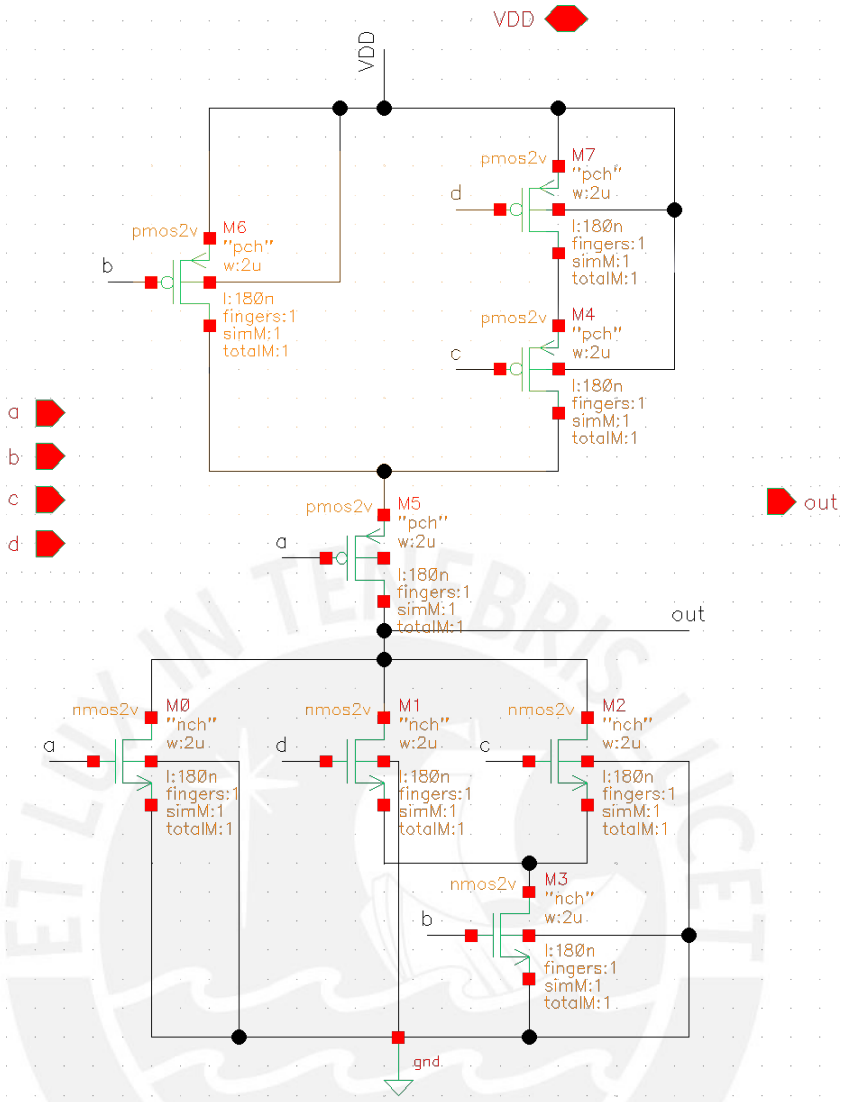


Figura 3.9: Esquemático de la compuerta lógica de *!a*!b+!a*!c*!d* generada por el método FAC-PD.

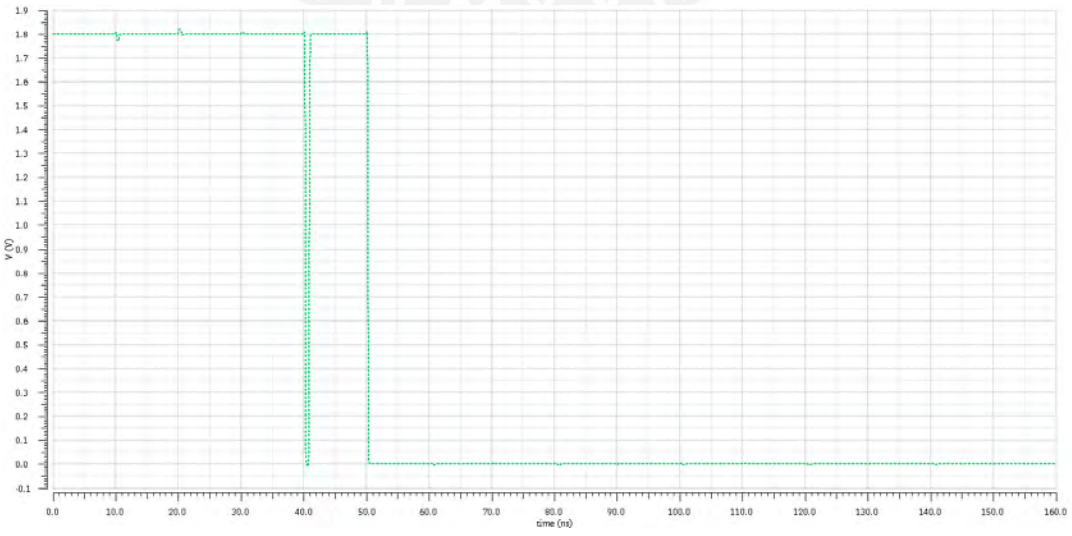


Figura 3.10: Respuesta transitoria de *i(t)* que entrega la fuente de alimentación V_{DD}.

La respuesta transitoria de la corriente *i(t)* que entrega la fuente de alimentación V_{DD} se puede observar en la Figura 3.10. Por otra parte, el software *Virtuoso Analog Design Enviroment* permite ingresar funciones matemáticas que utilizan como variables las señales, parámetros de voltaje o corriente del circuito. Lo cual nos permite realizar el cálculo de la integral de la Ecuación 3.1 y obtener el deseado valor de la potencia disipada.

3.4.3. Automatización de las simulaciones

En virtud de que el estudio eléctrico consta de evaluar 3,982 compuertas lógicas para cada método, resulta indispensable en plantear un proceso de medición automatizado. Teniendo en cuenta que los estímulos de entrada, la tecnología de transistores y la fuente de alimentación son los mismos para todos los circuitos es posible crear un proceso de medición en serie.

En primer lugar, se generaron los esquemáticos de los circuitos en el software de CADENCE. Para ello, se realizaron modificaciones en las netlists de los circuitos obtenidos por SwitchCraft, los cuales están descritos en un formato *Spice* como se muestra a continuación:
$< nombre\_componente > < nodo_1 > < nodo_2 > \ ... < nodo_n > < nombre\_modelo >$

- <*nombre_componente*>: Nombre del componente.
- <*nodo_n*>: Nodos del circuito o terminales del componente.
- <*nombre_modelo*>: Nombre del modelo que utiliza el componente.

Una primera modificación consistió en identificar los componentes de los circuitos y asociarlos con su respectivo modelo NMOS o PMOS. Además, cabe resaltar que las netlists de las compuertas SCCG generadas por SwitchCraft poseen circuitos inversores que no son objetivo para este análisis. El motivo es explicado en el Capítulo 4. Por lo mencionado, se identificaron las compuertas con inversores y se eliminaron de su respectiva descripción para ser reemplazadas por fuentes pulsos que emulen el comportamiento inversor. Asimismo, se agregaron a dichas descripciones una conexión a un componente *c_cap*, el cual corresponde a la carga capacitiva de cada circuito.

Para lograr la modificaciones expuestas, se desarrollaron scripts en Python para editar la descripción de los 11,946 subcircuitos correspondientes a los tres métodos evaluados. Estos scripts se pueden observar en las Listing 3.3, 3.4.

```
a_file = open("netlist_fac.txt", "r")
list_of_lines = a_file.readlines()
list_of_lines_able = list_of_lines
a_file.close()
```

```
5
6  list_of_lines_modified = []
7  subckt_line_number = []  # En este arreglo se guarda en que numero de linea se
       encuentra el inicio de un subcircuito
8  ends_line_number = []  # En este arreglo se guarda en que numero de linea se
       encuentra el fin de un subcircuito
9  ckt_names = []  # En este arreglo se guardan los nombres de cada subcircuitos
10 inverters_per_subckt_line_number = [0 for y in range(3982)]  # Se listan las
       lineas donde inician la descripcion de los
11 # inversores para cada uno de los 3982 SUBCKT
12 inverters_in_subckt = [[0 for x in range(5)] for y in range(3982)]  # Se listan
        los entradas invertidas para cada uno de
13 # los 3982 SUBCKT
14 for index,line in enumerate(list_of_lines):
15     if line[:7] == ".SUBCKT":
16         subckt_line_number.append(index)
17     if line[:5] == ".ENDS":
18         ends_line_number.append(index)
19         ckt_names.append(line[6:14])
20
21 for j in range(3982):
22     # En el primer for anidado, se identifican los inversores utilizados en
       cada SUBCKT.
23     inverter_in_subckt_count = 0
24     for k in range(subckt_line_number[j], ends_line_number[j]):
25         if list_of_lines[k][:3] == "MP_":
26             words_in_line = list_of_lines_able[k].split(" ")
27             inverters_in_subckt[j][inverter_in_subckt_count] = words_in_line[1]
28             inverter_in_subckt_count = inverter_in_subckt_count + 1
29     # En el segundo for, se halla la linea en la que empiezan las descripciones
        de los inversores, es decir hasta
30     # donde termina la descripcion de los planos de PU y PD del SCCG. Por el
       break utilizado, se separa del for anterior.
31     for k in range(subckt_line_number[j], ends_line_number[j]):
32         if list_of_lines[k][:3] == "MP_":
33             inverters_per_subckt_line_number[j] = k
34             break  # Se emplea el break porque hay circuitos que tienen mas de
       un inversor y se perderia la linea donde inicia el primer inversor
35
36 # Con las siguientes iteraciones, y ya conociendo las posiciones de lineas de:
       Inicio del SUBCKT, Inicio Descripcion
37 # Inversores y Fin de un SUBCKT, para cada uno de los 3982 circuitos; se
```

```
    procede a generar las lineas modificadas.
for j in range(3982):
    # If para circuitos que no poseen inversores. Por defecto, los circuitos que no tienen inversores tienen 0 como valor en el array inverters_per_subckt_line_number
    if inverters_per_subckt_line_number[j] == 0:
        for x in range(subckt_line_number[j], ends_line_number[j]):
            list_of_lines_modified.append(list_of_lines_able[x])
        list_of_lines_modified.append("JAIRc out gnd c_cap\n")  # Agregamos la carga capacitiva
        list_of_lines_modified.append(list_of_lines_able[ends_line_number[j]])
        list_of_lines_modified.append("\n")
    else:   # Si se identifica un inversor en el circuito
        for x in range(subckt_line_number[j], inverters_per_subckt_line_number[j]):  # Realizamos la descripcion de SCCG
            list_of_lines_modified.append(list_of_lines_able[x])
        for inversor_available in inverters_in_subckt[j]:  # Se describen los respectivos inversores
            if inversor_available != 0 and inversor_available == "not_out":
                list_of_lines_modified.append("MP_invOut not_out out vdd vdd PMOS L=180nm W=2000nm\nMN_invOut not_out out gnd gnd NMOS L=180nm W=2000nm\n")
                break
            elif inversor_available != 0 and inversor_available != "not_out":
                list_of_lines_modified.append("JAIRinv" + inversor_available[3:5] + ' ' + inversor_available + ' ' + "gnd invk" + inversor_available[3:5] + '\n')
        if "not_out" in inverters_in_subckt[j]:
            list_of_lines_modified.append("JAIRc not_out gnd c_cap\n")  # Agregamos la carga capacitiva
        else:
            list_of_lines_modified.append("JAIRc out gnd c_cap\n")  # Agregamos la carga capacitiva
        list_of_lines_modified.append(list_of_lines_able[ends_line_number[j]])  # Se agrega el respectivo .ENDS
        list_of_lines_modified.append("\n")  # Linea nueva

a_file = open("netlist_no_inverters.txt", "w")
a_file.writelines(list_of_lines_modified)
a_file.close()
```

Listing 3.3: Script Python para asociar los componentes MP y MN con sus respectivos modelos.

Las modificaciones en las netlists después de ejecutar los scripts se puede ver reflejado en la Figura 3.11. Se tomó como ejemplo el circuito de la función 2,596 generado por el método BDD.

```
.SUBCKT 2596_BDD a c d not_out vdd gnd
*.PININFO a:I c:I d:I not_out:O vdd:P gnd:G
*.EQN not_out=!(!d + (a * !c));
MP1 out not_a pu_n1 vdd PMOS L=180nm W=2000nm
MP2 out d vdd vdd PMOS L=180nm W=2000nm
MP3 pu_n1 c vdd vdd PMOS L=180nm W=2000nm
MN4 out not_a pd_n1 gnd NMOS L=180nm W=2000nm
MN5 pd_n1 d gnd gnd NMOS L=180nm W=2000nm
MN6 out c pd_n1 gnd NMOS L=180nm W=2000nm
MP_inv7 not_a a vdd vdd PMOS L=180nm W=2000nm
MN_inv8 not_a a gnd gnd NMOS L=180nm W=2000nm
MP_inv9 not_out out vdd vdd PMOS L=180nm W=2000nm
MN_inv10 not_out out gnd gnd NMOS L=180nm W=2000nm
.ENDS 2596_BDD
```

(a) Netlist inicial.

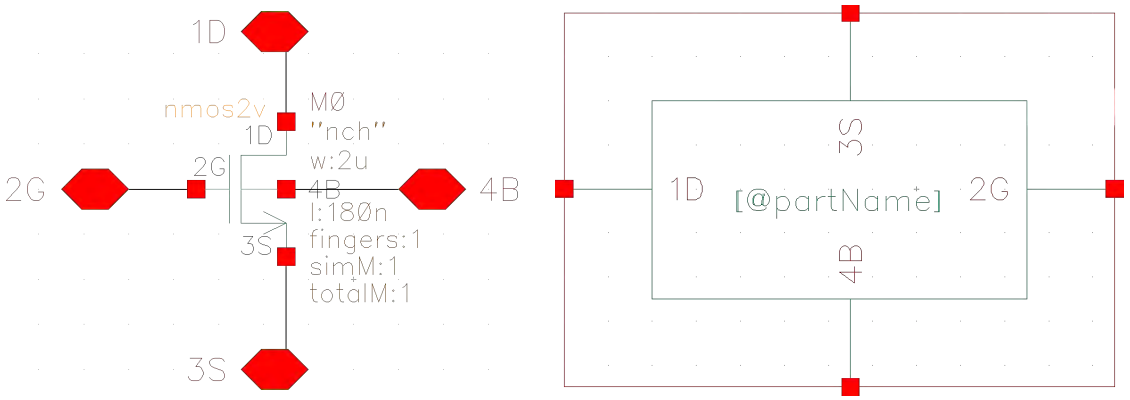
```
.SUBCKT 2596_BDD a c d not_out vdd gnd
*.PININFO a:I c:I d:I not_out:O vdd:P gnd:G
*.EQN not_out=!(!d + (a * !c));
MP1 out not_a pu_n1 vdd PMOS L=180nm W=2000nm
MP2 out d vdd vdd PMOS L=180nm W=2000nm
MP3 pu_n1 c vdd vdd PMOS L=180nm W=2000nm
MN4 out not_a pd_n1 gnd NMOS L=180nm W=2000nm
MN5 pd_n1 d gnd gnd NMOS L=180nm W=2000nm
MN6 out c pd_n1 gnd NMOS L=180nm W=2000nm
JAIRinv_a not_a gnd invk_a
MP_invOut not_out out vdd vdd PMOS L=180nm W=2000nm
MN_invOut not_out out gnd gnd NMOS L=180nm W=2000nm
JAIRc not_out gnd c_cap
.ENDS 2596_BDD
```

(b) Netlist con modificaciones.

Figura 3.11: Resultado de las modificaciones de las netlists.

El software CADENCE acepta como formatos de entradas circuitos descritos en formato *Spice*. Por lo tanto se utilizaron las netlists desarrolladas para generar el esquemáticos de los circuitos.

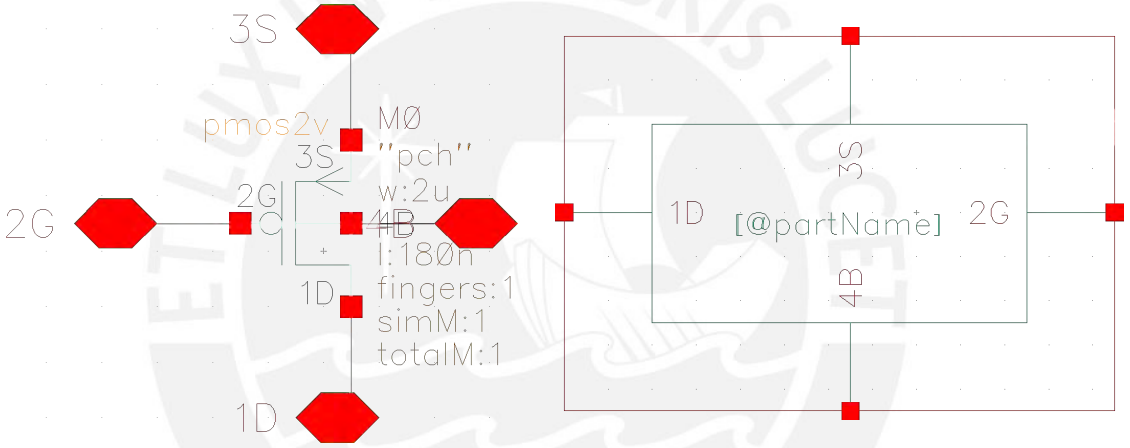
Como se puede observa en las netlists de la Figura 3.11 se utilizan las celdas PMOS y NMOS para describir las redes de pull-up y pull-down, respectivamente. Cabe mencionar que no se emplean directamente los transistores de la librería *tsmc18* porque no cuentan con los permisos para editar la librería y añadir dentro de esta un nuevo esquemático. Por consiguiente, se crearon en una nueva librería las celdas PMOS y NMOS con sus respectivos terminales de entradas y salidas, y utilizando los transistores de la librería *tsmc18*. En las Figuras 3.12 y 3.13, se pueden observar detalladamente las celdas creadas con sus respectivos símbolos asociados.



(a) Esquemático para el transistor NMOS.

(b) Símbolo para el transistor NMOS.

Figura 3.12: Celda PMOS.

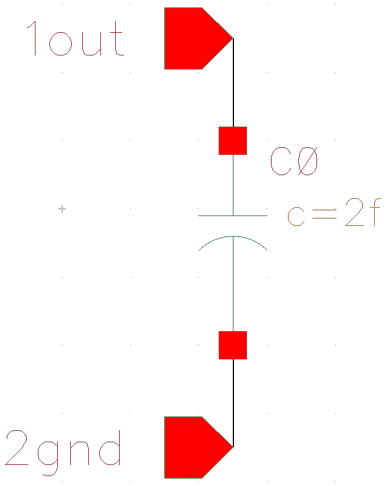


(a) Esquemático para el transistor PMOS.

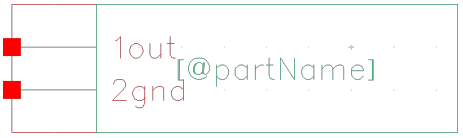
(b) Símbolo para el transistor PMOS

Figura 3.13: Celda PMOS.

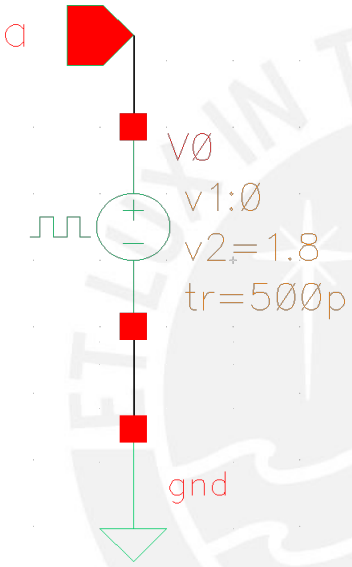
Análogamente, se crearon celdas para los componente *c_cap* y para las fuentes pulsos que estimulan los circuitos. En la Figura 3.14, se puede visualizar las nuevas celdas creadas.



(a) Esquemático para el capacitor.

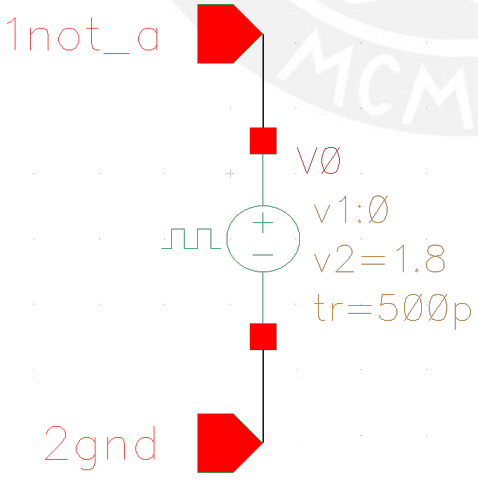


(b) Símbolo para el capacitor.

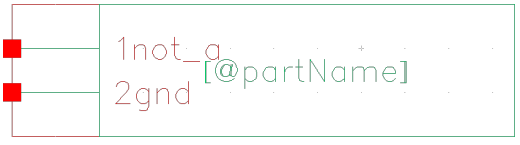


(c) Esquemático para la fuente pulso A.

(d) Símbolo para la fuente pulso A.



(e) Esquemático para la fuente pulso negada de A.



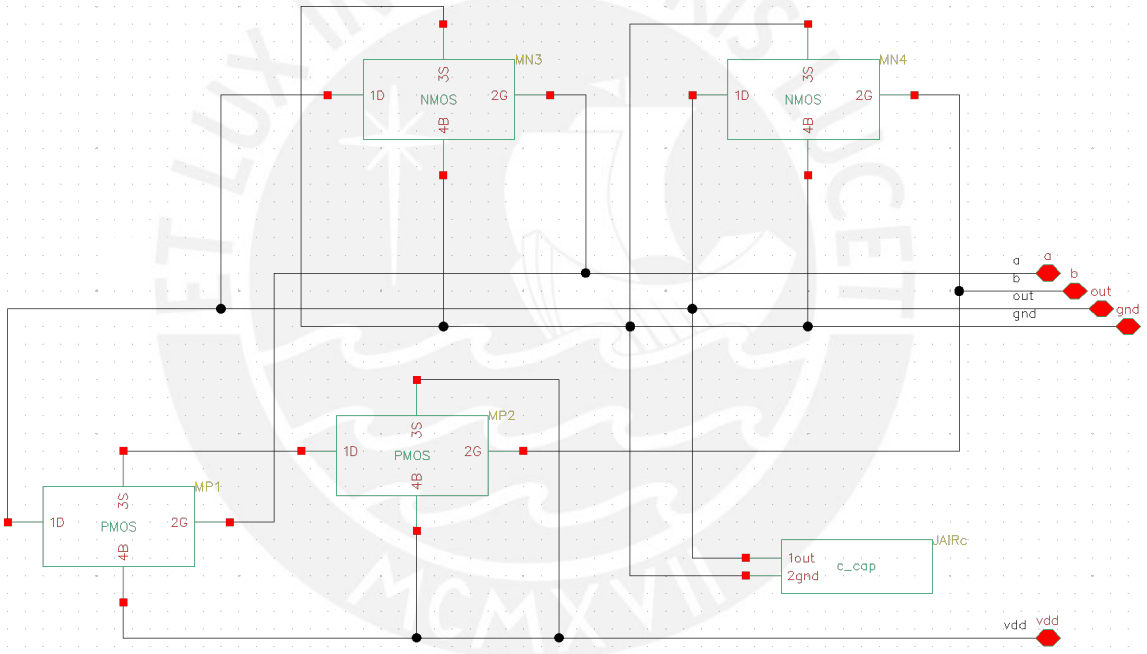
(f) Símbolo para la fuente pulso negada de A.

Figura 3.14: Celdas *c_cap*, *A* e *invk_A*.

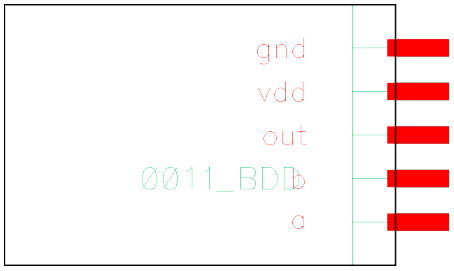
Cabe mencionar que en la Figura 3.14 no se reflejan todas las fuentes pulsos utilizadas, ya que las demás son análogas a las fuentes que se muestran en (c), (d), (e) y (f).

Al importar las netlist al CADENCE se obtiene el esquemático y su respectivo símbolo como se muestra en la Figura 3.15. Una primera ventaja de emplear este procedimiento para crear los esquemáticos de los circuitos es que se pueden generar de una manera automatizada. La segunda ventaja consiste en aplicar nuevamente este método pero esta vez para generar un esquemático que contenga los símbolos de los 3,982 circuitos. A este método nos referiremos como *"Port mapping"* en esta literatura.

Con el gran conjunto de circuitos obtenido es posible optar por alimentar todas las compuertas con una única fuente V_{DD}. Es decir, con medir la corriente que entrega la fuente se logra el objetivo de determinar la potencia promedio que se consume al aplicar la Ecuación 3.1.



(a) Esquemático.



(b) Símbolo.

Figura 3.15: Celda de la compuerta generada por BDD de la función *!a*!b*

Para lograr la juntura de las 3,982 compuertas lógicas es necesario añadir las siguientes líneas de código del Listing 3.5 en el script del Listing 3.4.

```
# Port map de los circuitos
list_port_map_lines = []
for j in range(3982):
    if j == 500:
        b_file = open("netlist_no_inverters1.txt", "w")
        b_file.writelines(list_of_lines_modified_final)
        b_file.writelines('\n')
        list_port_map_lines.append("BV0 a A\nBV1 b B\nBV2 c C\nBV3 d D\n.END")
        b_file.writelines(list_port_map_lines)
        list_port_map_lines = []
        b_file.close()
    elif j == 1000:
        c_file = open("netlist_no_inverters2.txt", "w")
        c_file.writelines(list_of_lines_modified_final)
        c_file.writelines('\n')
        list_port_map_lines.append("BV0 a A\nBV1 b B\nBV2 c C\nBV3 d D\n.END")
        c_file.writelines(list_port_map_lines)
        list_port_map_lines = []
        c_file.close()
    elif j == 1500:
        d_file = open("netlist_no_inverters3.txt", "w")
        d_file.writelines(list_of_lines_modified_final)
        d_file.writelines('\n')
        list_port_map_lines.append("BV0 a A\nBV1 b B\nBV2 c C\nBV3 d D\n.END")
        d_file.writelines(list_port_map_lines)
        list_port_map_lines = []
        d_file.close()
    elif j == 2000:
        d_file = open("netlist_no_inverters4.txt", "w")
        d_file.writelines(list_of_lines_modified_final)
        d_file.writelines('\n')
        list_port_map_lines.append("BV0 a A\nBV1 b B\nBV2 c C\nBV3 d D\n.END")
        d_file.writelines(list_port_map_lines)
        list_port_map_lines = []
        d_file.close()
```

```
36      elif j == 2500:
37          d_file = open("netlist_no_inverters5.txt", "w")
38          d_file.writelines(list_of_lines_modified_final)
39          d_file.writelines('\n')
40          list_port_map_lines.append("BV0 a A\nBV1 b B\nBV2 c C\nBV3 d D\n.END"
    )
41          d_file.writelines(list_port_map_lines)
42          list_port_map_lines = []
43          d_file.close()
44      elif j == 3000:
45          d_file = open("netlist_no_inverters6.txt", "w")
46          d_file.writelines(list_of_lines_modified_final)
47          d_file.writelines('\n')
48          list_port_map_lines.append("BV0 a A\nBV1 b B\nBV2 c C\nBV3 d D\n.END"
    )
49          d_file.writelines(list_port_map_lines)
50          list_port_map_lines = []
51          d_file.close()
52      elif j == 3500:
53          d_file = open("netlist_no_inverters7.txt", "w")
54          d_file.writelines(list_of_lines_modified_final)
55          d_file.writelines('\n')
56          list_port_map_lines.append("BV0 a A\nBV1 b B\nBV2 c C\nBV3 d D\n.END"
    )
57          d_file.writelines(list_port_map_lines)
58          list_port_map_lines = []
59          d_file.close()
60
61      words_in_line = list_of_lines_able[subckt_line_number[j]].split(" ")
62      number_of_words = len(words_in_line)
63      if (number_of_words == 9):    # Hay nueve palabras Ej: ".SUBCKT 0200_BDD
    a b c d out vdd gnd"
64          list_port_map_lines.append("B" + str(j + 1) + " a b c d out" + str(j
    + 1) + " vdd gnd " + words_in_line[1] + '\n')
65      elif (number_of_words == 8):  # Hay ocho palabras Ej: ".SUBCKT 0389_BDD a
     b c out vdd gnd"
66          list_port_map_lines.append("B" + str(j + 1) + " " + words_in_line[2]
    + " " + words_in_line[3] + " " + words_in_line[4] + " out" + str(j + 1) + "
     vdd gnd " + words_in_line[1] + '\n')
67      elif (number_of_words == 7):  # Hay siete palabras Ej: ".SUBCKT 0503_BDD
    a b out vdd gnd"
68          list_port_map_lines.append("B" + str(j + 1) + " " + words_in_line[2]
```

```
    + " " + words_in_line[3] + " out" + str(j + 1) + " vdd gnd " +
    words_in_line[1] + '\n')
      else:  # Hay seis palabras Ej: ".SUBCKT 1030_FAC a out vdd gnd"
          list_port_map_lines.append("B" + str(j + 1) + " " + words_in_line[2]
    + " out" + str(j + 1) + " vdd gnd " + words_in_line[1] + '\n')

# Ultimo port map
f_file = open("netlist_no_inverters8.txt", "w")
f_file.writelines(list_of_lines_modified_final)
f_file.writelines('\n')
list_port_map_lines.append("BV0 a A\nBV1 b B\nBV2 c C\nBV3 d D\n.END")
f_file.writelines(list_port_map_lines)
f_file.close()
```

Listing 3.4: Script Python para crear las celdas de los circuitos en un esquemático.

La salida del script Listing 3.5 consiste en una descripción la cual permitirá realizar un *"Port mapping"* de las celdas de los circuitos que fueron creados previamente, logrando de esta manera obtener el conjunto de circuitos en un único esquemático. El script genera 8 archivos los cuales corresponden a 7 conjuntos de 500 circuitos y 1 de 482 circuitos. Un fragmento de la descripción agregada se puede observar en la Figura 3.16 y el esquemático generado se puede observar en la Figura 3.17

```
1   B1 a b c d out1 vdd gnd 0001_BDD
2   B2 a b c d out2 vdd gnd 0002_BDD
3   B3 a b c out3 vdd gnd 0003 BDD
4   B4 a b c d out4 vdd gnd 0004_BDD
5   B5 a b c d out5 vdd gnd 0005_BDD
6   B6 a b c d out6 vdd gnd 0006_BDD
7   B7 a b c d out7 vdd gnd 0007_BDD
8   B8 a b d out8 vdd gnd 0008_BDD
9   B9 a b c d out9 vdd gnd 0009_BDD
10  B10 a b c d out10 vdd gnd 0010_BDD
11  B11 a b out11 vdd gnd 0011_BDD
12  B12 a b c d out12 vdd gnd 0012_BDD
13  B13 a b c d out13 vdd gnd 0013_BDD
14  B14 a b c d out14 vdd gnd 0014_BDD
```

Figura 3.16: Fragmento de la descripción para generar el gran conjunto de circuitos

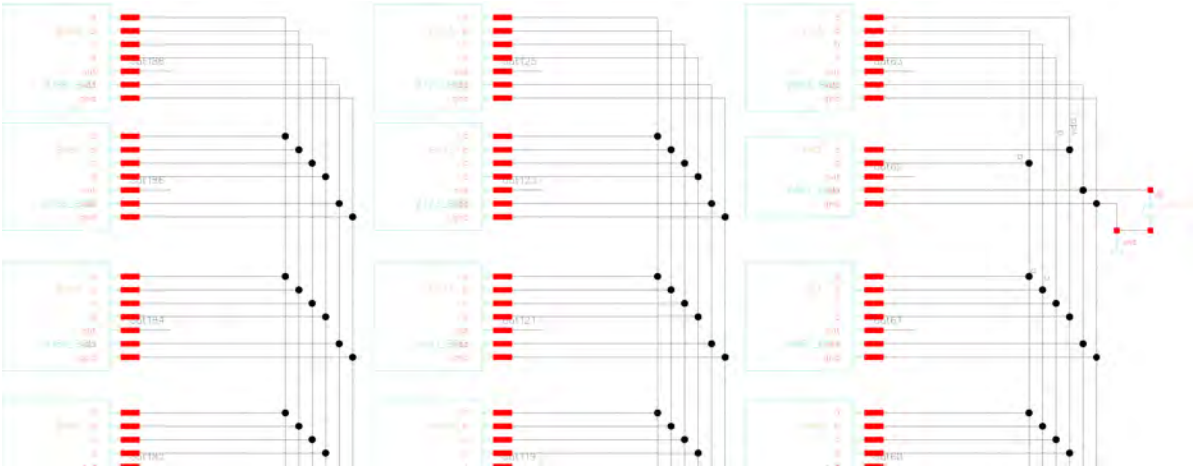


Figura 3.17: Fragmento del esquemático de un conjunto de los primeros 500 circuito para el método BDD.

3.5. Síntesis de los layout de las compuertas lógicas

3.5.1. ASTRAN

Automatic Synthesis of Transistor Networks (ASTRAN) es una herramienta académica que fue desarrollada con el propósito de automatizar la síntesis física de circuitos VLSI. Se destaca por implementar celdas a un nivel de capas (layouts) para procesos de fabricación [19].

Esta herramienta soporta un amplio espectro de arreglo de transistores, lo cual es conveniente para nuestro estudio por el hecho de que se analizan tres métodos de generación de compuertas SCCG. En la Figura 3.18 se puede visualizar la interfaz gráfica de ASTRAN.

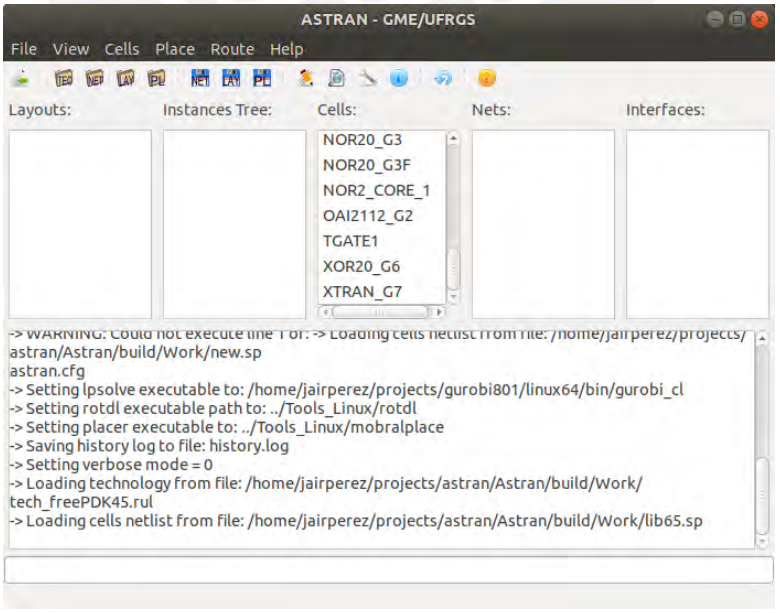


Figura 3.18: GUI de ASTRAN [20].

La Figura 3.19 muestra los elementos de entrada que la herramienta necesita:

- Reglas de diseño.
- Descripción del circuito en formato *Spice* (Netlist).
- Configuración de la plantilla.

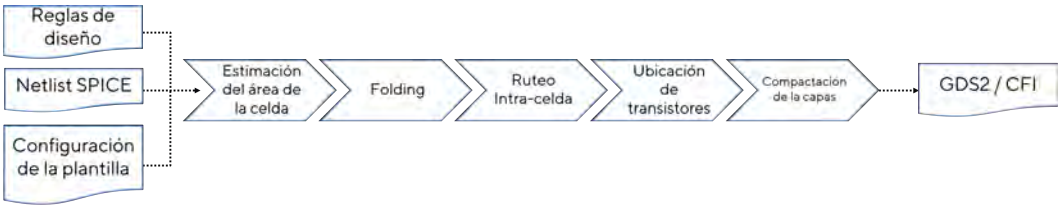


Figura 3.19: Diagrama de flujo de ASTRAN para la generación del layout [20].

3.5.2. Reglas de diseño

La máscara física (layout) de un circuito integrado para ser producido por un proceso de manufactura debe cumplir con ciertas reglas de diseño. Básicamente, las reglas son introducidas con la finalidad de crear circuitos funcionales en áreas pequeñas.

En la Figura 3.20, se pueden identificar los espaciados y dimensiones que deben estar registrados en el archivo *.rul* para que la herramienta ASTRAN pueda utilizarlos como las reglas de diseño. Este archivo se encuentra ubicado dentro la carpeta de instalación de ASTRAN.

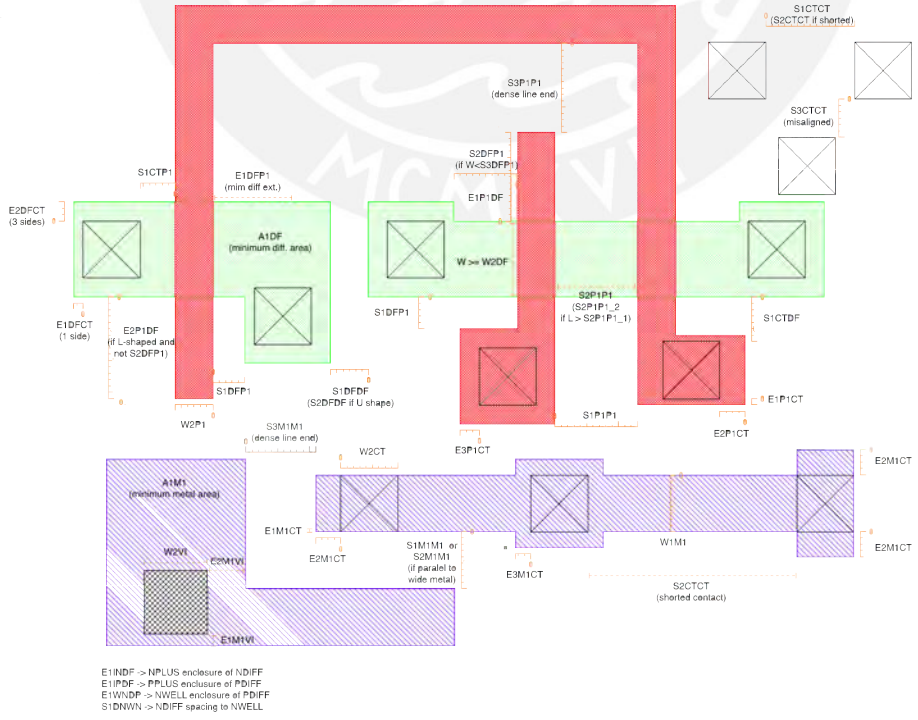


Figura 3.20: Reglas de diseño para la generación de los layouts.

En la Sección 3.4 se destacó que la tecnología utilizada es la TSMC 180 nm; por lo tanto, los layouts también deben son generados considerando esta misma tecnología. En consecuencia, el archivo *.rul* debe editarse según los espaciados y dimensiones de los archivos PDK (Process Design Kit) de la tecnología TSMC 180 nm.

3.5.3. Descripción del circuito

A diferencia del la evaluación eléctrica, las netlist que ingresan a la herramienta ASTRAN contienen todos los inversores de las entradas. Es decir, se utilizan directamente las netlist impresas de la herramienta SwitchCraft.

3.5.4. Configuración de la plantilla

Los parámetros de entrada indicados en la plantilla corresponde directamente a las dimensiones de las capas del layout. Dependiendo de los valores asignados se podrán generar diversas soluciones, ya que la ubicación de los transistores es dada según el área disponible. Debido a que uno de los objetivos del estudio consiste en realizar una comparativa respecto al área que ocupan los layouts, se optó por variar únicamente los valores de la altura de la celda. En la Tabla 3.1 se puede observar los parámetros con los valores asignados y su función.

Tabla 3.1: Parámetros a utilizar en la plantilla de ASTRAN.

Parámetro	Valor asignado	Función
Nombre del diseño	astran_project	Nombre del proyecto
Cuadrícula horizontal	0.5	Dimensión en um del vértice horizontal de la cuadrícula
Cuadrícula vertical	0.5	Dimensión en um del vértice vertical de la cuadrícula
Nombre del la red VDD	VDD	Nombre del nodo VDD utilizado en las netlist
Nombre de la red GND	GND	Nombre del nodo GND utilizado en las netlist
Altura de la fila	13	Parámetro que permite variar la altura de la celda
Tamaño de la fuente	0.8	Parámetro que determina la altura de la fuente
Posición del pozo N	3	Parámetro que permite variar el tamaño del pozo N
Borde del pozo N	0.7	Parámetro que determina el grosor de los bordes del pozo N
Borde de la capa N-Select/P-Select	0.2	Parámetro que determina el grosor de los bordes de las capas N-Select o P-Select

3.6. Análisis de las características físicas de las compuertas lógicas

3.6.1. Visualización de los Layouts

En el diagrama de flujo de la Figura 3.19, se puede percibir que la salida de la herramienta ASTRAN es un archivo en formato GDSII o CFI. En consecuencia, es necesario depender de un visualizador que nos permita observar el layout de los circuitos generados.

El software de CADENCE posee un visualizador llamado *Virtuoso Layout Suite*, esta herramienta nos permite visualizar un archivo en formato *.gds* como se muestra en la Figura 3.21.

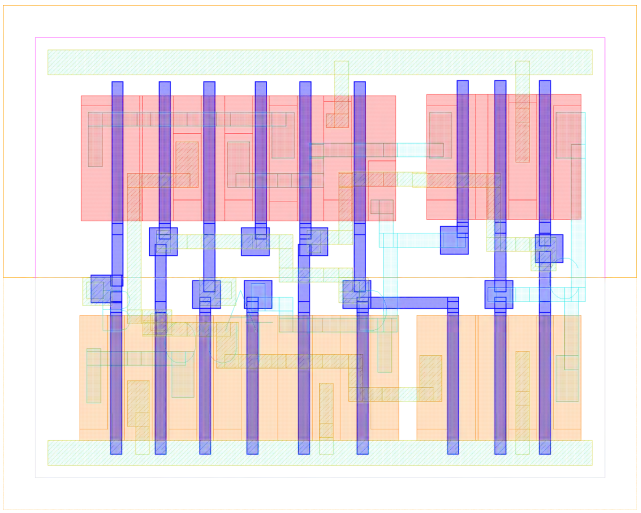


Figura 3.21: Layout de la función *!a*c*d+!a*!c*!d+!b*!c*!d* generada por el método BDD.

3.6.2. Medición de la longitud de los polisilicios y área de los layouts.

La longitud total de polisilicio de cada layout corresponde a la suma total de las longitudes de cada una de las secciones. El software *Virtuoso Layout Suite* posee una regla con la cual se puede determinar el largo y ancho de las secciones de las mascarás. Por ejemplo, en la Figura 3.22 se visualizan distintas secciones de polisilicios que pueden ser dimensionadas por la herramienta regla.

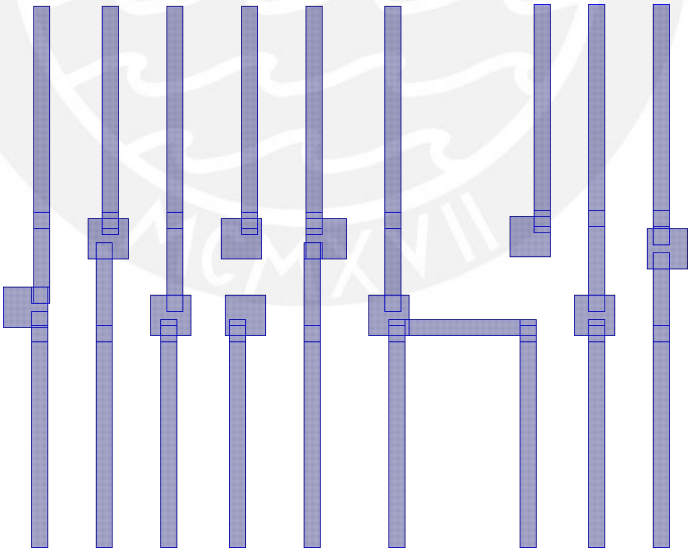


Figura 3.22: Polisilicios del layout de la función *!a*c*d+!a*!c*!d+!b*!c*!d* generada por el método BDD.

De la misma forma, con la herramienta regla se puede medir el largo y ancho de la celda para determinar el área total. No obstante, la actividad de realizar este proceso de medición para cada uno de los layouts requiere de abundante tiempo y está sujeto a errores de medición. Por esta razón,

se ha desarrollado un script en Python llamado LORELAY que automatiza este procedimiento.

3.6.3. LORELAY

LORELAY fue desarrollado con la finalidad de automatizar el proceso de medición de longitud de polisilicios y el área total de un layout. En síntesis, el algoritmo de esta herramienta está basado en el formato ASCII de los layouts, el cual nos indica en coordenadas la ubicación de los vértices de cada sección que conforman el layout. Por lo tanto, utilizando esta información se determinan las dimensiones para calcular el área y largo de cada sección. En la Figura 3.23 se observa un ejemplo de los datos que imprime esta herramienta: Máscaras disponibles, dimensiones cada sección, área y largo de los polisilicios, y el área total del layout.

```
Layers availables:
   LAYER 1
   LAYER 2
   LAYER 3
8 BOX AREA WHERE FOUND

Box Area per Mask:
   1   MASK 1  BOX:0  Width:20 Large:20  Area:400
   2   MASK 1  BOX:0  Width:20 Large:20  Area:400
   3   MASK 1  BOX:0  Width:20 Large:20  Area:400
   4   MASK 2  BOX:0  Width:20 Large:20  Area:400
   5   MASK 2  BOX:0  Width:20 Large:20  Area:400
   6   MASK 2  BOX:0  Width:20 Large:20  Area:400
   7   MASK 3  BOX:0  Width:20 Large:20  Area:400
   8   MASK 3  BOX:0  Width:20 Large:20  Area:400

TOTAL POLY AREA:   44.896
TOTAL POLY LARGE:  2.552
TOTAL LAYOUT AREA: 277.296
```

Figura 3.23: Ejemplo de la salida de LORELAY.

Cabe mencionar que inicialmente los archivos de los layouts están en formato GDSII. Por consiguiente, se ha empleado la herramienta OwlVision, la cual posee un módulo que traduce el formato GDSII al formato ASCII. En la Figura 3.24 se puede observar la interfaz gráfica de este software.

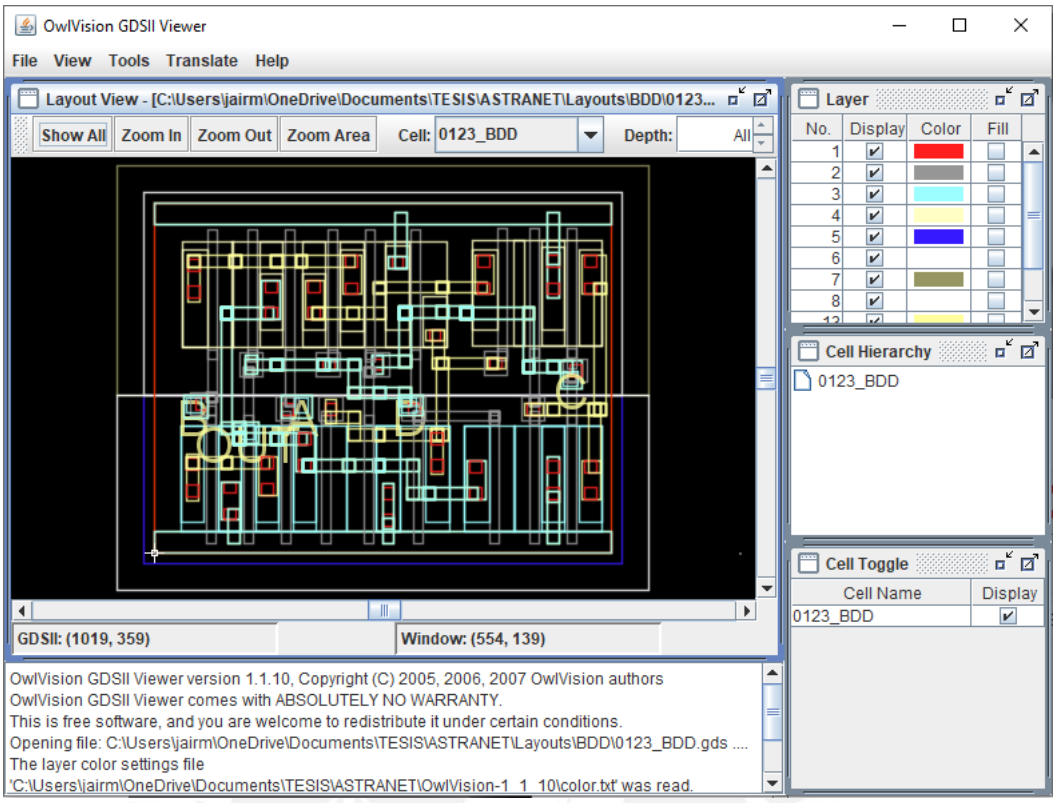


Figura 3.24: GUI OwlVision.

Capítulo 4

Resultados y Simulaciones

El presente capítulo aborda las simulaciones y resultados de la propuesta planteada en el Capítulo 3. La simulaciones mostradas del análisis de potencia de los métodos de generación FAC, FAC-PD y BDD, son realizadas en el software *Virtuoso Analog Design Enviroment* de CADENCE con el uso del simulador Spectre. Las curvas mostradas corresponden a las corrientes que suministra la fuente de voltaje V_{DD}, las cuales se integra para obtener la potencia promedio consumida. En cuanto al análisis físico se definieron las reglas de diseño a partir de la documentación correspondiente a la tecnología TSMC de 180 nm. Las pseudocapas fueron generadas a partir de estas reglas mediante la herramientas ASTRAN, y se determinaron el largo de la secciones de los polisilicios y el área de una muestra de layouts con un script en Python titulado LORELAY. Los resultados obtenidos fueron contrastados con las literaturas mencionadas en el estado del arte.

4.1. Análisis eléctrico

Como se mencionó en el Capítulo 3, se realizó una simulación transitoria para obtener la corriente que entrega la fuente V_{DD}. De esta manera, empleando la Ecuación 3.1 se puede obtener la potencia promedio que disipan los circuitos. Para las simulaciones se seccionaron las 3,982 funciones lógicas en siete conjuntos de 500 funciones y uno restante de 482. La razón de este seccionamiento fue para aligerar la cantidad de procesamientos que realiza la computadora en obtener las simulaciones de cada conjunto.

Es de importancia mencionar que algunas compuertas requieren de la respuesta inversa de las entradas (A, B, C o D). Para ello se emuló el comportamiento inverso con fuentes pulsos en lugar de utilizar un circuito inversor. El motivo principal se debe a que en caso de emplearse un circuito CMOS inversor la respuesta no corresponde exactamente a la inversa de la entradas por la

presencia de retraso en la respuesta, picos de corriente y una diferencia mínima en los niveles de voltaje de alto y bajo respecto a las fuente pulsos de las entradas. Este hecho influye directamente en la respuesta de las compuertas. Asimismo, se destaca que en las mediciones realizadas no se consideraron la potencia consumida por las fuentes pulsos de las entradas A, B, C y D ni las de sus respectivas señales inversas.

Se definió un tiempo de simulación de 160 ns, y un tiempo de subida y bajada de 500 ps para las fuentes pulso de las entradas. En las Figura 4.1-4.24 se observan los resultados de las corrientes proporcionadas por la fuente V_{DD} para cada uno de conjuntos de los tres métodos: FAC, FAC-PD y BDD.

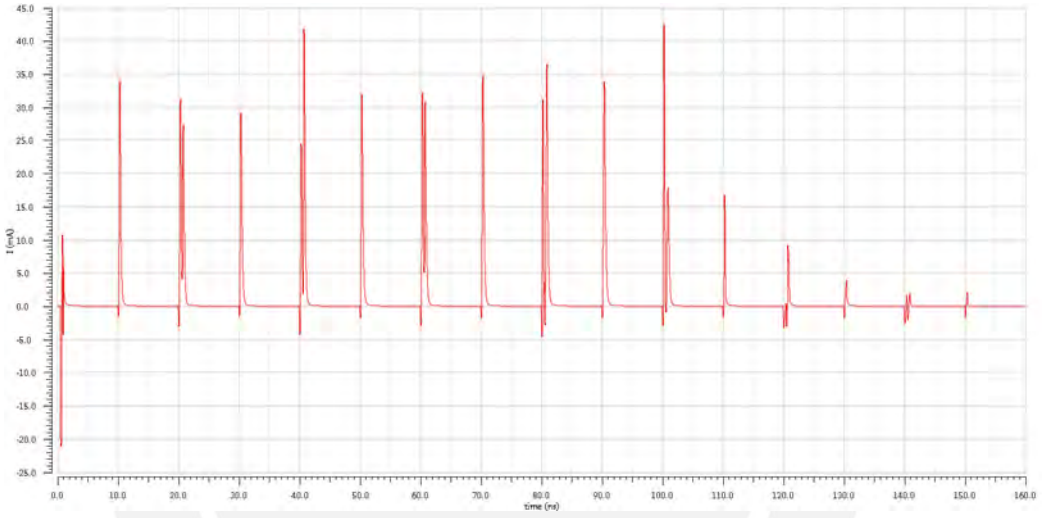


Figura 4.1: Corriente entregada por V_{DD} para el primer conjunto del método FAC.

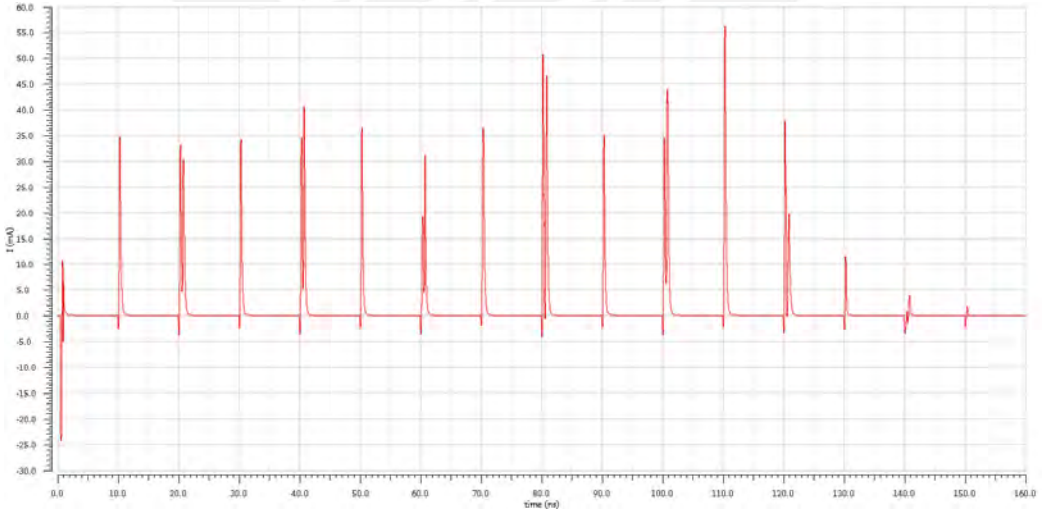


Figura 4.2: Corriente entregada por V_{DD} para el segundo conjunto del método FAC.

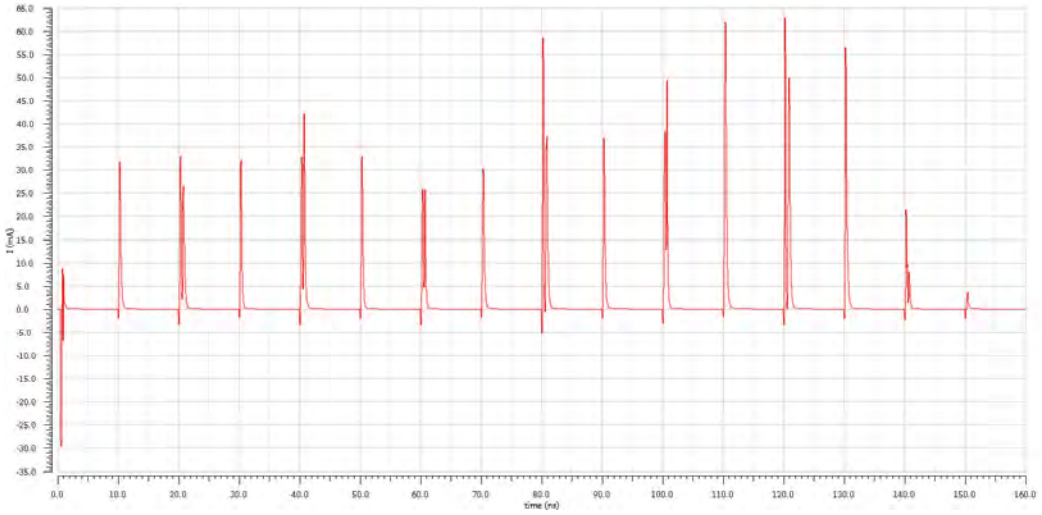


Figura 4.3: Corriente entregada por V_{DD} para el tercer conjunto del método FAC.

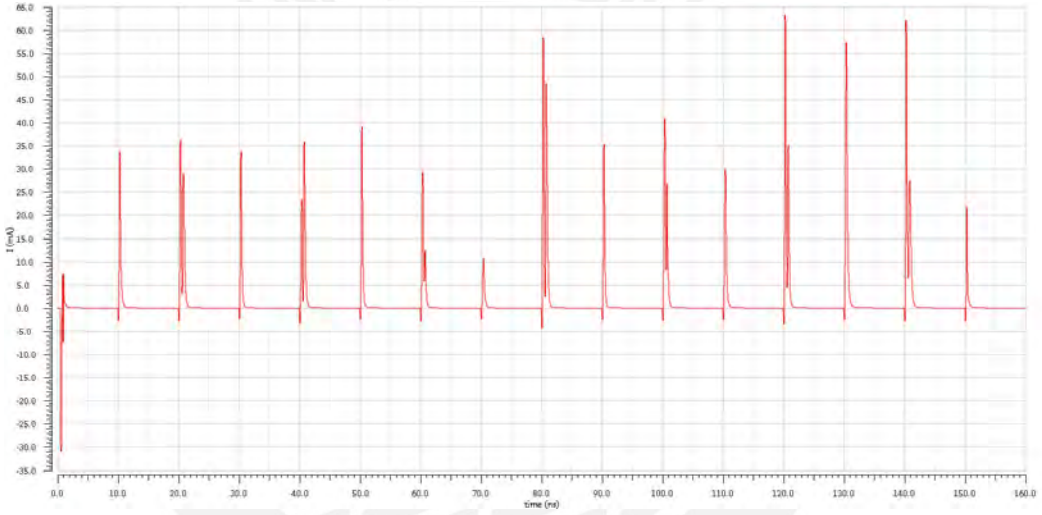


Figura 4.4: Corriente entregada por V_{DD} para el cuarto conjunto del método FAC.

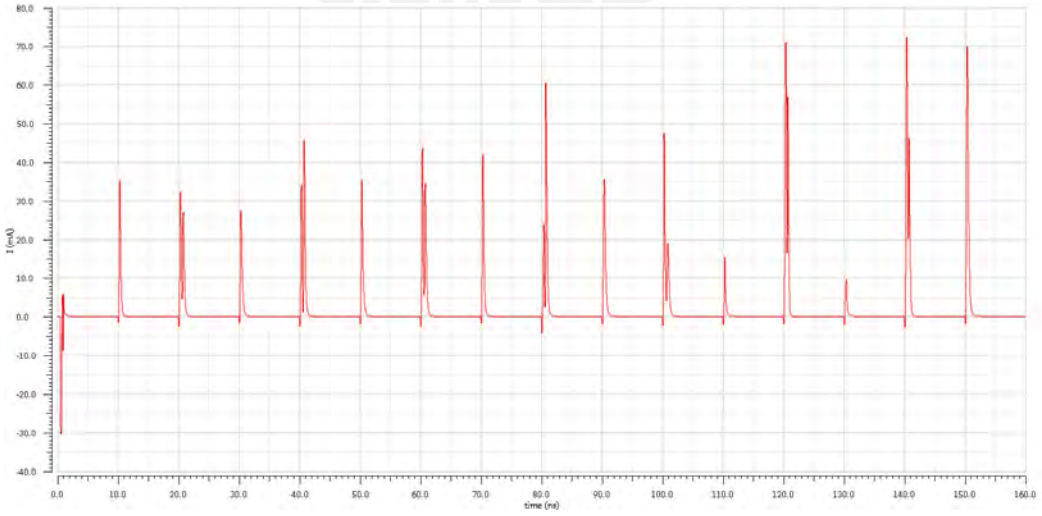


Figura 4.5: Corriente entregada por V_{DD} para el quinto conjunto del método FAC.

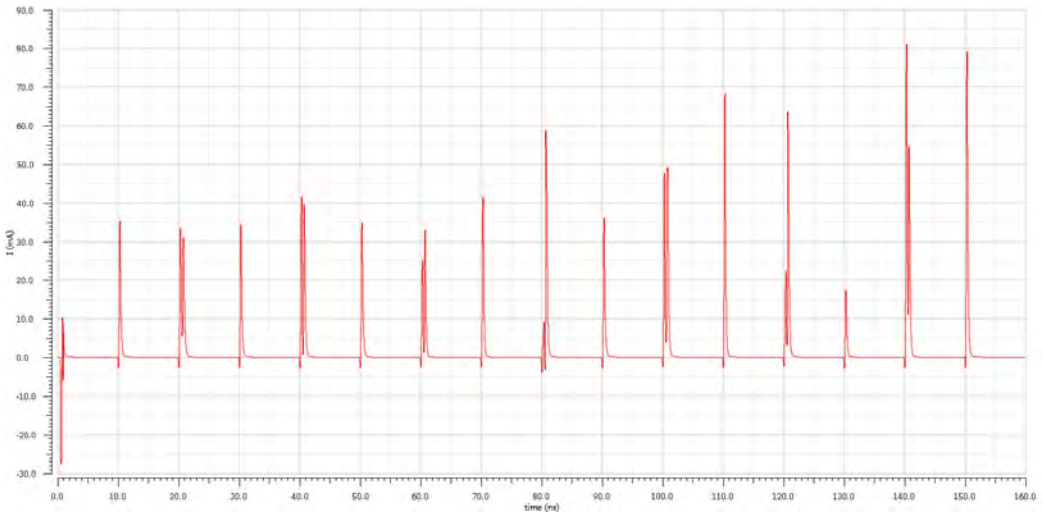


Figura 4.6: Corriente entregada por V_{DD} para el sexto conjunto del método FAC.

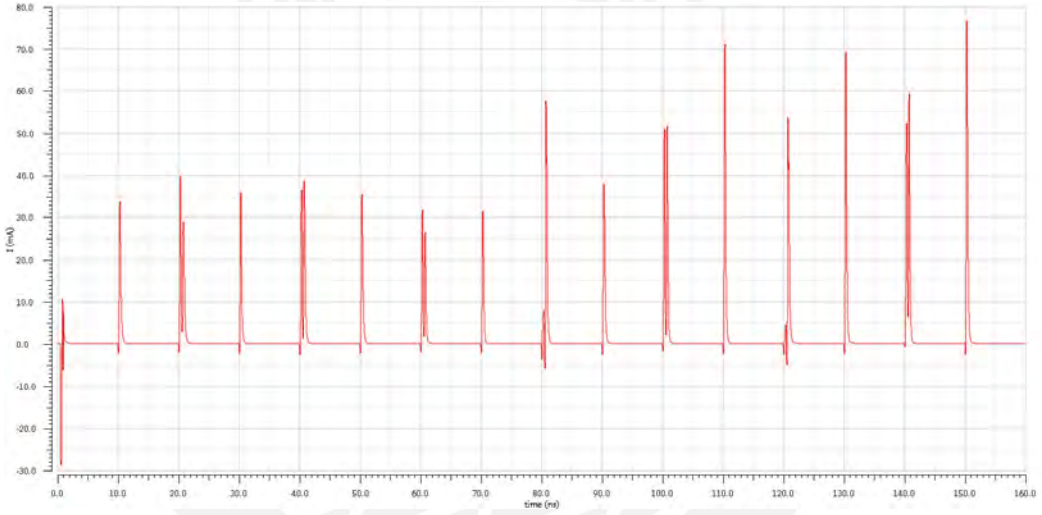


Figura 4.7: Corriente entregada por V_{DD} para el séptimo conjunto del método FAC.

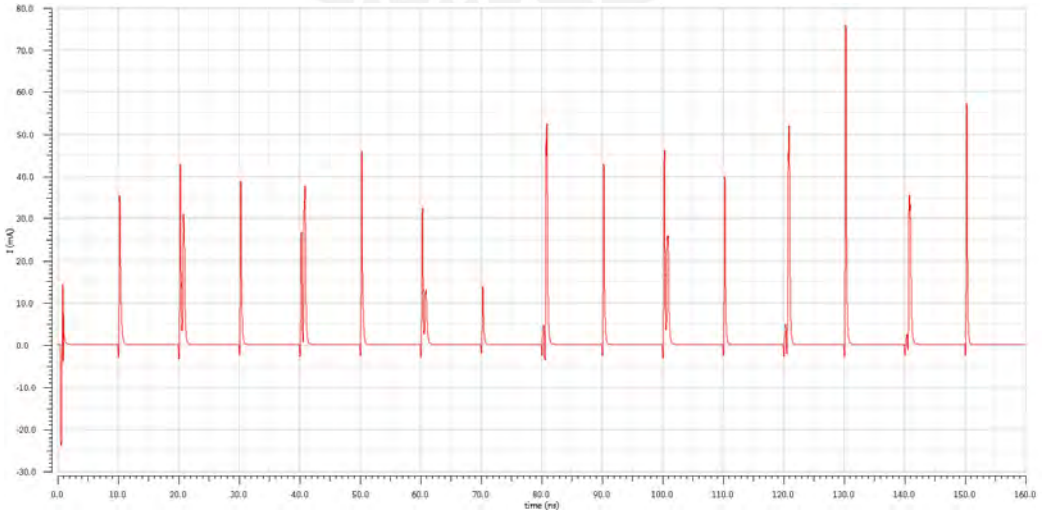


Figura 4.8: Corriente entregada por V_{DD} para el octavo conjunto del método FAC.

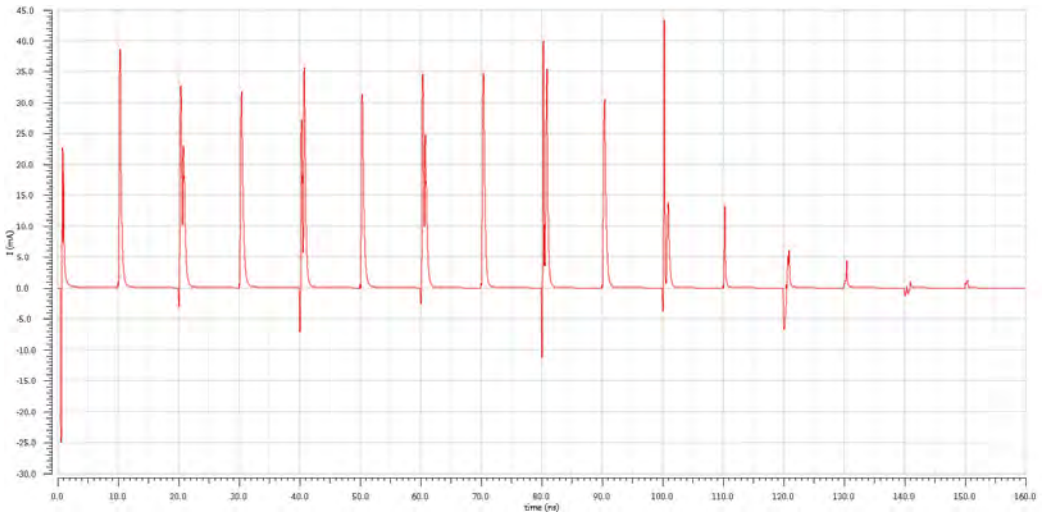


Figura 4.9: Corriente entregada por V_{DD} para el primer conjunto del método FAC-PD.

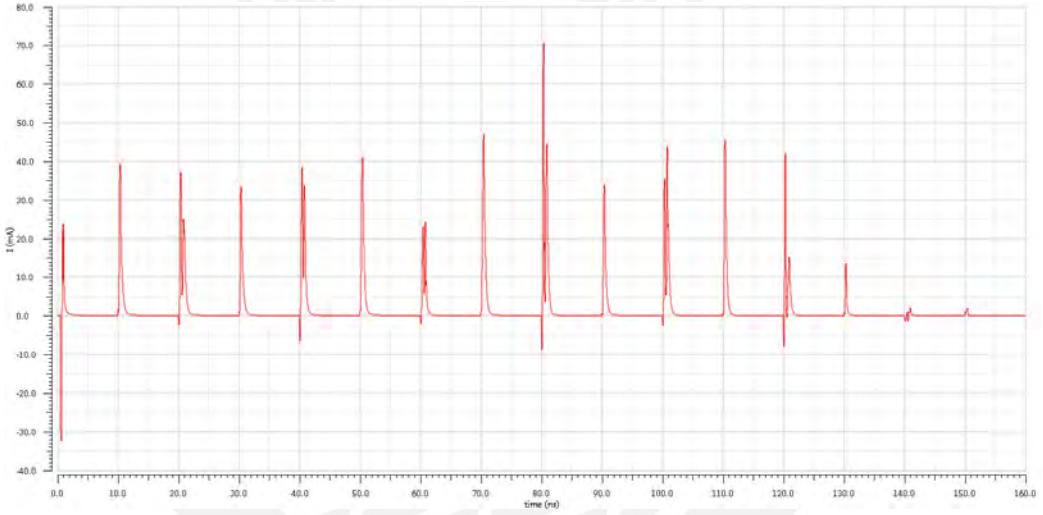


Figura 4.10: Corriente entregada por V_{DD} para el segundo conjunto del método FAC-PD.

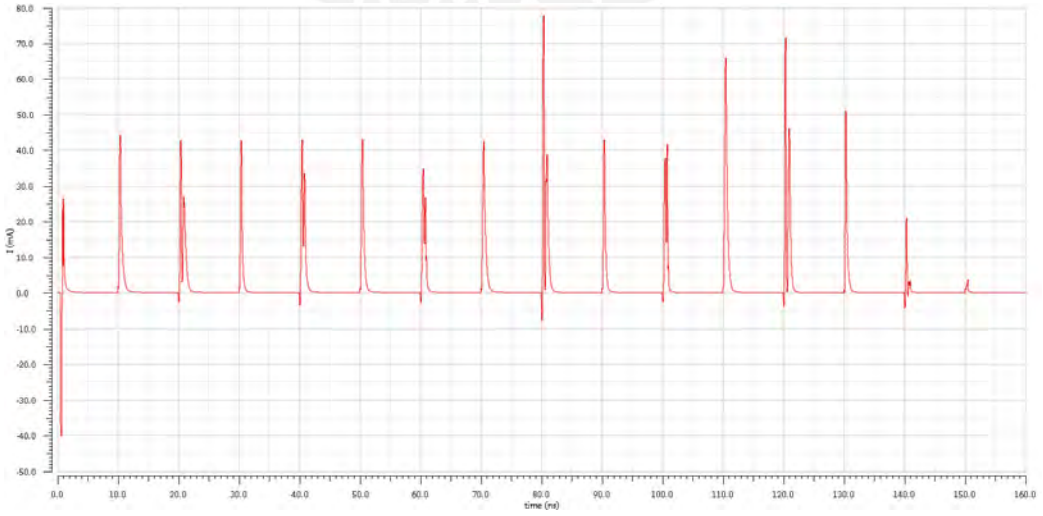


Figura 4.11: Corriente entregada por V_{DD} para el tercer conjunto del método FAC-PD.

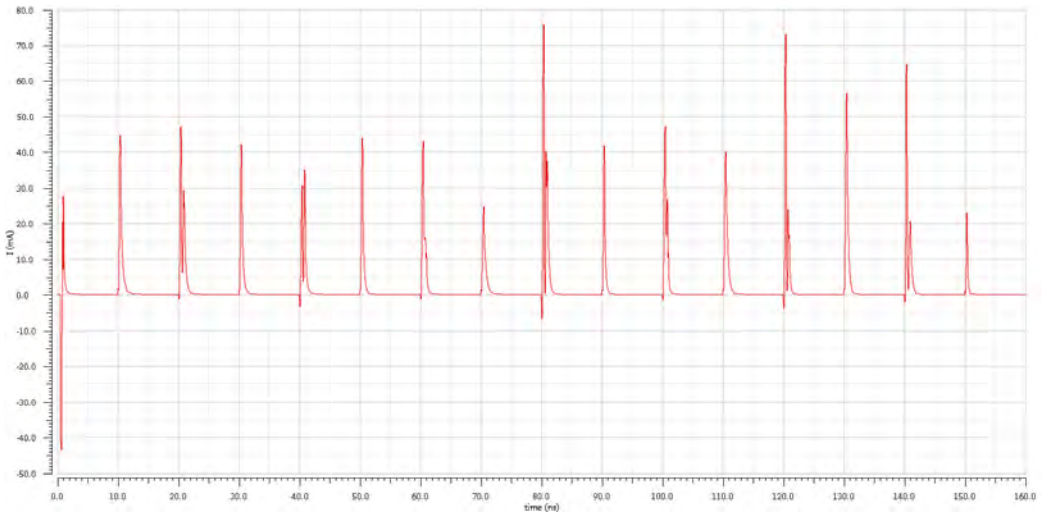


Figura 4.12: Corriente entregada por V_{DD} para el cuarto conjunto del método FAC-PD.

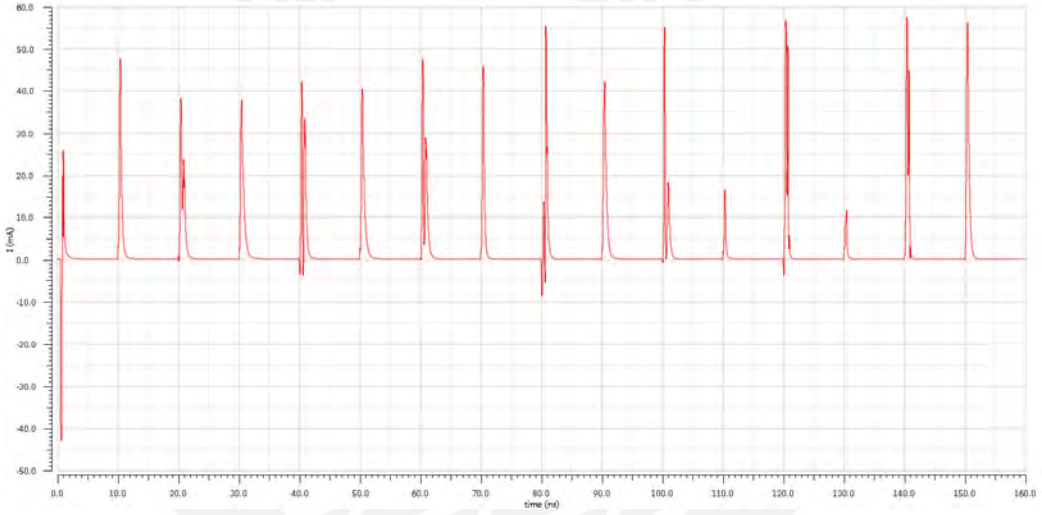


Figura 4.13: Corriente entregada por V_{DD} para el quinto conjunto del método FAC-PD.

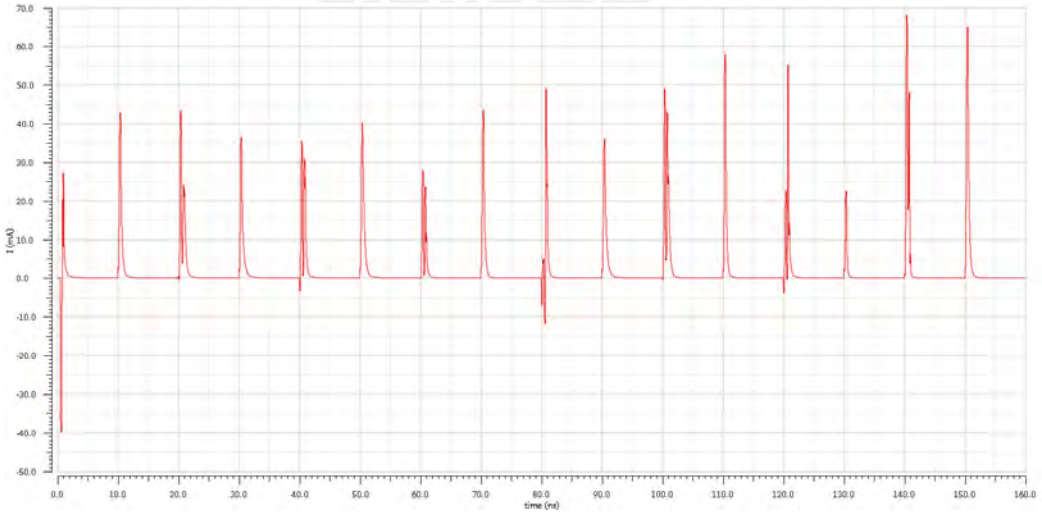


Figura 4.14: Corriente entregada por V_{DD} para el sexto conjunto del método FAC-PD.

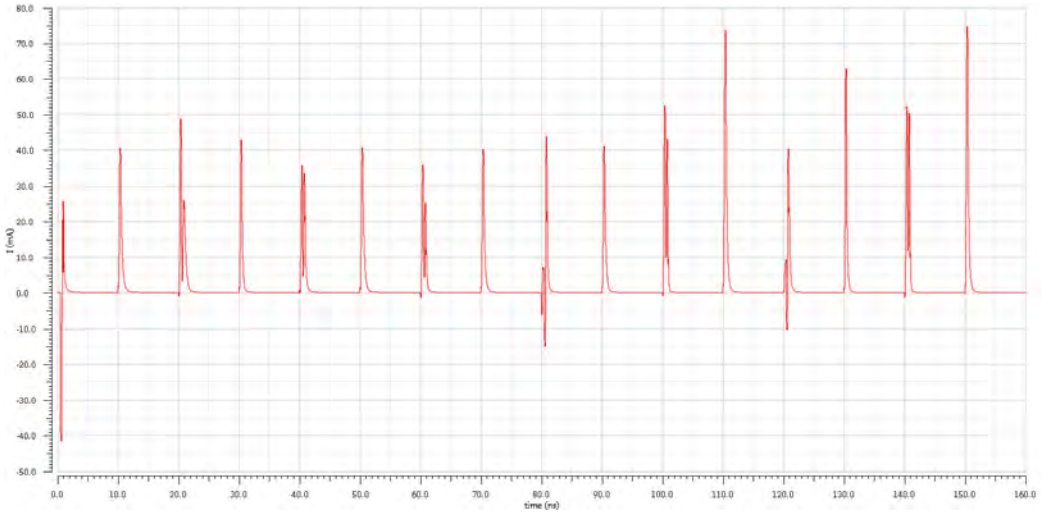


Figura 4.15: Corriente entregada por V_{DD} para el séptimo conjunto del método FAC-PD.

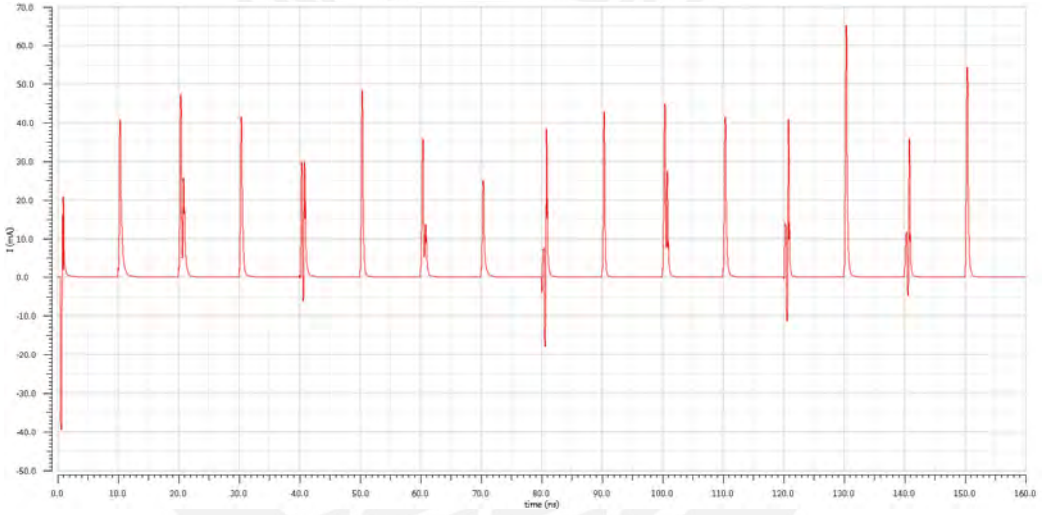


Figura 4.16: Corriente entregada por V_{DD} para el octavo conjunto del método FAC-PD.

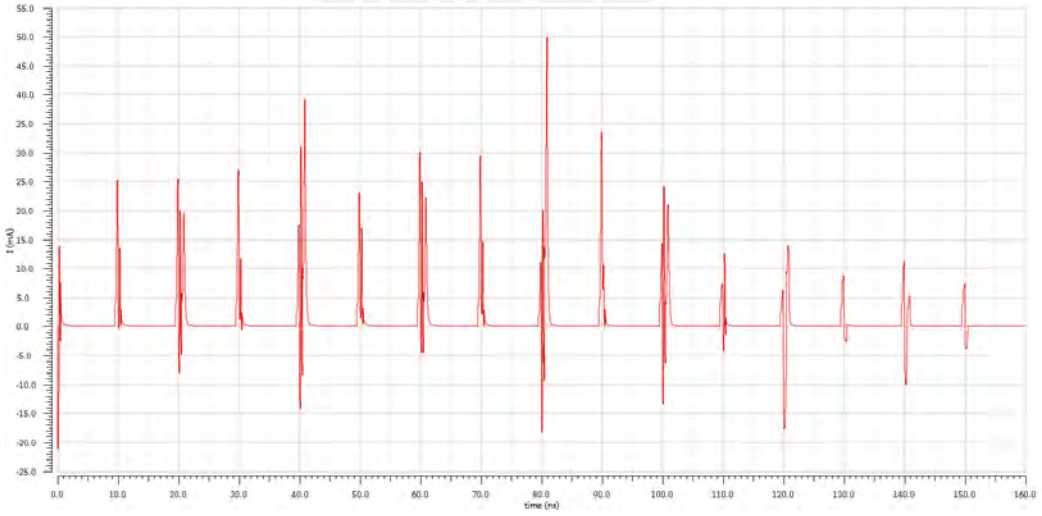


Figura 4.17: Corriente entregada por V_{DD} para el primer conjunto del método BDD.

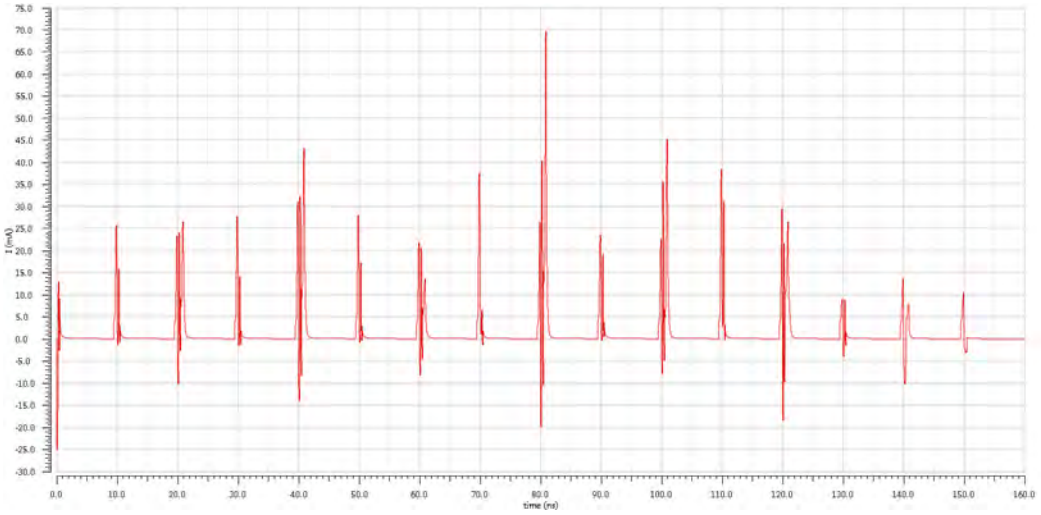


Figura 4.18: Corriente entregada por V_{DD} para el segundo conjunto del método BDD.

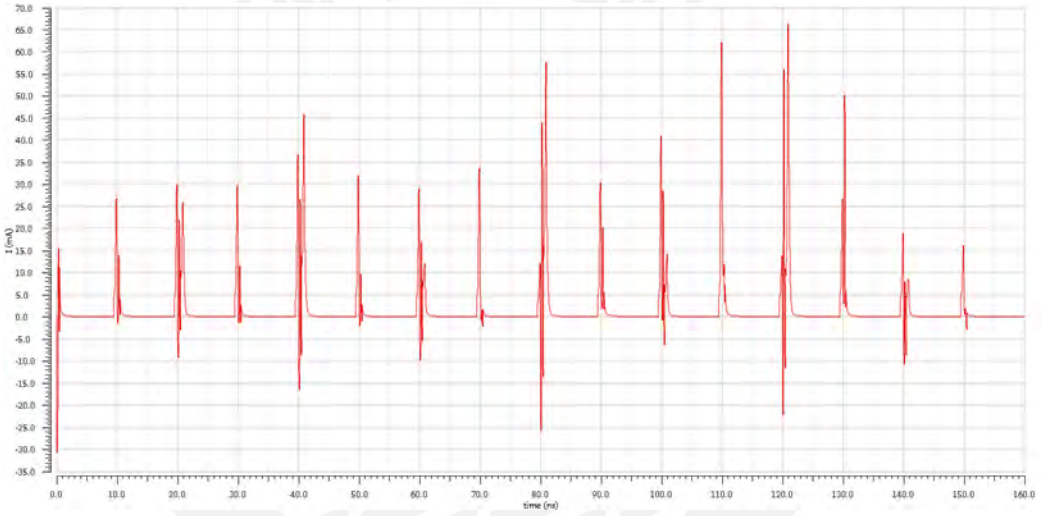


Figura 4.19: Corriente entregada por V_{DD} para el tercer conjunto del método BDD.

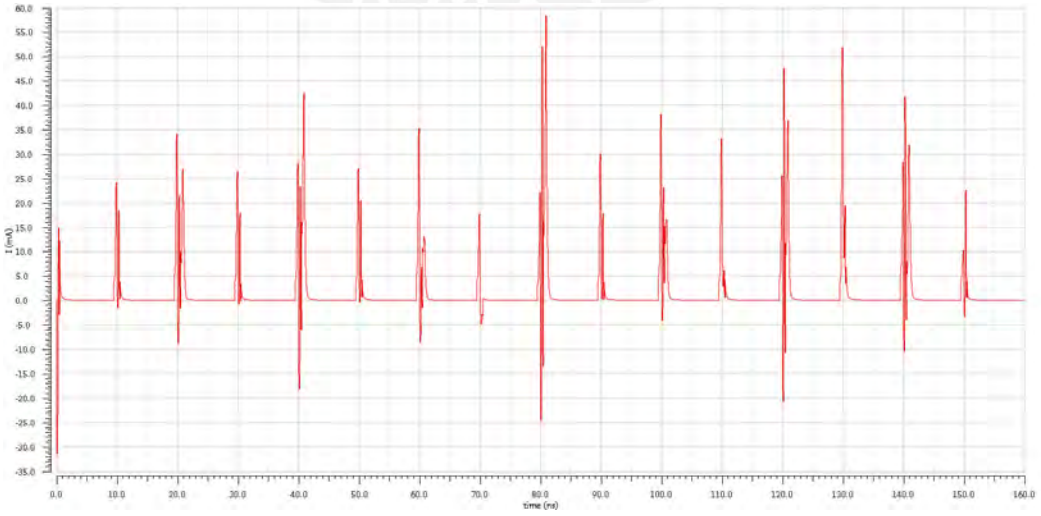


Figura 4.20: Corriente entregada por V_{DD} para el cuarto conjunto del método BDD.

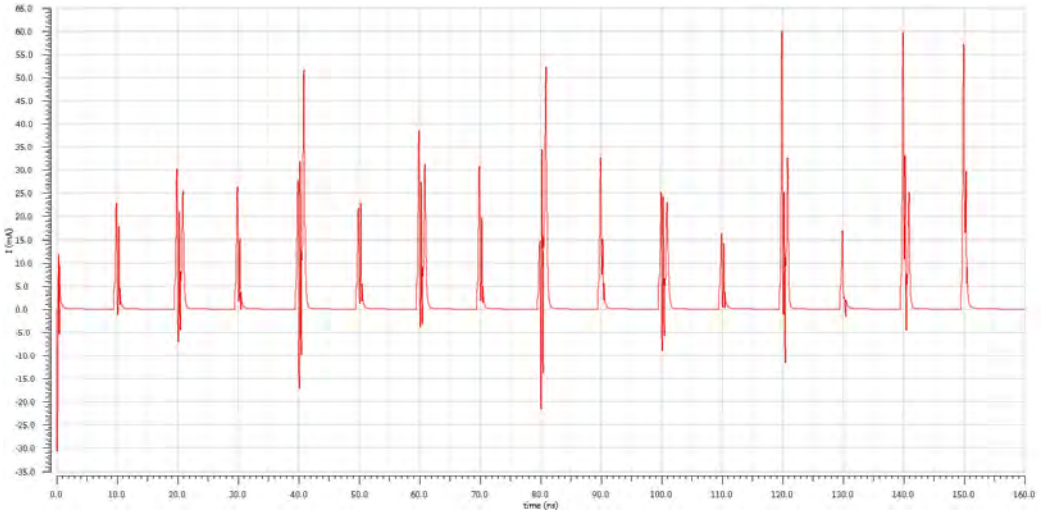


Figura 4.21: Corriente entregada por V_{DD} para el quinto conjunto del método BDD.

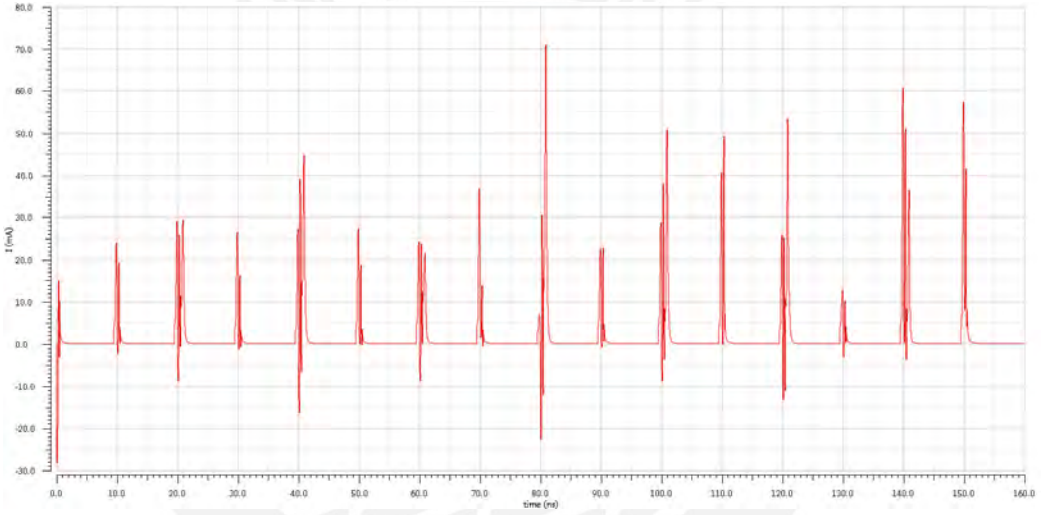


Figura 4.22: Corriente entregada por V_{DD} para el sexto conjunto del método BDD.

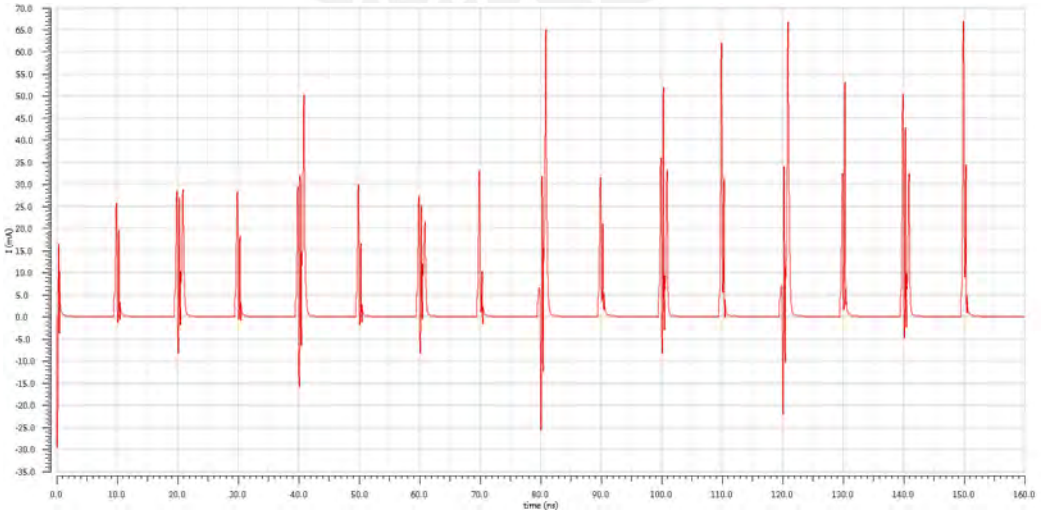


Figura 4.23: Corriente entregada por V_{DD} para el séptimo conjunto del método BDD.

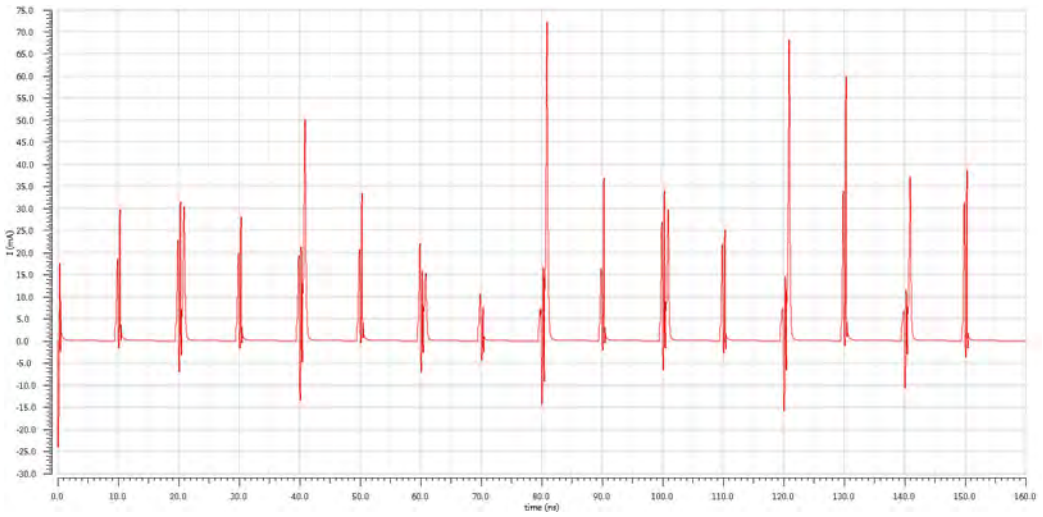


Figura 4.24: Corriente entregada por V_{DD} para el octavo conjunto del método BDD.

Empleando la Ecuación 3.1 con cada una de las corrientes obtenidas de los conjuntos se formó la Tabla 4.1 y se determinó la potencia promedio total consumida por cada método.

Tabla 4.1: Resultados de la medición de la potencia promedio de los métodos FAC, FAC-PD y BDD.

Método	**Potencia promedio (mW)**								
	(1-500)	**(501-1000)**	**(1001-1500)**	**(1501-2000)**	**(2001-2500)**	**(2501-3000)**	**(3001-3500)**	**(3501-3982)**	**Total**
FAC	1.548	2.014	2.367	2.323	2.728	2.685	2.602	2.102	18.369
FAC-PD	1.965	2.638	3.339	3.379	3.329	3.316	3.420	2.615	24.001
BDD	1.716	2.348	2.808	2.829	3.190	3.296	3.419	2.550	22.157

Los resultados muestran que las compuertas SCCG generadas por el método FAC-PD son las que más potencia disipan. Con respecto a los resultados de los métodos FAC y BDD, el que registra una menor disipación es el FAC con 18.369 mW.

Al observar los resultados obtenidos en la literatura [4], se identifico que los investigadores en sus mediciones consideraron los circuitos inversores de las entradas de cada compuerta, y buffers para simular entradas y salidas mas realistas. No obstante, es de importancia realizar una comparativa, para la cual se normalizaron los resultados respecto al método FAC. Los resultados comparativos se observan en la Tabla 4.2

Tabla 4.2: Comparativa de resultados normalizados al método FAC.

Método	**Potencia promedio**	
	Resultados de [4] con inversores	**Resultados del presente trabajo**
FAC	1.000	1.000
FAC-PD	1.248	1.307
BDD	1.060	1.206

En la Figura 4.25, se insta una comparativa de los métodos de generación utilizados en el

presente estudio. En vista de que el método FAC-PD es el que posee peor rendimiento en términos de potencia, se consideran los porcentajes de optimización respecto al ya mencionado método FAC-PD.

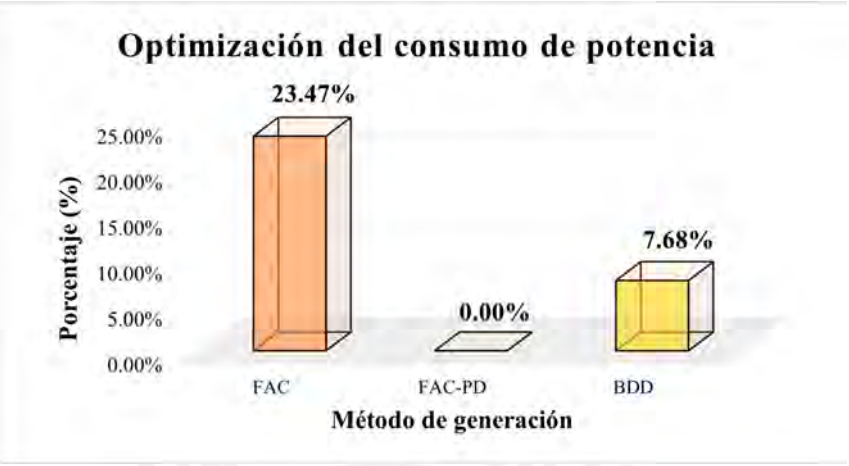


Figura 4.25: Geometría comparativa entre los métodos FAC, FAC-PD y BDD para el consumo de potencia promedio.

4.2. Análisis físico

A diferencia del análisis eléctrico en esta parte del estudio se consideraron todos los inversores CMOS de las entradas para la generación de las pseudocapas. Asimismo, se definieron las reglas de diseño a partir de la documentación confidencial de la empresa TSMC para la tecnología de 180 nm. En dicho documento se identificaron las dimensiones y/o espaciados que definen las mencionadas reglas de diseño. Por motivos anteriormente aludidos, no se pueden detallar en este trabajo el archivo *.rul* utilizado para la generación de las capas físicas con la herramienta ASTRAN.

Por lo tanto, como primer análisis se procede a determinar la cantidad de transistores que emplea cada método. Como se puede visualizar en la Tabla 4.3, el método que utiliza una menor cantidad de transistores para conformar una compuerta SCCG es el FAC-PD.

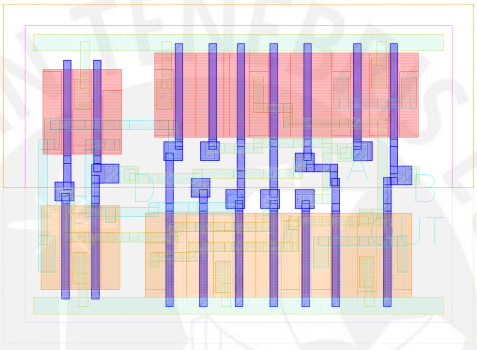
Tabla 4.3: Número de transistores utilizados por cada método.

Método	**Número de transistores**			
	Por compuerta lógica	**Plano pull-up**	**Plano pull-down**	**Total Tr.**
FAC	25.636	12.816	12.820	102082
FAC-PD	31.122	15.561	15.561	123928
BDD	25.739	12.858	12.880	102491

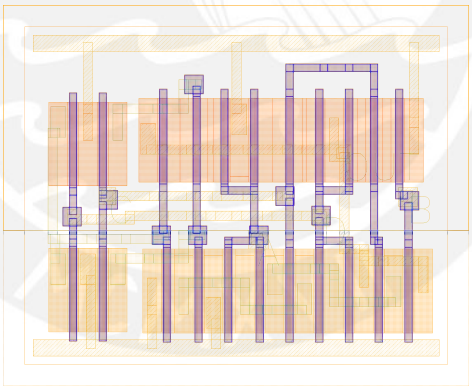
En cuanto a las capas físicas generadas, el criterio utilizado para definir la altura de las mismas consistió en partir de una altura de 13. En caso de que no se encontrara una solución para

generar la respectiva celda, se optaba por aumentar la altura en una unidad. Estos valores de altura establecidos dependen estrictamente de que los valores de la cuadrícula horizontal y vertical tengan un valor de 0.5, ya que cada unidad de la altura corresponde a una cuadricula. Cabe mencionar que a pesar de que la herramienta ASTRAN permite un proceso de *folding* para obtener un menor área este no se está tomando en cuenta, ya que se pretende analizar el impacto de cada método sin otros procesos de optimización de área.

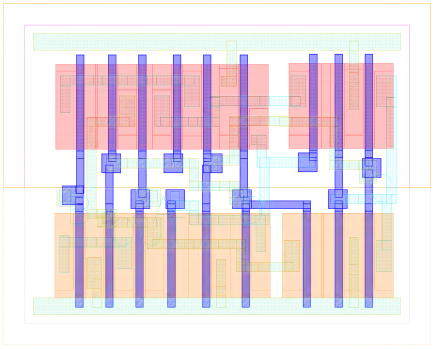
En la Figura 4.26 se visualizan las capas obtenidas para la función *!a*c*d+!a*!c*!d+!b*!c*!d*. En la figura (b) se aprecia que la capa generada por el método FAC-PD es la que ocupa un mayor área con respecto a las demás. En la Tabla 4.4, se visualiza que esta tendencia se mantiene para los 433 layouts analizados.



(a) FAC



(b) FAC-PD



(c) BDD

Figura 4.26: Capas físicas de la función *!a*c*d+!a*!c*!d+!b*!c*!d* obtenidas por ASTRAN.

Tabla 4.4: Resultados de la medición del área promedio de los layouts generados por los métodos FAC, FAC-PD y BDD.

Método	**Área**	
	Layouts evaluados	**Área promedio (μm^2)**
FAC	433	60.208
FAC-PD	433	67.536
BDD	433	60.779

En torno a la longitud de los polisilicios se seleccionó la misma muestra de 433 layouts. Los resultados de la Tabla 4.5 muestran que el método que utiliza menos cantidad de material de polisilicio en los layouts generados por ASTRAN es el método BDD.

Tabla 4.5: Resultados de la medición de la longitud de los polisilicios de los layouts.

Método	**Longitud de los polisilicios**	
	Layouts evaluados	**Longitud promedio (μm)**
FAC	433	65.523
FAC-PD	433	79.148
BDD	433	65.437

En la literatura [1], se empleó una metodología que los autores denominan Kernel Finder. Esta consiste en la generación de una compuerta SCCG a través de un método basado en gráficos. Por ello, al revisar la literatura se pudo observar que los métodos BDD y Kernel Finder son los mismos. En consecuencia, con el objetivo de realizar una comparativa, nuestros resultados se normalizarán respecto al método BDD y los resultados de [1] se normalizarán respecto al método Kernel Finder.

En las Tablas 4.6-4.11 se visualizan los resultados comparativos normalizados respecto a los métodos gráficos. En primer lugar, en cuanto a la cantidad de transistores, se puede observar que los resultados del presente trabajo demuestran que el método FAC-PD emplea una menor cantidad de semiconductores; no obstante, la mejora en relación al método BDD es de 0.004. En segundo lugar, en relación al tamaño de área de los layouts, nuestros resultados corroboran que el utilizar un método gráfico para generar compuertas SCCG acarrea a una optimización del área, pero otras metodologías como el FAC y el Algoritmo usado en [1] permiten obtener un mejor ahorro en el área. En tercer lugar, la metodologías BDD es la que emplea menor cantidad de polisilicio para implementar el layout de una compuerta lógica.

Tabla 4.6: Comparación respecto a cantidad de transistores en [1] normalizados a Kernel Finder.

Método	**Número de transistores**	
	Por compuerta lógica	**Total Tr.**
Kernel Finder	1.000	1.000
Algoritmo 1	1.064	1.065

Tabla 4.7: Comparación respecto a cantidad de transistores para los métodos FAC, FAC-PD y BDD normalizados respecto a BDD.

Método	**Número de transistores**	
	Por compuerta lógica	**Total Tr.**
FAC	0.996	
FAC-PD	1.209	
BDD	1.000	

Tabla 4.8: Comparación respecto al área de los métodos propuesto en [1] normalizados a Kernel Finder.

Método	**Área**
	Área promedio
Kernel Finder	1.000
Algoritmo 1	0.841

Tabla 4.9: Resultados de la medición del área promedio normalizados respecto a BDD.

Método	**Área**	
	Layouts evaluados	**Área promedio**
FAC	433	0.991
FAC-PD	433	1.111
BDD	433	1.000

Tabla 4.10: Resultados de la medición de la longitud promedio de los polisilicios normalizados respecto a BDD.

Método	**Longitud de los polisilicios**
	Longitud promedio
Kernel Finder	1.000
Algoritmo 1	0.968

Tabla 4.11: Resultados de la medición de la longitud promedio de los polisilicios normalizados respecto a BDD.

Método	**Longitud de los polisilicios**	
	Layouts evaluados	**Longitud promedio**
FAC	433	1.001
FAC-PD	433	1.210
BDD	433	1.000

Por último, las Figuras 4.27-4.29 muestran una comparativa geométrica de las optimizaciones en: uso de transistores, área de los layouts y longitud de los polisilicios. Debido a que el método FAC-PD es el que posee peor rendimiento en todos los aspectos estudiados, se consideraron los porcentajes de optimización respecto a este.

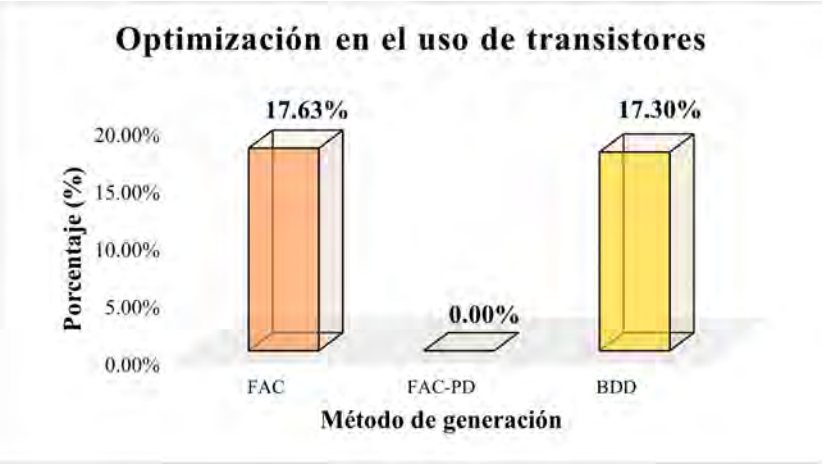


Figura 4.27: Geometría comparativa entre los métodos FAC, FAC-PD y BDD para el uso de transistores.

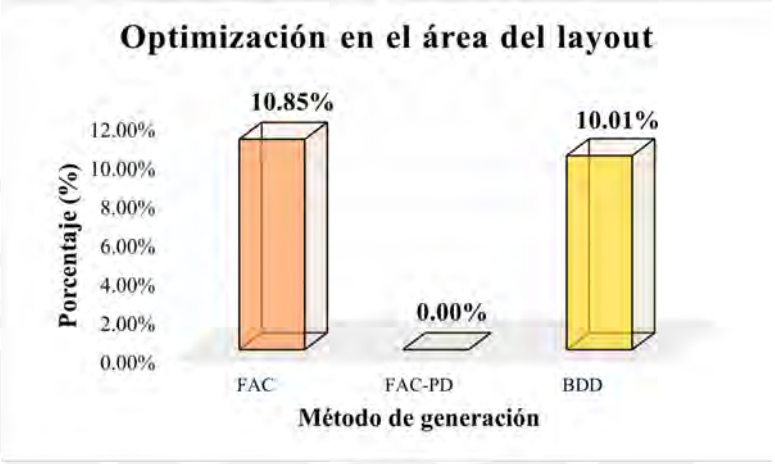


Figura 4.28: Geometría comparativa entre los métodos FAC, FAC-PD y BDD para el área de los layouts.

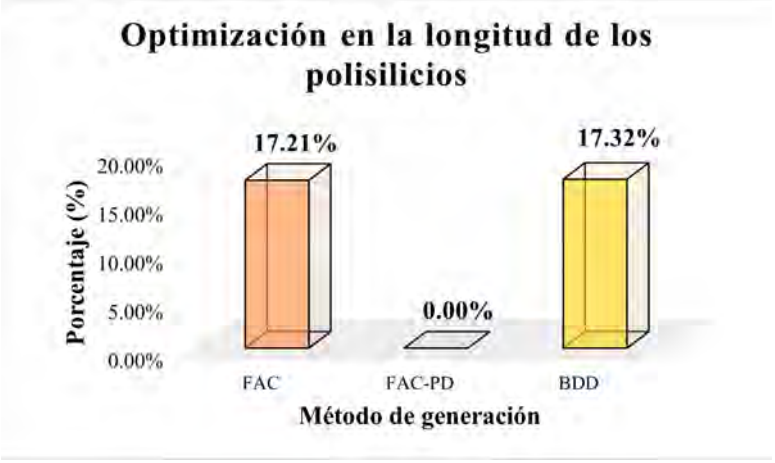


Figura 4.29: Geometría comparativa entre los métodos FAC, FAC-PD y BDD para la longitud de los polisilicios.

Conclusiones

1. En relación a las características eléctricas, el método de generación FAC propuesto en el framework SwitchCraft es el que mejor propiedades posee en términos de potencia con un 23.47 % de mejora respecto al FAC-PD.

2. Referente a las características físicas, se logró identificar porcentajes significativo de mejora en el uso de transistores de 17.63 % y 17.30 % para los métodos FAC y BDD, tomando como base al método FAC-PD.

3. Asimismo, la misma tendencia se mantiene al estudiar la métrica del área activa de las celdas de 433 layouts generados por ASTRAN. Considerando el método FAC-PD como referencia, se identificó un porcentaje de mejora de 10.85 % y 10.01 % para los métodos FAC y BDD, correspondientemente.

4. En cuanto al porcentaje de optimización en la longitud de los polisilicios, se determinó que respecto al método FAC-PD las metodologías FAC y BDD poseen 17.21 % y 17.32 % de mejora, respectivamente.

5. Es relevante mencionar que para los porcentajes de mejora en el área y longitud de los polisilicios es necesario ampliar la cantidad de las muestras a medir. Esto, con finalidad de realizar una comparativa más coherente con la cantidad de muestras del análisis eléctrico.

6. El análisis demuestra que el método de generación FAC-PD es el que presenta las peores métricas de optimización. Por lo que para un enfoque de diseño on-the-fly para circuitos VLSI no se recomienda optar por esta metodología.

7. En función de los resultados expuestos, se puede definir que el método FAC-PD es el más adecuado si se considera como criterios los costos de diseño y la probabilidad de error, ya que es la que ocupa una menor cantidad de transistores y utiliza una menor área activa en sus celdas físicas.

8. En cuanto a la longitud de los polisilicios, la metodología BDD es el que posee mejor porcentaje de optimización; sin embargo, la metodología FAC-PD difiere en 0.11 % del método BDD. Por lo tanto, ambos son métodos son óptimos en términos del uso de polisilicio.

9. Los resultados obtenidos en el presente estudio mantienen la tendencia con respecto a los porcentajes del estudio [4]. La diferencia recae en el no uso de los inversores CMOS y el software de simulación. No obstante, se recomienda optar por el uso de software de simulación potentes como CADENCE, el cual permite obtener resultados más realistas.

10. A partir de lo puntos expuestos anteriormente se puede afirmar que el primer objetivo planteado fue alcanzado, debido a que se determinó la metodología de generación de compuertas SCCG más óptima en términos de costos de diseño y probabilidad de error. De la misma forma, los objetivos que plantean realizar el análisis eléctrico y físico fueron alcanzados. Los mismos fueron base fundamental para obtener resultados y lograr el primer objetivo.

Recomendaciones

1. La medición de la longitud de los polisilicios requiere de tiempo y precisión, es por ello que en el estudio se desarrolló LORELAY. Por lo tanto, como trabajo a futuro se plantea optimizar el código para ser programado en Java y desarrollar una GUI que permita su uso en el ámbito académico.

2. La generación de los layouts requiere de abundante tiempo (7-15 minutos según el tiempo configurado). A pesar de utilizar un script para automatizar este procesamiento se recomienda ejecutar ASTRAN en una estación de trabajo que posee el hardware adecuado para simulaciones pesadas.

3. Como trabajo a futuro se puede evaluar nuevamente las métricas estudiadas, pero comparando diferentes tecnologías de transistores.

Bibliografía

[1] M. S. Cardoso, G. H. Smaniotto, A. A. O. Bubolz, L. S. da Rosa Junior, and F. S. de Marques, "Area-Aware Design of Static CMOS Complex Gates," in *2018 16th IEEE International New Circuits and Systems Conference (NEWCAS)*. Montreal, QC: IEEE, Jun. 2018, pp. 282–286, (Última fecha de consulta: 29 septiembre 2021). [Online]. Available: https://ieeexplore.ieee.org/document/8585570/

[2] "NVIDIA AMPERE GA102 GPU ARCHITECTURE," 2021, (Última fecha de consulta: 26 noviembre 2021). [Online]. Available: https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf

[3] V. N. Possani, V. Callegaro, A. I. Reis, R. P. Ribas, F. de Souza Marques, and L. S. da Rosa, "Graph-Based Transistor Network Generation Method for Supergate Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 2, pp. 692–705, Feb. 2016, (Última fecha de consulta: 14 octubre 2021). [Online]. Available: http://ieeexplore.ieee.org/document/7064758/

[4] H. Kessler, M. Muñoz, P. Finkenauer, L. Da Rosa Jr., and V. Camargo, "Electrical evaluation of logic network generation methods for on-the-fly supergate design," vol. 16, no. 3, pp. 1–7. [Online]. Available: https://jics.org.br/ojs/index.php/JICS/article/view/527

[5] T. Sasao, *Switching Theory for Logic Synthesis*. Boston, MA: Springer US, 1999, (Última fecha de consulta: 23 noviembre 2021). [Online]. Available: http://link.springer.com/10.1007/978-1-4615-5139-3

[6] V. Callegaro, F. d. S. Marques, C. E. Klock, L. S. da Rosa, R. P. Ribas, and A. I. Reis, "SwitchCraft: A Framework for Transistor Network Design," in *Proceedings of the 23rd symposium on Integrated circuits and system design - SBCCI '10*. São Paulo, Brazil: ACM Press, 2010, p. 49, (Última fecha de consulta: 28 septiembre 2021). [Online]. Available: http://portal.acm.org/citation.cfm?doid=1854153.1854167

[7] G. Ammes, J. J. Baqueta, A. I. Reis, and R. P. Ribas, “Improvements of the SwitchCraft framework.”

[8] S. D. Brown and Z. G. Vranesic, *Fundamentals of digital logic with VHDL design*, 2nd ed., ser. McGraw-Hill series in electrical and computer engineering. Dubuque, IA: McGraw-Hill Companies, 2005.

[9] L. S. da Rosa Junior, “Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles,” Ph.D. dissertation, Universidade Federal Do Rio Grande Do Sul, Porto Alegre, Jul. 2008, (Última fecha de consulta: 23 noviembre 2021). [Online]. Available: https://www.inf.ufrgs.br/logics/docman/work_phd_leomar.pdf

[10] P. A. Mason, “ECE 410: VLSI Design Course Lecture Notes,” East Lansing, MI, USA, (Última fecha de consulta: 23 noviembre 2021). [Online]. Available: https://www.egr.msu.edu/classes/ece410/mason/files/Ch2.pdf

[11] V. Callegaro, “SwitchCraft: Um ambiente computacional para síntese e análise de redes lógicas,” Bachelor’s thesis, Universidade Federal Do Rio Grande Do Sul, Porto Alegre, Dec. 2009, (Última fecha de consulta: 23 noviembre 2021). [Online]. Available: https://www.lume.ufrgs.br/handle/10183/18570

[12] L. S. da Rosa Junior, F. S. Marques, T. M. G. Cardoso, R. P. Ribas, S. S. Sapatnekar, and A. I. Reis, “Fast disjoint transistor networks from BDDs,” in *Proceedings of the 19th annual symposium on Integrated circuits and systems design - SBCCI ’06*. Ouro Preto, MG, Brazil: ACM Press, 2006, p. 137, (Última fecha de consulta: 16 noviembre 2021). [Online]. Available: http://portal.acm.org/citation.cfm?doid=1150343.1150381

[13] F. Harary, *Graph Theory*, ser. Addison-Wesley series in mathematics. Addison-Wesley Publishing Company, 1969.

[14] L. S. da Rosa, F. R. Schneider, R. P. Ribas, and A. I. Reis, “Switch level optimization of digital CMOS gate networks,” in *2009 10th International Symposium on Quality of Electronic Design*. San Jose, CA, USA: IEEE, Mar. 2009, pp. 324–329, (Última fecha de consulta: 03 noviembre 2021). [Online]. Available: http://ieeexplore.ieee.org/document/4810315/

[15] C. Mead and L. Conway, *Introduction to VLSI systems*. United State of America: Addison-Wesley Publishing Company, Oct. 1980.

[16] R. Reis, “Concepção de Circuitos Integrados,” Porto Alegre, Brazil, (Última fecha de consulta: 23 noviembre 2021). [Online]. Available: https://cic.unb.br/~rjacobi/ensino/Microeletronica/Concepcao3.pdf

[17] C. M. d. O. Conceicao and R. A. d. L. Reis, “Transistor Count Reduction by Gate Merging,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, pp. 2175–2187, Jun. 2019, (Última fecha de consulta: 03 noviembre 2021). [Online]. Available: https://ieeexplore.ieee.org/document/8698854/

[18] V. P. Correia, A. I. Reis, and P. Alegre, “Classifying n-Input Boolean Functions,” p. 10. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.6167&rep=rep1&type=pdf#:~:text=This%20way%2C%20the%20set%20of,aspects%20of%20Boolean%20function%20implementation.

[19] G. H. Smaniotto, M. T. Moreira, A. M. Ziesemer, F. S. Marques, and L. S. da Rosa, “Toward better layout design in ASTRAN CAD tool by using an efficient transistor folding,” in *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*. Abu Dhabi, United Arab Emirates: IEEE, Oct. 2016, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/7870097/

[20] A. Ziesemer, R. Reis, M. T. Moreira, M. E. Arendt, and N. L. V. Calazans, “Automatic layout synthesis with ASTRAN applied to asynchronous cells,” in *2014 IEEE 5th Latin American Symposium on Circuits and Systems*. Santiago, Chile: IEEE, Feb. 2014, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/6820314/