UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DESARROLLO Y APLICACIÓN DE METODOLOGÍA PARA LA EVALUACIÓN DE TÉCNICAS DE REDUCCIÓN DE POTENCIA DINÁMICA EN CIRCUITOS INTEGRADOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

DIEGO BENAVENTE MARTINEZ

PROFESOR GUÍA:
RONALD VALENZUELA FICA

MIEMBROS DE LA COMISIÓN:
RICARDO FINGER CAMUS
MARCOS DIAZ QUEZADA

SANTIAGO, CHILE
2020

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO
POR: **DIEGO BENAVENTE MARTINEZ**
FECHA: 04/03/2020
PROF. GUÍA: RONALD VALENZUELA FICA

DESARROLLO Y APLICACIÓN DE METODOLOGÍA PARA LA EVALUACIÓN DE TÉCNICAS DE REDUCCIÓN DE POTENCIA DINÁMICA EN CIRCUITOS INTEGRADOS

En la industria de semiconductores actual existe una gran demanda por productos de bajo consumo debido al constante desarrollo de la tecnología móvil, así como aplicaciones con restricciones fuertes de potencia como la computación de alto rendimiento en grandes centros de datos [11]. Por lo tanto, hay una necesidad de producir herramientas EDA (*Electronic Design Automation*) que optimicen el consumo de potencia de los futuros *chips*, aplicando técnicas de reducción que apuntan a corregir aspectos cada vez más sutiles en los que se encuentra que se desperdicia energía. Dado que el diseño y fabricación de circuitos integrados es un esfuerzo que requiere de grandes cantidades recursos humanos y económicos, existe un alto nivel de interés por evaluar estos métodos tanto en términos de su efectividad como efecto en otras métricas de calidad de resultados.

En el presente trabajo se desarrolla una metodología de evaluación basada en la implementación y simulación de un diseño de pruebas desde su descripción en RTL (*Register-Transfer Level*) hasta obtener el *layout* del circuito. Además, se cumple con una serie de requerimientos como permitir la modificación del diseño, flujo de implementación, estímulos de simulación y entregar la mayor flexibilidad en cuanto a instrumentalización para realizar distintos tipos de análisis. Para lograr esto se utiliza el proyecto OpenPiton que consiste en el diseño de un procesador *manycore* de código abierto. Se genera el flujo de implementación en las herramientas de síntesis que se desea evaluar, asi como flujos de análisis de potencia con herramientas de *sign-off* especializadas y simulación *Full Timing Gate Level* de la cual se obtienen datos de actividad de la mayor precisión que se utilizan como referencia en la metodología.

Para finalizar se aplica la metodología a dos casos de estudio donde el primero consiste en evaluar el desempeño de un estimador de *glitch* para la herramienta de síntesis en donde se encuentra que la correlación de actividad de *glitch* del simulador con respecto a la estimada es alta con un error promedio de 0.04 %, y que, sin embargo, la correlación de potencia es baja con un error promedio de 67.68 %. Luego para el segundo caso, en el que se evalúan técnicas asociadas a *self gating* se obtuvo una reducción de potencia de 9.44 % al aplicar selección automática de compuerta detectora con un incremento de 1.31 % en cobertura de *self gating*, y al aplicar un flujo alternativo de análisis de potencia se obtiene una disminución promedio de 35.01 % en el tiempo que las *self gates* están activas con una disminución en el error de propagación del 52.65 % en promedio, un aumento de cobertura de 4.34 % y una reducción de 6.42 % de la potencia dinámica.

Agradecimientos

Agradezco a mi familia por apoyarme siempre y darme la oportunidad de completar esta carrera.

A los integrantes del grupo Application Engineering de Synopsys por toda la ayuda en esta etapa, especialmente a Ronald por ser mi profesor guía y su gran contribución a la realización de esta memoria.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación

Desde los inicios de la era digital la industria se desarrolló basándose en la creación de dispositivos de cada vez mayor desempeño y densidad. Así en esta primera etapa la necesidad de una mayor capacidad de procesamiento y velocidad llevó que se perfeccionaran las capacidades de manufactura de dispositivos semiconductores, para que permitiesen integrar mayor cantidad de transistores por unidad de área a la vez que se incrementaba la velocidad de estos [9]. Por otro lado, se concibieron nuevas arquitecturas de mayor complejidad como el paralelismo en el caso de los procesadores. Esta etapa se enfocó en el desarrollo de componentes de alto desempeño para uso en grandes y costosos computadores centrales [11].

Luego, durante los inicios de la década de 1990, cuando se buscaba mejorar el rendimiento de los computadores portátiles, se tuvo que tomar en consideración en el diseño de los *chips* la restricción de potencia que imponía la operación alimentada por baterías. En este periodo se comenzó a gestar el proceso de modificar las herramientas de diseño para optimizar adecuadamente el consumo [5] [3]. Para mediados de la década del 2000 el creciente mercado de los celulares, que iban adquiriendo cada vez más funcionalidad, había continuado impulsando el diseño consciente de la energía y el nivel de complejidad que alcanzaron los procesadores provoco que se topara con la *pared de energía*, es decir, se alcanzó el límite de aproximadamente 150W correspondiente a la potencia que puede ser disipada por un sistema de refrigeración de bajo costo. Lo anterior produjo un cambio en el flujo de diseño de los procesadores de alto rendimiento que hasta la fecha consideraban el objetivo de incrementar la velocidad como primordial [11].

Posteriormente se consolida la importancia de las técnicas de reducción de potencia en *chips* cuando la expansión Internet y las telecomunicaciones produjo la aparición de los centros de datos, que aprovechan las economías de escala para reducir costos a la vez que alcanzan gran eficiencia en espacio y facilidad de operación [10]. En este escenario los fabricantes debieron superar el desafió energético, tanto de consumo como de disipación de calor, que suponía operar cientos de procesadores en un espacio reducido. Si a esto se le suma la explosiva proliferación del *smartphone* para la década del 2010, está claro que la especificación de potencia debía pasar a ser de alta prioridad, al mismo nivel que desempeño y densidad, trayendo consigo un cambio de paradigma en las herramientas para integrar este parámetro

en todos los niveles del flujo de diseño de circuitos integrados. [13]

En el contexto actual, producto de la evolución resumida anteriormente, existe un gran abanico de técnicas de reducción de potencia que abarcan el proceso completo de diseño e implementación de circuitos digitales. Estás son soportadas por las herramientas de automatización de diseño electrónico y corresponden a metodologías para producir que estás sean conscientes del gasto energético y lo incorporen a la función objetivo lo que se traduce en una mayor complejidad en el sentido de como interactúan esta variable con otras como velocidad y área. Además, la reducción que permiten alcanzar las técnicas depende considerablemente del tipo de diseño y a las otras restricciones que lleven asociadas. El presente trabajo se realizará utilizando las herramientas de verificación e implementación de Synopsys, donde se requiere un estudio de estas características para orientar la evolución de sus productos de implementación automatizada de circuitos integrados.

1.2. Objetivos

1.2.1. Objetivo general

El objetivo general de la memoria consiste en el desarrollo y aplicación de una metodología para evaluar técnicas de reducción de potencia en la implementación asistida de circuitos integrados. Dicha metodología debe generar recomendaciones que permitan guiar el desarrollo del producto, las cuales abarcan aspectos como la efectividad de las técnicas para reducir el consumo, efectos secundarios en otras métricas de desempeño del *chip* y exactitud de los algoritmos de análisis.

1.2.2. Objetivos específicos

Los objetivos específicos de este trabajo son:

1. Obtener un conjunto de diseños de prueba sobre los que realizar las evaluaciones.

2. Definir figuras de mérito a considerar en la evaluación y la forma de obtenerlas en el flujo de diseño.

3. Realizar la implementación de los diseños aplicando diversas técnicas de reducción de potencia dinámica.

4. Analizar los resultados y caracterizar del desempeño de las técnicas.

1.3. Estructura de la memoria

La estructura del documento es:

Capítulo 2. Marco teórico: Esta sección consiste en una revisión de los conceptos básicos para entender el trabajo. Además, se entregará una descripción general del flujo de diseño de circuitos integrados, así como, de las figuras de mérito que se utilizarán para la metodología de evaluación.

Capítulo 3. Metodología: Se detalla el proceso para obtener la metodología de evaluación. Esto incluye la obtención de diseños de prueba y su configuración para los propósitos del trabajo, además de la discusión de las figuras de mérito, donde se explicará la elección de estas y como se obtienen a partir del flujo de diseño.

Capítulo 4. Resultados y análisis: En este capítulo se verán los resultados de aplicar la metodología sobre un conjunto de técnicas que se desea evaluar para luego analizar los datos y categorizarlas.

Capítulo 5. Conclusiones: Finalmente se extraerán conclusiones de los análisis realizados, obteniéndose el objetivo final consistente en una propuesta de recomendaciones para el desarrollo de la herramienta. También se verán las limitaciones de la metodología planteada y las posibilidades de extenderla en trabajos futuros.

Capítulo 2

Marco Teórico

2.1. Flujo de diseño de circuitos integrados

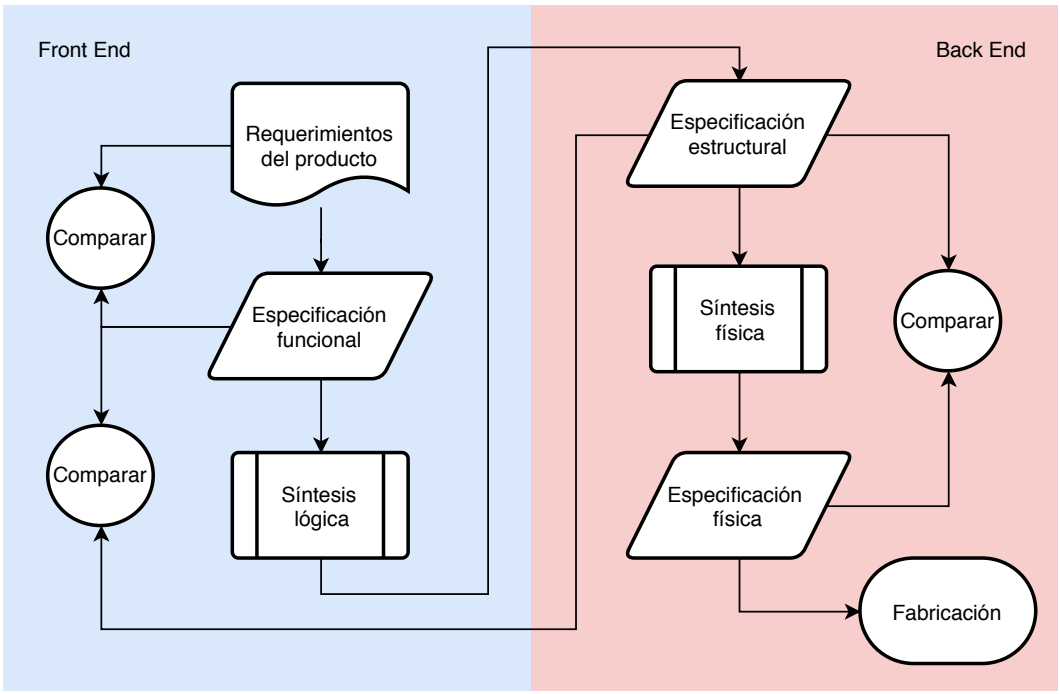


Figura 2.1: Flujo generalizado de diseño de circuitos integrados.

El flujo de diseño es una metodología para diseñar un circuito integrado a partir de un conjunto de requerimientos, lográndose una implementación sin errores que cumple con una serie de restricciones de eficiencia y desempeño. A continuación, se detallan las etapas que constituyen el flujo según la bibliografía especializada. [11]

2.1.1. Diseño lógico a nivel transferencia entre registros

El flujo inicia con la definición de un conjunto de requerimientos para cumplir el objetivo del producto final, estos fundamentalmente son la descripción funcional, parámetros de

desempeño y restricciones físicas. Luego los diseñadores hacen la especificación del circuito en un lenguaje de descripción de hardware (HDL), como VHDL o Verilog entre otros. Lo que se obtiene de esta etapa es el circuito a nivel de transferencia entre registros (RTL) y es independiente de la implementación.

2.1.2. Simulación dinámica

En esta etapa se verifica que el circuito RTL cumpla con los requerimientos funcionales del producto correctamente y sin errores, lo que se realiza mediante una simulación temporal del diseño. Para esto se debe desarrollar un código en HDL que sirva de banco de pruebas. En este se debieran generan entradas del circuito que sean capaces de ejercitarlo completamente, a modo de encontrar cualquier error potencial. Sin embargo, los diseños actuales son de tal complejidad que esto no es factible por lo que se tiende a utilizar un subconjunto de vectores de entrada elegidos cuidadosamente. Para ejecutar este paso se utiliza el software VCS, que permite compilar y ejecutar simulaciones a partir descripciones en Verilog, VHDL, SystemVerilog, entre otros.

2.1.3. Síntesis lógica

Una vez verificado el diseño se procede a sintetizar el circuito RTL para obtener una interconexión de compuertas conocida como *netlist*, para esto se utiliza software especializado como Design Compiler o Fusion Compiler. Además, en esta etapa se optimiza el diseño para cumplir con las restricciones de velocidad, área y potencia, se mapean las compuertas a celdas específicas de la tecnología que se desea utilizar y se inserta lógica con el propósito exclusivo de verificar el *chip* una vez fabricado. Para la realización exitosa del proceso de síntesis es importante contar con una biblioteca de celdas caracterizada en términos de retardos, área y potencias en todos los puntos de operación en los que se desempeñara circuito integrado.

2.1.4. Verificación formal

Debido a que en la etapa anterior se realizan variadas optimizaciones al circuito RTL es necesario verificar que exista consistencia lógica con la *netlist*. Para esto se utilizan herramientas de verificación formal, como el software Formality, que verifican equivalencia lógica entre circuitos a nivel matemático, es decir, de forma estática y con validez para todas las posibles entradas del diseño.

2.1.5. Análisis estático de restricciones de tiempo

Luego de verificar la funcionalidad del circuito sintetizado se procede a analizar su funcionamiento con respecto a la evolución temporal de las señales, esto es, que el diseño cumpla con los requerimientos de velocidad correctamente. Este tipo de análisis se dice que es estático debido a que al igual que el paso anterior es independiente de las entradas y por lo tanto es válido para todos los estados del circuito, siendo también mucho más eficiente en costo

computacional que una simulación dinámica.

En su ejecución se utilizan los retardos nominales de las compuertas obtenidos de la biblioteca utilizada para síntesis lógica y una estimación de los retardos de las rutas, pues en esta etapa aún no se conoce el enrutamiento definitivo. Con estos retardos la herramienta calcula los retardos de todos los caminos posibles en el circuito para luego verificar que se cumplan las restricciones de *setup* y *hold* en cada elemento secuencial. El tiempo de *setup* corresponde al mínimo tiempo que la señal debe estar estable antes del flanco activo de reloj para que el dato pueda ser capturado correctamente, mientras que el tiempo de *hold* es la mínima cantidad de tiempo que la señal debe estar estable después del flanco activo de reloj.

Los resultados del análisis de tiempos se reportan utilizando el parámetro de holgura para cada camino combinacional en el circuito, este se define como el margen entre el tiempo de propagación calculado y el tiempo de *setup* o *hold* según corresponda. Una holgura negativa indica que el diseño tiene violaciones de tiempo que deben ser corregidas. Así los indicadores más usados son la peor holgura negativa (*Worst Negative Slack* o WNS), que pertenece al camino crítico, y la holgura total negativa (*Total Negative Slack* o TNS) que indica que tan factible es realizar el diseño con las restricciones de tiempo impuestas. Para efectuar este análisis se utiliza el software PrimeTime.

2.1.6. Análisis de potencia

En este paso se analiza el consumo de potencia del diseño y se verifica la conformidad con el requerimiento energético fijado al principio del flujo, de no cumplirse se debe retroceder y hacer las modificaciones que correspondan. Este análisis se realiza en una forma parecida a las dos etapas anteriores, es decir, en forma estática, sin utilizar simulación temporal pues sería muy costosa de procesar.

Para elaborarlo se utiliza el software PrimePower y se requiere conocer la *switching activity* de cada nodo del circuito, este parámetro corresponde a la probabilidad estática del estado del nodo, es decir el porcentaje de tiempo que el nodo está en 0-lógico o 1-lógico, y la tasa de conmutación que es simplemente la cantidad de transiciones por segundo. Estos valores pueden obtenerse desde un archivo SAIF (*Switching Activity Interchange Format*) generado por simulación dinámica con los vectores de prueba de la verificación funcional, o bien, se pueden estimar mediante propagación a partir de valores fijados manualmente o con los valores por defecto estimados por la herramienta.

Una vez que se tiene la *switching activity* del circuito se procede a calcular tanto la potencia estática (a partir de las especificaciones para cada celda de la biblioteca y las probabilidades estáticas), como la potencia dinámica (a partir de las tablas de consumo interno de las celdas, estimaciones de las capacitancias en las rutas y las velocidades de conmutación). Con estos valores es posible entregar un reporte de las potencias estáticas y dinámicas en el circuito identificando además los nodos críticos en los que se consume mayor potencia dinámica, así como los módulos que consumen mayor potencia estática.

2.1.7. Restricciones físicas

Antes de realizar la síntesis física es necesario especificar un plano de las ubicaciones y formas de las regiones para el posicionamiento de módulos y macro-celdas como memorias RAM o módulos analógicos. También deben especificarse las ubicaciones de los puertos o *pads* de entrada-salida del *chip*. Este paso es importante pues se determinarán limites en área, potencia, velocidad, así como evitar congestión al ubicar los módulos de forma inteligente. Por ejemplo, es evidente que los módulos de comunicación con el exterior deberían, en lo posible, ser adyacentes a los *pads* que utilicen, así como a los módulos internos a los que estén conectados.

2.1.8. Síntesis física

El objetivo de esta etapa consiste en obtener una implementación física o *layout* del circuito, esto corresponde a un conjunto de polígonos que especifican la geometría de las distintas capas del circuito integrado, esto incluye tanto las celdas cuya geometría esta especificada en la biblioteca, como las interconexiones o rutas que conforman los nodos. El resultado final son los planos para la fabricación de máscaras para los procesos de litografía con los que se fabricará el *chip*. La síntesis física consiste principalmente en los procesos de ubicación y enrutamiento.

Este proceso se lleva a cabo en softwares como IC Compiler o Fusion Compiler y conllevan una alta carga de procesamiento pues son procesos de optimización de complejidad considerable. Esto se debe en parte a que se deben cumplir con las restricciones de tiempo, potencia y área, lo que se logra en múltiples iteraciones y el proceso puede extenderse considerablemente dependiendo de la complejidad y los requerimientos del diseño. En cada iteración se estiman los valores de resistencia y capacitancia de las rutas para calcular los retardos y otros parámetros que se comparan con las restricciones, repitiendo hasta que se cumplen los objetivos.

2.1.9. Extracción de parásitos

Con el diseño implementado físicamente se pueden extraer los valores de resistencia y capacitancia de cada ruta. Para lo que se utilizan herramientas de software como StarRC.

2.1.10. Sign-off

Finalmente, y teniendo valores precisos para los elementos parásitos del circuito, se vuelven a realizar los análisis de verificación formal, tiempos y potencia para verificar la conformidad del diseño con las restricciones impuestas antes de pasar a la etapa de fabricación. En caso de no cumplir con alguna de estas se debe retroceder en el flujo y realizar las modificaciones pertinentes generándose una nueva iteración del diseño.

2.2. Consumo en dispositivos CMOS

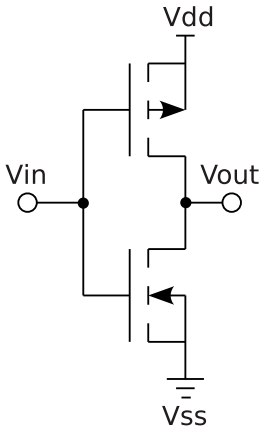


Figura 2.2: Diagrama del negador CMOS.

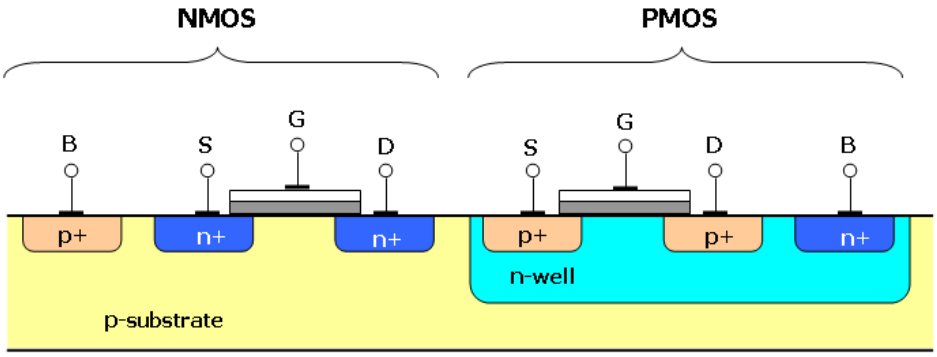


Figura 2.3: Sección transversal de transistores CMOS, cuyos terminales son: substrato B (*Body*), fuente S (*Source*), puerta G (*Gate*) y drenaje D (*Drain*)

Primero se definirán las magnitudes físicas con las que se cuantifica el consumo en circuitos electrónicos, estas son potencia instantánea, energía y potencia media. La potencia instantánea que consume o provee un elemento circuital se calcula como el producto del voltaje y la corriente sobre el elemento:

$$P(t) = V(t)I(t) \tag{2.1}$$

La energía es la integral de la potencia instantánea en el tiempo T que toma ejecutar una acción:

$$E = \int_0^T P(t)dt \tag{2.2}$$

Y la potencia media sobre el mismo intervalo de tiempo es:

$$P = \frac{1}{T}\int_0^T P(t)dt \tag{2.3}$$

$$\implies P = \frac{E}{T} \tag{2.4}$$

La potencia disipada en un circuito integrado se puede descomponer en dos fuentes conocidas como potencia estática y potencia dinámica. La potencia estática es la que se disipa cuando el circuito está en estado inactivo, es decir, los niveles de las entradas se mantienen constantes en el tiempo, mientras que la potencia dinámica es la que se consume cuando se generan transiciones en el circuito. Entonces la expresión para la potencia total es:

$$P_{\text{total}} = P_{\text{estática}} + P_{\text{dinámica}} \tag{2.5}$$

En las siguientes secciones se detallarán los modelos, interdependencias y métodos de cálculo de las componentes mencionadas. Para efectos ilustrativos y de análisis puede considerarse al circuito negador de la figura 2.2 como representativo de la tecnología CMOS, esto debido a que es la compuerta más básica y las demás siguen la misma estructura aumentando la cantidad de transistores. Además, en la figura 2.3 se muestra la sección transversal para un dispositivo CMOS fabricado.

2.2.1. Potencia estática

La disipación de potencia estática proviene de tres fuentes principales:

Corriente sub-umbral: Corriente conducida entre drenaje y fuente cuando el transistor esta apagado $V_{gs} < V_{th}$, se produce debido a la existencia de un transistor bipolar parásito entre fuente, substrato y drenaje cuyos dopajes son tipo N, P y N (NMOS) o P, N y P (PMOS) respectivamente. La base de este transistor bipolar es controlada por un divisor de tensión capacitivo formado entre el electrodo de puerta, la capa de oxido, la zona de agotamiento que se produce en el substrato en la región cercana a la capa de oxido cuando el MOSFET está en modo de operación sub-umbral o inversión débil $V_{gs} < V_{th}$ y el resto del substrato cuya tensión está dada por el puerto de cuerpo. A pesar de que esta corriente es generalmente pequeña su efecto cobra relevancia en dispositivos VLSI que contienen millones de transistores.

Corriente de puerta por efecto túnel: Esta corriente fluye a través de la capa de oxido presente entre el puerta y el canal. Se debe al efecto túnel que se produce a escala cuántica, en donde la naturaleza ondulatoria de la materia permite que una porción de las cargas acumuladas en el gate atraviese la barrera.

Corriente de fuga de juntura en polarización inversa: Corresponde a la corriente de fuga presente en las junturas PN que se forman entre fuente-drenaje y el sustrato, esta se debe a que en polarización inversa existe una pequeña corriente de cargas minoritarias.

Por otro lado, la potencia estática tiene varias interdependencias con otros parámetros del circuito, tanto de las condiciones de operación como de ciertas características propias de la tecnología y celdas que se utilicen. A continuación, se nombrarán las más importantes:

Tensión umbral: Emplear transistores con tensión umbral más alta disminuirá el efecto de conducción sub-umbral, sin embargo, estos también tendrán un desempeño más pobre en velocidad ya que su capacidad de conducción es menor.

Largo de canal: Los transistores con mayor largo de canal poseen menores corrientes de fuga, así como una menor capacidad de conducción de corriente lo que se traduce en un funcionamiento más lento. Por lo tanto, en el proceso de diseño de transistores se optimiza el largo de canal de acuerdo con requerimientos de velocidad, potencia y las capacidades de la tecnología de fabricación.

Temperatura: Al aumentar la temperatura la corriente sub-umbral se incrementa en forma no lineal, aumentando por lo tanto la potencia estática. [4]

Proceso: La potencia estática tiene una fuerte dependencia con las variaciones en el proceso de fabricación. Durante la manufactura de semiconductores existen variaciones en los parámetros de los transistores tanto entre lotes como dentro de un mismo chip. A grandes rasgos los parámetros que presentan variaciones son el voltaje de umbral, movilidad de electrones y parásitos RC [12]. Para considerar estas variaciones en la implementación de un chip se caracterizan las celdas en tres puntos denominados: "lento", "típico" y "rápido", donde el nombre se refiere a la velocidad de las celdas (retardo de propagación) con respecto al valor nominal ("típico") [19]. Luego debido a las variaciones en el voltaje umbral la potencia estática disipada es mayor para el proceso "rápido" y menor en el "lento".

Para modelar este tipo de potencia en las herramientas de diseño se utiliza un método que consiste en simular un conjunto de celdas a nivel de transistor en forma analógica, para luego extraer la potencia estática, luego esta se anota en la biblioteca en un archivo Liberty [20]. En este formato se puede especificar tanto la potencia estática total de la celda como valores distintos según el estado o valores de las entradas.

2.2.2. Potencia dinámica

La potencia dinámica en un circuito CMOS tiene dos fuentes principales: potencia de conmutación interna y potencia de carga de salida, las que serán abordadas a continuación.

Potencia de conmutación interna: Corresponde a la potencia que disipa internamente la celda al producirse un cambio de estado en alguna de sus entradas. La principal causa es la corriente de corto circuito, ilustrada en la figura 2.4 (a), esta se produce en durante la conmutación cuando el voltaje en la entrada se encuentra en el intervalo entre los voltajes de umbral de los transistores PMOS y NMOS ya que la transición no es instantánea. En este intervalo ambos transistores se encuentran parcialmente

encendidos lo que permite el flujo de la corriente, además mientras menor sea la velocidad de conmutación o *slew rate* mayor será la potencia disipada. Además, existe una potencia asociada al cambio de estado en las entradas, aunque no se produzca una transición en las salidas. Finalmente, dentro de esta componente además se consideran las capacitancias y resistencias parásitas asociadas a las interconexiones internas de la celda.

Potencia de carga de salida: Potencia requerida para cargar y descargar las capacitancias de puerta en las entradas de otras celdas tal como se aprecia en la figura 2.4 (b), también se debe considerar la capacitancia parásita de las interconexiones externas a la celda. Esta potencia está dada por la ecuación:

$$P_{\mathrm{OCP}} = C_L V_{\mathrm{DD}}^2 f a \tag{2.6}$$

Donde C_L corresponde a la capacitancia total vista por la salida de la celda (capacitancias de entrada del *fan out* más la capacitancia parásita de la interconexión), V_{DD} es el voltaje de alimentación, f es la frecuencia de reloj y a es el factor de actividad que es la probabilidad de que una señal conmute en cierta cantidad de ciclos de reloj. Estos parámetros son importantes pues el principio de funcionamiento de la mayoría de las técnicas de reducción de potencia consiste en minimizar alguno de ellos.

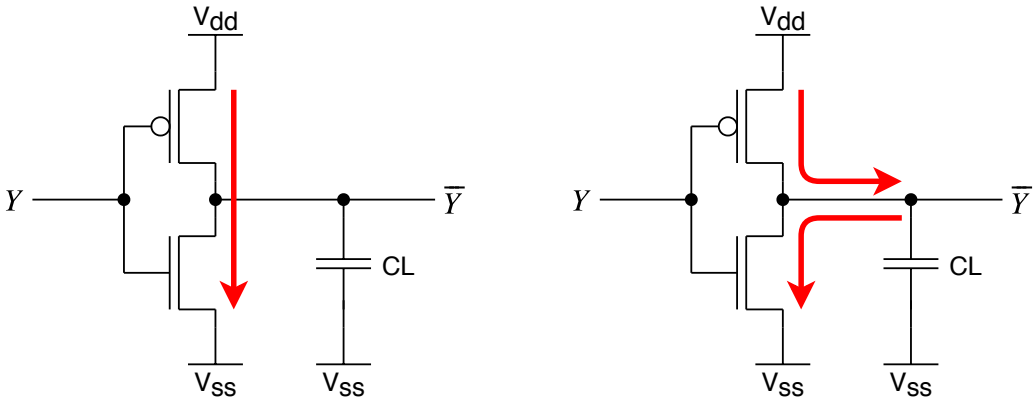


Figura 2.4: Principales flujos de corriente en potencia dinámica, a la izquierda potencia de conmutación interna (a) y a la derecha potencia de carga de salida (b).

De las componentes de potencia dinámica mencionadas anteriormente solo la primera es dependiente de las características propias de la celda. Potencia de carga en la salida solo depende de las condiciones de operación del circuito y de las capacitancias externas a la celda. Por lo tanto, a continuación, se describirán las relaciones de interdependencia con los parámetros de celda para la potencia de conmutación interna:

Tensión umbral: Celdas con menor tensión umbral tienen una mayor disipación de potencia de conmutación interna que las de alta tensión umbral. Esto se debe a la corriente de corto circuito que se produce cuando el voltaje umbral es menor a $\frac{V_{dd}}{2}$ al existir un rango de voltaje de entrada en el que ambos transistores están encendidos. [11]

Largo de canal: Dispositivos con canal más largo logran menor potencia de conmutación interna que los de canal corto. Esto pues los transistores de canal corto tienen menor impedancia de salida lo que produce mayores corrientes de corto circuito. [11]

Temperatura: A mayor temperatura se reduce la capacidad de conducción de los transistores por lo tanto las transiciones son más lentas, incrementando el consumo de potencia interna. Notar que esto afecta de mayor manera al proceso "rápido", con celdas de bajo tensión umbral y voltaje de alimentación mayor al nominal. [4] [11]

Proceso: Para los parámetros de proceso "rápido" se produce un mayor consumo de potencia de conmutación interna. Este efecto se produce al elevarse la capacidad de conducción. [4] [11]

Además, se debe mencionar que esta componente depende principalmente del voltaje de alimentación VDD siguiendo una ley cuadrática y que también tiene dependencia con el tamaño de celda pues se incrementa a mayor tamaño.

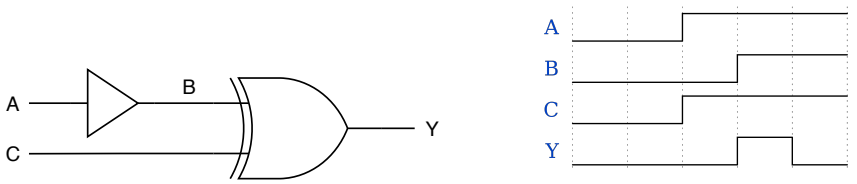
2.3. Glitch

Un *glitch* es un pulso no funcional que se produce cuando las transiciones de las señales de entrada de una compuerta no llegan simultáneamente y según tabla de verdad de la compuerta se produce un pulso en la salida. Se dice que este pulso no es funcional en el sentido de que no forma parte de la función lógica especificada, es decir, no se produce cuando no se consideran los retardos de compuertas e interconexiones.

Considerando lo anterior los *glitch* pueden constituir una fuente de error lógico, sin embargo, en circuitos sincrónicos esto no es un problema grave pues los registros se encargan de filtrar *glitches* entre las etapas combinacionales de un *pipeline*. No obstante, en circuitos asíncronos, o en ciertas partes asíncronas dentro de un circuito sincrónico, como la entrada de reloj de un registro, un *glitch* podría producir un error lógico.

Desde el punto de vista del consumo de potencia las transiciones producidas por *glitch* se pueden considerar como problemáticas pues consumen potencia dinámica de la misma forma que las funcionales, por lo que conforman un gasto de energía innecesario al aumentar la actividad en el circuito. Para facilitar el análisis de *glitch* en el contexto de análisis de potencia estos se clasifican en dos categorías, *glitches* de transporte y *glitch* inerciales [1], los cuales serán revisados en las secciones siguientes.

2.3.1. Glitch de transporte

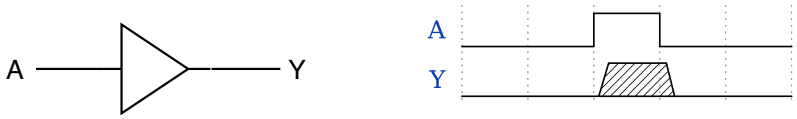


(a) Circuito de ejemplo. (b) Diagrama de tiempo.

Figura 2.5: *Glitch* de transporte.

Esta clase de *glitch* es la que se produce debido a un desbalance entre los retardos de propagación de cada pin de entrada de una compuerta lógica, es decir, las transiciones no llegan simultáneamente a cada *pin* por lo tanto, y sujeto a la tabla de verdad de la compuerta, se pueden producir pulsos indeseados en la salida. Lo anterior queda mejor representado por la figura 2.5, donde en el caso ideal o *zero delay* no debiera producirse ningún cambio en la salida cuando las entradas A y C transición de 0 a 1 simultáneamente, no obstante, cuando se consideran los retardos de propagación el buffer genera una diferencia momentánea en las entradas del B y C del XOR por lo que aparece un *glitch* en la salida. Es importante notar que un *glitch* de transporte es por definición un pulso entre niveles 0-1-0 o 1-0-1 por lo tanto la duración de este es mayor al retardo de propagación de la compuerta que lo produce. En caso contrario se produce un *glitch* inercial como se verá a continuación.

2.3.2. Glitch inercial

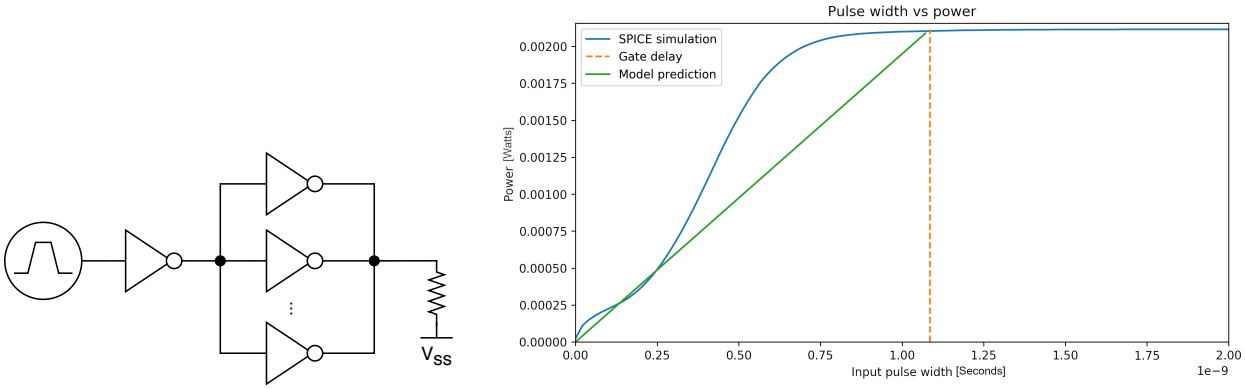


(a) Circuito de ejemplo. (b) Diagrama de tiempo.

Figura 2.6: *Glitch* inercial.

En este caso el *glitch* se produce cuando un pulso llega a una compuerta y tiene una duración menor al retardo de propagación de esta, por lo tanto, la salida realiza una transición, pero no alcanza a llegar a un valor lógico estable antes volver a conmutar y el pulso en la salida se modela como 0-X-0 o 1-X-1. Esto se ilustra en la figura 2.6 donde el pulso en A es de menor ancho que el retardo de propagación del buffer.

2.3.3. Modelamiento de potencia en glitch inercial



(a) Circuito simulado en HSPICE.

(b) Potencia total vs ancho de pulso.

Figura 2.7: Medición de potencia dinámica consumida por *glitch* inercial.

En la figura 2.7 se muestra un circuito de pruebas simulado en forma analógica con HSPICE para medir la potencia consumida por *glitch* inercial al variar el ancho de pulso en la entrada de la compuerta. Este circuito consiste en un generador de pulsos con ancho variable, una compuerta NOT en la que se mide la potencia entregada por la fuente de alimentación y un numero arbitrario de compuertas NOT en paralelo para simular el *fan out* o carga. Todas las compuertas fueron implementadas con 2 de transistores en la configuración clásica CMOS ilustrada en la figura 2.2. El resultado permite comprobar la utilidad del parámetro IK del formato SAIF para calcular la equivalencia entre *glitch* inercial y transiciones funcionales.

$$\mathrm{IK} = \frac{\sum_{i}^{k} \sum_{j}^{N_i} \frac{\delta_{ij}}{T_i}}{\sum_{i}^{k} N_i} \tag{2.7}$$

El factor de conversión definido por el estándar IEEE 1801 [1] se muestra en la ecuación 2.7. Este está generalizado para una compuerta con k retardos, uno para cada camino posible desde una entrada hacia una salida, donde cada retardo tiene un valor T_i con $i = 1 \ldots k$ y produce N_i pulsos de tipo *glitch* inercial dada una diferencia o desfase en el tiempo de arribo de las entradas δ_{ij} que genera cada *glitch* $j = 1 \ldots N_i$. Luego la potencia disipada por N *glitches* inerciales se modela según la ecuación 2.8.

$$\mathrm{P} = 2 \times \mathrm{IK} \times N \times P_0 \tag{2.8}$$

Donde P_0 corresponde a la potencia disipada por una transición funcional típica. Para el caso de la compuerta NOT de la figura 2.7 estas ecuaciones se reducen a:

$$\mathrm{IK} = \frac{\tau}{T_p} \tag{2.9}$$

$$P = \tau \times \frac{2P_0}{T_p} \tag{2.10}$$

En la que τ es el ancho de pulso generado y T_p es el retardo de la compuerta. Finalmente, la ecuación 2.10 se incluye en el gráfico de la figura 2.7 como potencia precedida por el modelo, y se puede concluir, para este caso, las curvas de valor estimado y lo resultado de simulación analógica son consistentes.

2.4. Análisis de potencia en VLSI

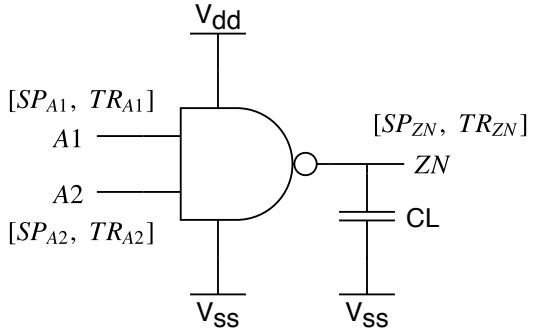


Figura 2.8: Ejemplo de celda NAND con anotaciónes de actividad, probabilidad estática SP (*Static Probabilty*) y tasa de conmutación TR (*Toggle Rate*)

En esta sección se revisarán las técnicas de análisis de potencia en circuitos integrados VLSI. Para esto se utilizará como ejemplo la celda NAND de la figura 2.8, la cual será analizada con una metodología similar a la que usan las herramientas de diseño automatizadas. [4]

Para comenzar se debe definir formalmente el concepto de *switching activity*, este corresponde a una métrica de la actividad de conmutación de cierto nodo, pin o puerto del circuito que se utiliza para realizar análisis de potencia en forma estática lo que permite obtener resultados en un tiempo de procesamiento mucho menor al requerido si se calculara a partir de las formas de onda resultantes de una simulación digital o analógica del circuito. Lo anterior claramente significa una pérdida de precisión con respecto a los métodos más lentos, sin embargo, la estimación a probado ser lo suficientemente exacta en las etapas iniciales del flujo de diseño [4], además las herramientas de optimización requieren de un método de cálculo rápido para evaluar la función objetivo y realizar las iteraciones necesarias en un tiempo razonable. Por los métodos basados en simulación temporal se utilizan en etapas finales del

diseño como para realizar *sign-off* lo que requiere un análisis preciso para determinar si el diseño cumple con las restricciones de potencia.

$$\begin{aligned} \text{Toggle rate} &= \frac{\text{TC}}{T} \\ \text{Static probability} &= \frac{T_1}{T} \end{aligned} \tag{2.11}$$

El *switching activity* como métrica está compuesto por dos campos conocidos como *toggle rate* o velocidad de conmutación y *static probability* o probabilidad estática, cuya definición se presenta en las ecuaciones 2.11 donde TC representa la cantidad de conmutaciones en el nodo, T representa el periodo de tiempo en el que se mide y T_1 es el tiempo que la señal está en estado 1 lógico. Por ejemplo, para un reloj de frecuencia 100MHz y ciclo de trabajo 50 % se tienen 2 conmutaciones en 10ns (periodo), por lo tanto, el *toggle rate* será de 200 millones de transiciones por segundo, y dado que la señal esta encendida durante 5ns de 10ns el *static probability* de 0.5.

El principal método para intercambiar datos de *switching activity* corresponde al formato SAIF (*Switching Activity Interchange Format*), definido en el estándar IEEE 1801 [1]. Este se puede generar a partir de simulación o bien mediante estimación en herramientas de síntesis o análisis de potencia. Este archivo de texto plano contiene una estructura tipo árbol de datos que representa las jerarquías dentro del diseño, como los distintos módulos que los componen. Los nodos hoja en este árbol corresponden a los nodos (nets), puertos y pines del circuito, en los cuales se anotan los datos especificados en la tabla 2.1. Además, en el encabezado del archivo se anotan los datos referentes a la simulación o escenario en que se generaron los datos, de estos los más relevantes para análisis son la duración o ventana de tiempo en la que se capturaron los datos (T en las ecuaciones 2.11) y la escala para los campos de tiempo.

Tabla 2.1: Campos de datos del formato SAIF.

Campo	**Descripción**
TC	Cantidad de conmutaciones funcionales en el nodo o flancos de subida y bajada
T0	Tiempo en estado 0 lógico
T1	Tiempo en estado 1 lógico
TX	Tiempo en estado X lógico
TZ	Tiempo en estado Z lógico
TB	Tiempo en estado contención de bus 2 o más *drivers* activos simultáneamente
TG	Cantidad de conmutaciones debidas a *glitch* de transporte $0 \to 1 \to 0$ y $1 \to 0 \to 1$
IG	Cantidad de conmutaciones debidas a *glitch* inercial $0 \to X \to 0$ y $1 \to X \to 0$
IK	Factor de conversión de *glitch* inercial IK $\times$ IG es la cantidad de conmutaciones funcionales equivalentes

Tabla 2.2: Modelo de potencia de la celda NAND

Símbolo	Parámetro	Condiciones
P_{L0}	Potencia de fuga (Celda)	$!A1 \cdot !A2$
P_{L1}		$!A1 \cdot A2$
P_{L2}		$A1 \cdot !A2$
P_{L3}		$A1 \cdot A2$
P_{In1}	Potencia interna de entrada (Pin A1)	$!A2$
P_{In2}	Potencia interna de entrada (Pin A2)	$!A1$
P_{Out1}	Potencia interna de salida (Pin ZN)	$A2$
P_{Out2}		$A1$

Luego se debe discutir el modelo de consumo para las celdas, mediante el cual se traduce *switching activity* en potencia para cada escenario o conjunto de condiciones de operación. En la tabla 2.2 se pueden ver los principales parámetros del modelo de consumo de potencia para la celda, en la práctica obtenidos desde un archivo Liberty. Las primeras cuatro filas corresponden a la potencia estática disipada en cada combinación de entradas posible. Luego se encuentra la potencia dinámica, la cual se entrega dividida en entradas que no producen un cambio en la salida (potencia interna de entrada) y entradas que sí producen un cambio en la salida (potencia interna de salida). Notar que en el último caso se anota la potencia interna disipada al haber una transición en la salida ZN debido a un cambio en la entrada $A1$ (P_{Out1}) y a un cambio en $A2$ (P_{Out2}) por separado.

Adicionalmente en el modelo se puede especificar la potencia dinámica en los flancos de subida y de bajada por separado del pin correspondiente, sin embargo, estos se promedian para obtener la potencia media ya que por cada flanco de subida existe uno de bajada. Por lo anterior se decidió trabajar con las potencias ya promediadas para simplificar el procedimiento. También se simplificaron los valores de potencia interna pues estos se pueden especificar como matriz para combinaciones de capacitancia de salida y tiempo de transición de entrada.

También debe mencionarse que existen modelos avanzados de potencia como los modelos: *Composite Current Source* (CCS) y *Effective Current Source Model* (ECSM). Estos se basan en la especificación de corrientes de fuga y transientes de la celda. En el caso de las corrientes de fuga se anotan las que provienen desde los pines de alimentación y la corriente que atraviesa la puerta, mientras que en el caso dinámico se entregan las formas de onda de corriente desde la alimentación para distintas capacitancias de salida y tiempos de transición de entrada. La mayor ventaja de los modelos avanzados es que permiten realizar simulaciones dinámicas de la alimentación de las cuales se pueden estimar efectos como transientes de corriente en fuentes de alimentación para dimensionar la capacitancia de filtro (o desacoplo) requerida. [4]

En las secciones siguientes se procederá con el análisis de potencia dividido en sus componentes principales.

2.4.1. Potencia de fuga

Para calcular la potencia de fuga se utilizan las probabilidades estáticas de cada entrada, ver figura 2.8. Con estas se calculan las probabilidades de cada combinación posible y luego se utilizan como ponderadores de las potencias del modelo. Las expresiones son:

$$\mathbb{P}(!A1 \cdot !A2) = (1 - SP_{A1})(1 - SP_{A2}) \tag{2.12}$$
$$\mathbb{P}(!A1 \cdot A2) = (1 - SP_{A1})SP_{A2} \tag{2.13}$$
$$\mathbb{P}(A1 \cdot !A2) = SP_{A1}(1 - SP_{A2}) \tag{2.14}$$
$$\mathbb{P}(A1 \cdot A2) = SP_{A1}SP_{A2} \tag{2.15}$$

Y el total esta dado por:

$$P_L = P_{L0}\mathbb{P}(!A1 \cdot !A2) + P_{L1}\mathbb{P}(!A1 \cdot A2) + P_{L2}\mathbb{P}(A1 \cdot !A2) + P_{L3}\mathbb{P}(A1 \cdot A2) \tag{2.16}$$

2.4.2. Potencia interna

Para calcular esta componente dinámica es necesario descomponer el *toggle rate* de la salida ZN según la contribución de cada entrada. Para esto se utilizan los *toggle rate* de las entradas como ponderadores según las ecuaciones:

$$TR_{A1 \to ZN} = TR_{ZN}\frac{TR_{A1}}{TR_{A1} + TR_{A2}} \tag{2.17}$$
$$TR_{A2 \to ZN} = TR_{ZN}\frac{TR_{A2}}{TR_{A1} + TR_{A2}} \tag{2.18}$$

Con estos valores se puede obtener la porción de los *toggle rate* de las entradas que no producen transiciones en la salida como:

$$TR'_{A1} = TR_{A1} - TR_{A1 \to ZN} \tag{2.19}$$
$$TR'_{A2} = TR_{A2} - TR_{A2 \to ZN} \tag{2.20}$$
$$\tag{2.21}$$

Luego la potencia interna asociada a cada pin de entrada corresponde a la suma de las potencias debido tanto a las transiciones que no afectan a la salida como las que si lo hacen:

$$P_I^{A1} = TR'_{A1}P_{In1} + TR_{A1 \to ZN}P_{Out1} \tag{2.22}$$
$$P_I^{A2} = TR'_{A2}P_{In2} + TR_{A2 \to ZN}P_{Out2} \tag{2.23}$$

Finalmente, la potencia interna de la celda es el total de ambos pines:

$$P_I = P_I^{A1} + P_I^{A2} \tag{2.24}$$

2.4.3. Potencia de conmutación

Esta componente es la potencia requerida para cargar y descargar las capacitancias de entrada de las compuertas del *fan out*, así como la capacitancia parásita de las interconexiones o rutas. Se calcula mediante:

$$P_S = \frac{1}{2} \cdot C_L \cdot Vdd^2 \cdot TR_{ZN} \tag{2.25}$$

Donde C_L corresponde a la capacitancia mencionada anteriormente, la cual se calcula como la suma de la capacitancia equivalente al realizar la extracción de capacitancias y resistencias parásitas a partir de la geometría del circuito y la sumatoria de capacitancias de entrada de cada pin en el *fan out*, las que se obtienen de la biblioteca de referencia. Vdd corresponde al voltaje de alimentación del circuito y TR_{ZN} es el *toggle rate* anotado para el pin ZN. [4]

2.4.4. Potencia total

La potencia total obtenida a partir de lo anterior es:

$$P = P_L + P_I + P_S \tag{2.26}$$

2.4.5. Análisis de potencia mediante herramientas EDA

```
Power           Internal   Switching   Leakage    Total
Group           Power      Power       Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network   8.529e-04 1.599e-05  4.398e-04   8.689e-04(40.56%)   i
register        1.882e-04 3.482e-05  6.666e-08   2.230e-04(10.41%)
combinational   4.068e-04 5.646e-04  1.836e-07   9.716e-04(45.36%)
sequential      2.327e-05 5.522e-05  9.163e-09   7.851e-05( 3.67%)
memory          0.0000    0.0000     0.0000      0.0000(0.00%)
io_pad          0.0000    0.0000     0.0000      0.0000(0.00%)
black_box       0.0000    0.0000     0.0000      0.0000(0.00%)
--------------------------------------------------------------------------------
Net Switching Power  = 6.707e-04 (31.31%)
Cell Internal Power  = 1.471e-03 (68.68%)
Cell Leakage Power   = 2.594e-07 ( 0.01%)
-----------------------------------------
Total Power          = 2.142e-03 (100.00%)
-----------------------------------------
```

Figura 2.9: Ejemplo de reporte de potencia.

En el contexto del diseño de circuitos integrados se utilizan herramientas automatizadas para realizar análisis similares al realizado para el ejemplo de la figura 2.8 en las secciones anteriores. Sin embargo, en el caso de un *chip* real se deben sumar las potencias de millones de compuertas e interconexiones para obtener un análisis completo.

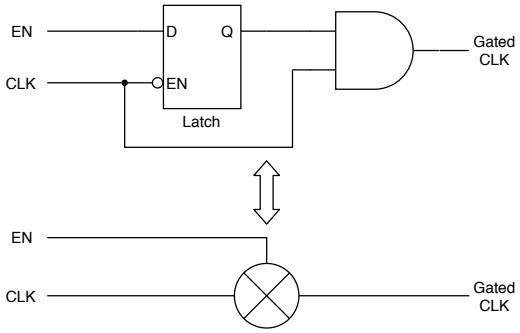
En la figura 2.9 se muestra un reporte de potencia de ejemplo presente en el manual de la herramienta PrimePower. En este se muestran los grupos de potencia discutidos en las secciones anteriores como columnas para las cuales se desglosa el consumo en distintos subgrupos del diseño, como red de interconexión de reloj, registros, lógica combinacional, lógica secuencial (que incluye *latches*, *clock gates*, etc.), memorias, *pads* de interconexión al exterior del *chip* y cajas negras que representan bloques *IP* para los que se desconoce la lógica interna y solo se cuenta con modelos de tiempo, potencia y *layout*, entre otros. [19]

2.5. Técnicas de reducción de potencia dinámica

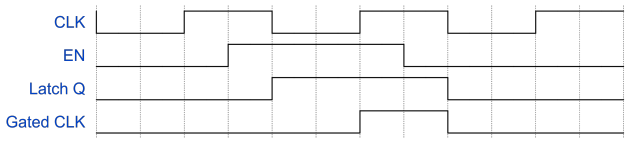
Existen diversas técnicas para reducir la potencia dinámica en un circuito VLSI. Estas se basan en reducir las componentes potencia de conmutación interna y potencia de carga de salida, ya sea por modificaciones estructurales que no afecten la lógica del circuito o mediante la alteración de parámetros físicos y de implementación del *chip* tomando en consideración el consumo de energía por concepto de conmutación. En esta sección se revisarán algunos de los métodos utilizados en la industria para conseguir este objetivo. [22]

2.5.1. Clock-gating

Se basa en cortar la propagación de la señal de reloj a registros que no están siendo utilizados en un determinado momento. De esta forma mientras el registro este desactivado se elimina el consumo de potencia en las interconexiones de reloj, que se denomina como *clock tree* debido a su topología. Las *clock gates* en un circuito pueden ser insertadas por el diseñador, por ejemplo cuando se utiliza una *clock gate* para controlar el reloj de un periférico cuya señal de control se desactiva en un modo de operación de bajo consumo, o bien pueden ser inferidas durante síntesis cuando se producen lazos de realimentación lógicos en los registros, es decir, cuando existen caminos combinacionales desde la salida (Q) a la entrada (D), como se ejemplifica en la figura 2.11. También existen otras técnicas para inferir *clock gates* como self y *sequential gating* las cuales serán discutidas en las siguientes secciones. Esta funcionalidad se implementa de forma discreta o integrada mediante celdas *Integrated Clock Gate* (ICG), el circuito y su funcionamiento están ilustrados en la figura 2.10. La implementación integrada tiene la ventaja de ser más robusta al ser menos propensa a producir *glitches*, dado que se logra un mejor control del *layout* y por lo tanto de los retardos.



(a) Circuito discreto y su equivalente integrado (ICG).



(b) Diagrama temporal.

Figura 2.10: *Clock gate*.

El ICG se comporta como una compuerta AND a la que se conecta la señal de reloj y una señal de *enable* sincronizada mediante un *latch* transparente para que pueda activarse solo mientras la señal de reloj es baja, de esta forma se asegura que no se produzcan flancos de subida falsos al activar el *enable*. Además, al desactivarse el *enable* el *latch* conserva la compuerta AND abierta hasta el siguiente flanco de bajada. De esta forma es posible asegurar que a la salida de la *clock gate* siempre se verán solo flancos de subida y bajada correspondientes con los del reloj y que aparecerán "instantáneamente" (en el mismo ciclo de reloj).

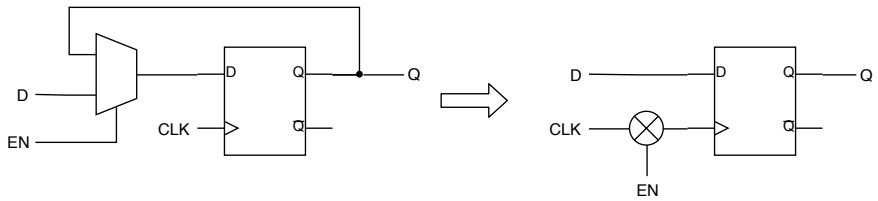


Figura 2.11: Ejemplo de uso de ICG.

El modo de uso de una ICG se demuestra en la figura 2.11, a la izquierda se tiene un circuito secuencial sintetizado sin *clock gating* y a la derecha con *clock gating*, en donde la ICG se utiliza para controlar el reloj en lugar de lógica combinacional que va desde la salida hacia la entrada del registro, en este caso un multiplexor. El ahorro energético se produce dado que se evita el consumo del *clock tree* mientras el registro esta desactivado, sin embargo, debe considerarse que la inserción de la ICG significa un costo en área y se tiene inevitablemente un consumo de potencia asociado a la celda misma, por lo tanto, al momento de la inserción debe evaluarse caso a caso si se produce un ahorro de energía efectivo. También debe considerarse el cumplimiento de restricciones temporales al insertar *clock gates* puesto que la señal de *enable*, por lo general, proviene de un registro cuyo reloj tiene distinta latencia al reloj que entra a la *clock gate*, debido a los retardos de propagación en el *clock tree*. Luego dado que este reloj acciona un *latch*, como se ve en la figura 2.10, deben cumplirse la condición de máximo retardo para el *latch*. Finalmente se debe optimizar la topología del *clock tree* a modo de cumplir con las restricciones de tiempo impuestas al diseño. [4] [11]

2.5.1.1. Self-gating

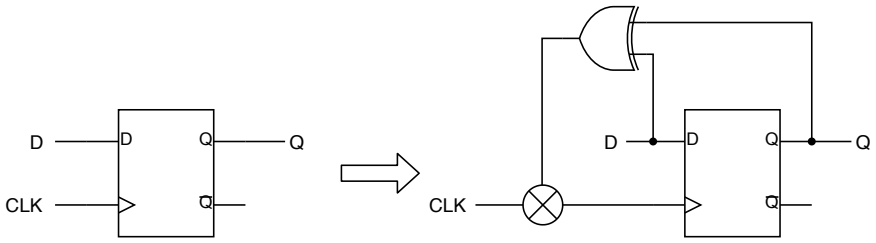


Figura 2.12: Ejemplo de uso de *self gating*.

Esta técnica corresponde a una variación de *clock gating* en la que se crea una señal de *enable* a partir de una compuerta XOR conectada a la entrada y a la salida de un registro, de esta forma la *clock gate* se activa solo cuando el dato en la entrada cambia y se desactiva mientras es constante. Se debe destacar que esta técnica solo es efectiva para señales de entrada de registros cuya *toggle rate* es bajo con relación a la frecuencia del reloj. Además, por lo general cuando se tienen bancos de registros de múltiples *bits* de ancho se utiliza la misma *self gate* con su señal de *enable* derivada de un árbol de compuertas XOR y OR para controlar el banco, sin embargo, si el número de *bits* es muy grande la efectividad de la *self gate* se pierde pues aumentan las probabilidades de que el *enable* este activo. Al igual que con la inserción de *clock gating* tradicional las herramientas deben evaluar cada registro antes de insertar las celdas pues de lo contrario se podría lograr un bajo impacto en términos de potencia con altos costos en área y congestión del *chip*.

Tabla 2.3: Compuertas de detección para *self gating*.

D	**Q**	**XOR**	**OR**	**NAND**
0	0	0	0	1
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

Para lograr una mayor efectividad y cobertura de *self gating* se pueden utilizar compuertas OR y NAND en lugar de XOR para detectar los cambios en el registro, esto es útil debido a que la última es la compuerta primitiva que requiere el mayor número de transistores para implementarse, por lo que presenta un mayor consumo y área. Como se observa en la tabla 2.3, la compuerta OR se desempeña de igual forma que XOR si la señal D es 0 la mayor parte del tiempo, es decir, tiene un *static probability* cercano a 0, sin embargo, el reloj quedará activado cuando D sea 1 por más de un ciclo. Aplicando el mismo razonamiento se concluye que para la compuerta NAND se requiere que la señal D tenga un *static probability* cercano a 1. Finalmente, se debe destacar que para aplicar exitosamente esta técnica es importante contar con datos de *switching activity* obtenidos de una simulación representativa del funcionamiento real del diseño.

2.5.1.2. Sequential-gating

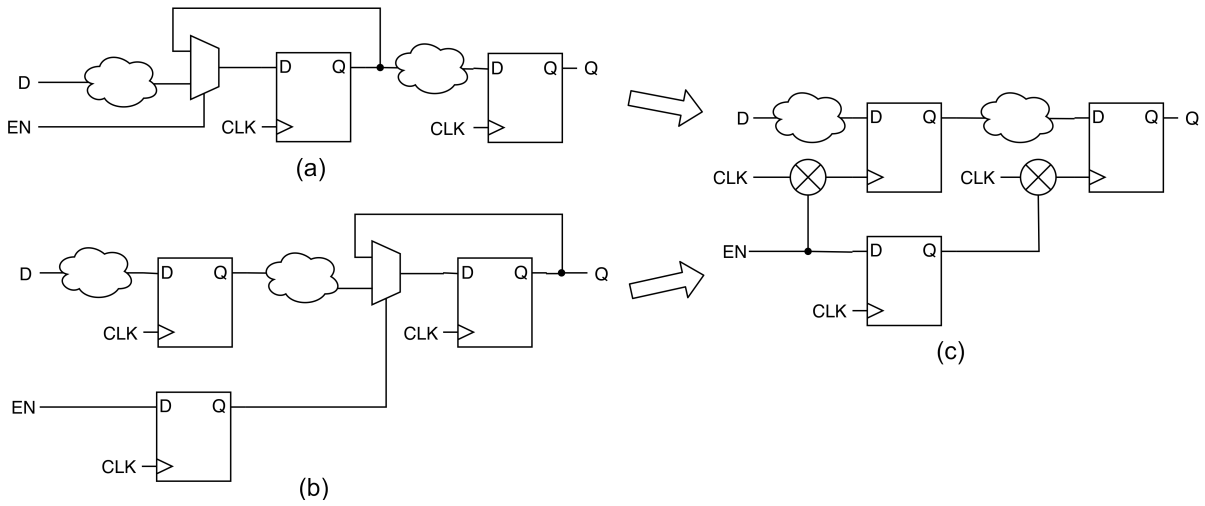


Figura 2.13: Ejemplos de uso de *sequential gating*.

Con este método se busca aprovechar condiciones de inactividad que provienen de etapas anteriores o posteriores en el *pipeline* tal como aprecia en la figura 2.13. En el caso (a) el *enable* del primer registro afecta la actividad del segundo pues, para que su valor cambie, es necesario que el primero lo haga en el ciclo anterior, es por esto que en la versión del circuito sintetizada con *sequential gating* se inserta un registro extra para obtener una señal de *enable* retrasada en un ciclo de reloj la que utiliza para controlar la ICG del segundo registro, mientras que el primero es sujeto a una ICG controlada por la señal de *enable* en el ciclo actual. En el caso (b) el segundo registro es controlado por una señal de *enable* y los cambios en el valor del primer registro mientras el segundo esta desactivado no cumplen ninguna utilidad y constituyen un gasto de potencia innecesario, por lo tanto, nuevamente se utiliza la señal de *enable* y su versión retrasada para controlar las ICGs.

Al igual que las anteriores esta técnica está sujeta a una evaluación cuidadosa de su efectividad en cada registro, y en mayor medida dado que en algunos casos implica insertar un registro adicional. Es esperable que este tipo de técnicas de reducción de potencia dinámica que requieren la inserción de múltiples celdas sean efectivas en una menor cantidad de esce-

narios pues los consumos adicionales diluyen el ahorro neto y tienen efectos colaterales como el aumento de área utilizada, mayor congestión durante el proceso de enrutamiento y pueden incrementar la complejidad de la verificación funcional del *chip* al manipular la lógica.

2.5.2. Posicionamiento de baja potencia

Dado que las interconexiones en un circuito tienen asociadas capacitancias y resistencias parásitas se puede ahorrar energía al acortar las distancias entre compuertas y es justamente esto lo que se busca con esta técnica. A través de simulación se anota la actividad en cada nodo del circuito y luego en la etapa de *placement*, donde se ubican físicamente las celdas, se intenta acortar los caminos de mayor actividad tal como muestra la figura 2.14. Notar que se deben considerar las restricciones de tiempo al modificar los largos de las rutas, y por lo tanto también los retardos, de los caminos. [6]

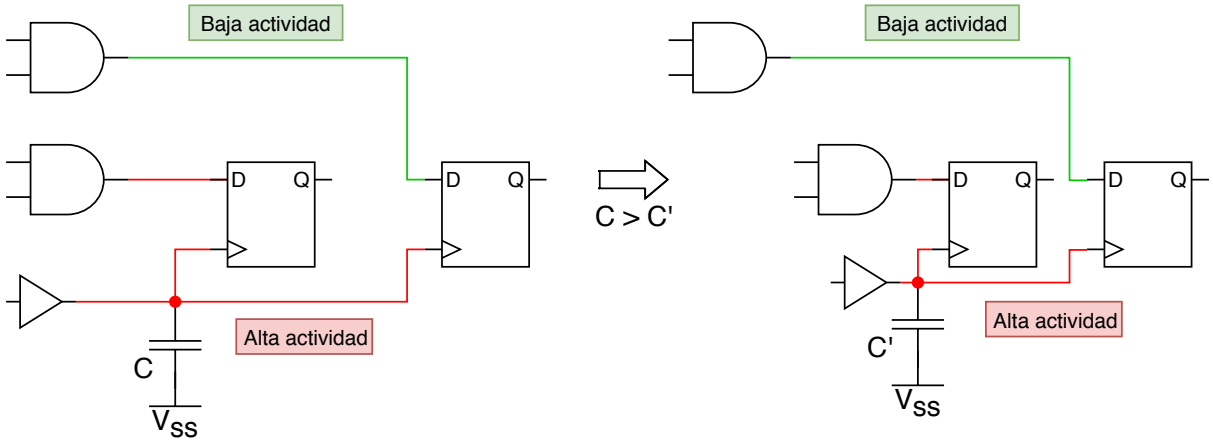


Figura 2.14: Ejemplo de posicionamiento de baja potencia.

2.5.3. Re-implementación de compuertas

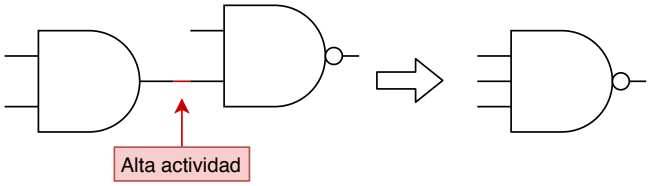


Figura 2.15: Ejemplo de uso de re-implementación de compuertas.

Basándose en los principios de la técnica anterior esta favorece las celdas complejas para acortar los caminos de alta actividad, colapsando múltiples compuertas en una sola cuando se tiene la celda apropiada y se cumplen las restricciones de tiempo, de forma que se obtiene una reducción de potencia dinámica. [4]

2.5.4. IPs de bajo consumo

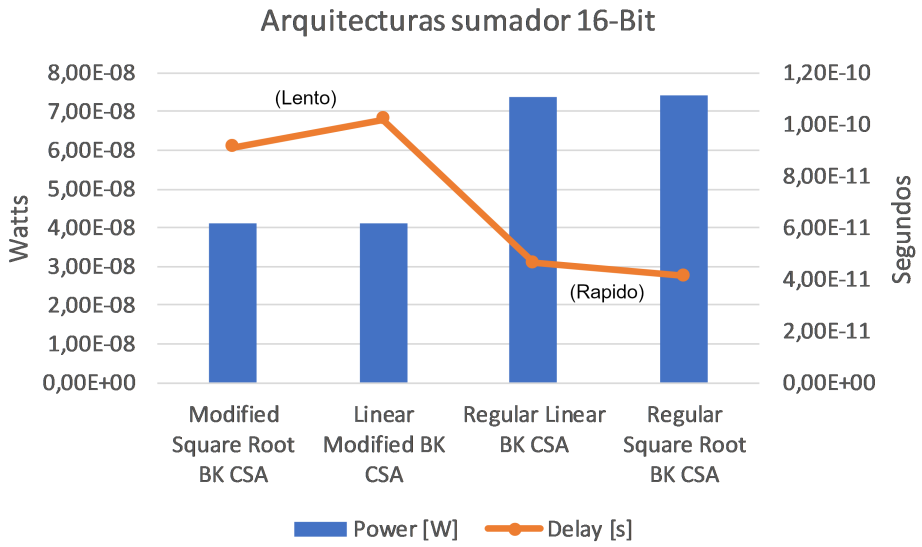


Figura 2.16: Comparación de distintas arquitecturas para un sumador de 16 *bits* (BK = Brent Kung, CSA = Carry Select Adder).

El uso de bloques IP o propiedad intelectual es una práctica ampliamente extendida en la industria de circuitos integrados, estos constituyen la base para construir sistemas en *chip* complejos como los que se utilizan cotidianamente en los procesadores de computadores y celulares. Por lo tanto, es natural que existan IPs diseñadas específicamente para bajo consumo, utilizando técnicas a nivel RTL el uso de arquitecturas que son más eficientes en términos de potencia y que se optimizan cuidadosamente para este objetivo.

Algunas de las formas en que un diseño puede optimizarse para bajo consumo es la reutilización de recursos, donde se prefiere disminuir el rendimiento con tal de disminuir la lógica y por lo tanto la potencia disipada. También es posible escribir código RTL poniendo énfasis en minimizar las transiciones en puntos de alto consumo como por ejemplo en *pads* de entrada/salida o en nodos que se sabe de antemano terminaran recorriendo largas distancias en la implementación del *chip* como buses e interconexiones varias.

A modo de ejemplo en la figura 2.16 se muestran distintas arquitecturas de sumadores con su respectivo consumo de potencia y retardo total [16]. De la figura se concluye que es posible reducir el consumo en aproximadamente 50 % tan solo escogiendo una arquitectura distinta, con el costo de un incremento en el retardo. Por lo tanto, estos sumadores de bajo consumo serán útiles en módulos de baja velocidad.

Las IPs de bajo consumo pueden ser instanciadas manualmente por el diseñador o bien pueden ser inferidas automáticamente por las herramientas de síntesis lógica cuando se realizan ciertas operaciones en el código RTL como sumas, multiplicaciones, conversión de paralelo/serial o serial/paralelo, etc.

2.5.5. Dimensionamiento de celdas

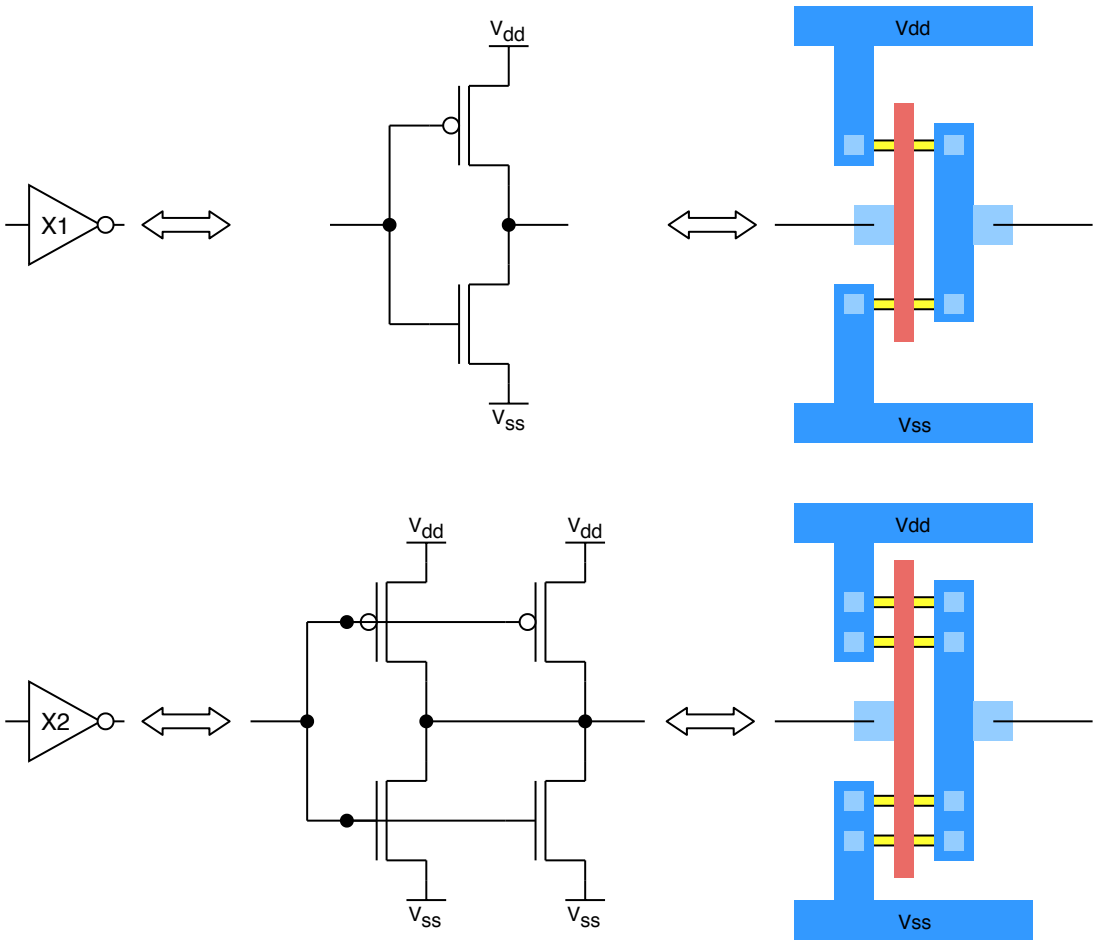


Figura 2.17: Celda NOT de tamaño X1 y X2.

Una biblioteca de celdas estándar normalmente contiene distintos tipos de compuertas lógicas de las cuales se tienen varios *layouts* para una misma compuerta según su capacidad de entregar corriente en las salidas, propiedad denominada como fuerza de conducción. Para incrementar esta capacidad se debe agrandar la celda físicamente, ya sea aumentando el ancho del canal o bien poniendo transistores en paralelo para disminuir la resistencia asociada a la salida. Con esto se logra aumentar la capacidad para entregar o recibir corriente por lo que generalmente en el nombre de las celdas se incluye el sufijo Xn donde n es el factor por el que se multiplica la fuerza de conducción relativa a la celda de menor tamaño, en la figura 2.17 puede verse un ejemplo para una compuerta NOT. [7]

En general las compuertas de mayor tamaño pueden conducir corrientes más grandes lo que se traduce en una mayor velocidad para cargar y descargar los transistores de su *fan out*, sin embargo, como se aprecia en los *layouts* de la figura 2.17 las compuertas grandes también tienen una mayor capacitancia de entrada al tener un canal más ancho o al tener varios canales en paralelo. Es por esto por lo que calcular el tamaño de cada celda en un

camino combinacional tal que el retardo total sea mínimo es un problema de optimización que debe ser resuelto considerando las capacitancias de entrada y salida de cada etapa. [11]

Además de las implicancias sobre la velocidad de las celdas, el tamaño de estas está relacionado directamente con el consumo, es decir, las celdas de mayor tamaño tienen un mayor consumo que las de menor tamaño dado que su capacitancia asociada es mayor, así como su capacidad de conducción corriente. Considerando el párrafo anterior, su uso no siempre se traduce en un incremento en velocidad y en general las celdas de gran tamaño son óptimas cuando tienen un gran *fan out* o bien se conectan a interconexiones extensas. Por lo tanto, es posible optimizar el tamaño de las celdas en forma posterior a la implementación inicial, en el que se considera el retardo como objetivo principal. Esta optimización posterior se conoce como recuperación de potencia o área, y busca reducir el tamaño de las celdas donde sea posible considerando las restricciones temporales. [4]

2.5.6. Técnicas de reducción de glitch

Como se vio en la Sección 2.3 la potencia dinámica consumida por *glitch* es absolutamente desperdiciada pues no cumple ningún propósito en términos funcionales, es por lo tanto de gran interés minimizar la cantidad de *glitch* en un circuito. En las secciones siguientes se presentarán algunas de las técnicas que se han estudiado para lograr este propósito.

2.5.6.1. Inserción de registros

La generación de *glitch* se produce cuando las señales de entrada de un circuito combinacional llegan con desfase y la función lógica produce transiciones inesperadas en la salida. La primera condición tiene una alta probabilidad de cumplirse, pues puede darse que las diferencias se deban a los largos de las interconexiones, cantidades de celdas en los caminos combinacionales o los distintos tamaños de celdas. Para la segunda condición, ciertos circuitos tienen lógica especialmente propensa a propagar conmutaciones como sumadores, multiplicadores, comparadores y en general cualquier circuito que contenga grandes cantidades de compuertas XOR.

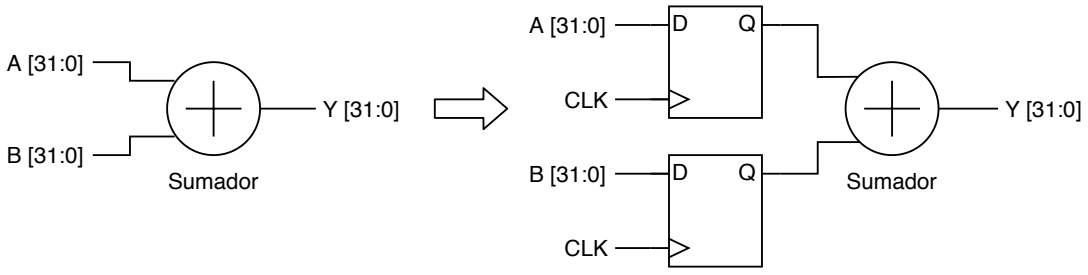


Figura 2.18: Ejemplo de inserción de registros.

En la figura 2.18 se muestra un sumador de 32 *bits* cuyas 64 entradas con alta probabilidad llegarán en desfase por lo que en la salida se producirá una alta cantidad de conmutaciones por *glitch* antes de detenerse en el resultado esperado. Una forma simple pero efectiva de suprimir

estos *glitch* es colocar una etapa de registros antes del sumador a modo de sincronizar las entradas localmente. Esto a costa de la penalización en latencia de un ciclo y el área requerida por los registros.

2.5.6.2. Balance de retardos

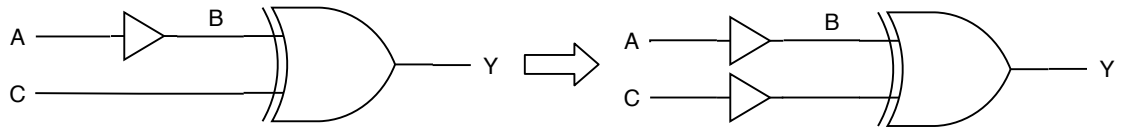


Figura 2.19: Ejemplo de balance de retardos.

Esta técnica consiste en ecualizar los retardos de cada camino combinacional hacia las entradas de un circuito, por ejemplo, la compuerta XOR de la figura 2.19, en la que se inserta un buffer de similares características al que ya estaba presente en la otra entrada para cumplir este propósito. También es posible cambiar el tamaño (fuerza de conducción) de las celdas para modificar los retardos, así como los largos de las rutas en el *layout* [8]. A diferencia de la técnica anterior esta no implica añadir nuevas etapas de registros por lo su aplicación es posible en una mayor cantidad de escenarios, aunque se debe notar que está sujeta al cumplimiento de restricciones temporales.

2.5.6.3. Lógica optimizada para reducción de glitch

Existen ciertas arquitecturas lógicas menos propensas a la generación y propagación de *glitch* en las que se evita la propagación de actividad cuando no es necesaria o bien están inherentemente diseñadas de forma que no se produzcan grandes diferencias en los retardos [14]. Por ejemplo el circuito comparador de la figura 2.20 a la izquierda generara una gran cantidad de *glitch* pues cualquier conmutación en las entradas se propaga hasta la salida, sin embargo el circuito optimizado para *glitch* de la derecha aprovecha el hecho de que el *bit* más significativo decide si la comparación es positiva en el 50 % de los casos por lo tanto la comparación de este *bit* se utiliza para inhibir la propagación de los *bits* menos significativos en la mitad de los casos posibles, y esta técnica se podría aplicar recursivamente para cubrir el 75 % de los casos con el siguiente *bit* más significativo y así sucesivamente. Además de este ejemplo hay otras arquitecturas para aritmética, decodificadores, entre otros, que son especialmente resistentes a *glitch* por diseño [15]. Muchas veces estos componentes pueden ser insertados en forma de bloques IP por la herramienta de síntesis de la misma forma en que se aprovechan las IPs especializadas para minimizar la propagación de actividad de la Subsección 2.5.4.

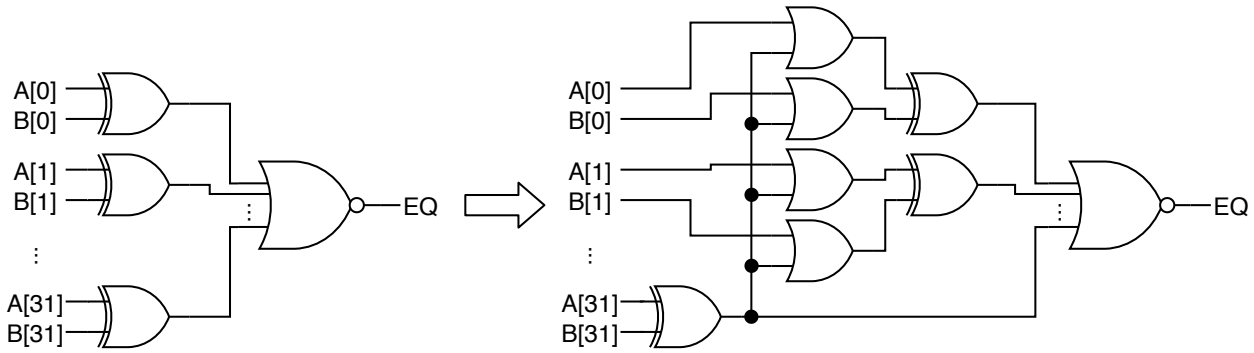


Figura 2.20: Ejemplo de lógica optimizada para reducción de *glitch*.

2.6. Figuras de merito

Para caracterizar un diseño se utilizan un conjunto de métricas con las que se puede estimar la calidad de la implementación de un *chip*, tanto para medir el progreso de las distintas etapas del flujo de diseño, como para evaluar el resultado final. Para esto generalmente se utilizan tres atributos del *chip* los cuales se describen a continuación.

Rendimiento Corresponde a la velocidad del circuito y por lo general se cuantifica con una holgura negativa de *setup* y *hold*, tanto del camino critico (WNS) como el total (TNS).

Potencia Describe el consumo energético del *chip*, el cual las herramientas de análisis pueden desglosar en potencia interna, conmutación y fuga, según se vio en secciones anteriores, y también se pueden subdividir en otras categorías como combinacional y secuencial, entre otras.

Área Se refiere al área total de silicio utilizada por el *chip*, así como al porcentaje de utilización o densidad de transistores e interconexiones implementadas.

Capítulo 3

Metodología

3.1. Discusión de requerimientos para la metodología de evaluación propuesta

Como se mencionó en la Sección 1.2, el objetivo de esta memoria es obtener una metodología para evaluar nuevas técnicas de reducción de potencia en diseños de circuitos integrados, las que fueron revisadas en la Sección 2.5. A partir de esta revisión se establece que es necesario contar con una buena estimación de la actividad del circuito para identificar correctamente las oportunidades de mejora, así como evaluar el cumplimiento de las restricciones de diseño. Por lo tanto, se requiere que la metodología permita evaluar la calidad de estimadores de actividad.

La metodología deberá permitir también realizar cambios a nivel de arquitectura, simulación (estímulos) e implementación lógica y física. Para esto se requerirá contar con todos los archivos fuente tanto a nivel RTL como la descripción en Verilog de la lógica, un marco de simulación como bancos de prueba y herramientas para generar estímulos, y los scripts del flujo de diseño completo desde síntesis lógica a implementación física del *chip*. Con esto será posible ajustar la arquitectura del diseño por ejemplo para hacer más efectiva la aplicación de alguna optimización. Desde el punto de vista del análisis de potencia que está muy relacionado con la actividad del *chip* durante su funcionamiento, será útil contar con la posibilidad de cambiar los estímulos para así ejercitar de forma más localizada los estimadores de actividad. Por último, tener disponibles las fuentes o scripts de implementación permitirá activar el uso de nuevas características relacionadas con análisis y optimización de potencia de la herramienta que se desean evaluar.

También se necesita tener capacidad de observación del consumo de potencia del diseño, lo que se traduce en el requerimiento de un mecanismo de medición capaz de observar el consumo tanto en un alto nivel como al detalle de cada nodo. Los resultados de las mediciones deben ser representativas de la implementación real en silicio y factibles de realizar con los recursos disponibles por lo que existe una limitación en tanto solo es posible emplear estimaciones mediante softwares especializados.

Se deberán generar flujos de extracción y análisis de datos automatizados para la generación de figuras de mérito y visualización de datos. Esto con el objetivo de posibilitar la

aplicación de la metodología a nuevos diseños y técnicas en forma eficiente. Además, se debe hacer énfasis en la exactitud y consistencia de los análisis, para asegurar que los resultados obtenidos sean fidedignos y lo más representativos posibles de la realidad, pues el objetivo final de este tipo de evaluación es dirigir el desarrollo de las herramientas de síntesis para que impacten positivamente el diseño de nuevos *chip*s.

Finalmente, un requerimiento importante es contar con un flujo de diseño que utilice las herramientas que se desean evaluar, que en este caso corresponden a las capacidades de síntesis lógica y física de Fusion Compiler. Por lo tanto, es conveniente un diseño de pruebas cuyo flujo de implementación haya sido desarrollado con herramientas de Synopsys, o bien que cuente con la documentación y archivos fuente suficientes para permitir crearlo o portarlo.

3.1.1. Requerimientos prácticos

En esta sección se resumirán los requerimientos concretos que se imponen a la metodología, empezando por una lista de hitos que serán abordados en los capítulos siguientes:

Elección de un diseño: Elegir un diseño de pruebas sobre el que se aplicaran las técnicas a evaluar, el cual debe contar con todos los archivos fuente indicados en la tabla 3.1.

Flujo de implementación: Es necesario generar este flujo en la herramienta Fusion Compiler.

Verificación formal: El diseño sintetizado debe cumplir verificación formal de equivalencia con respecto a su descripción RTL, el cual se realiza con la herramienta Formality de Synopsys. Además, la equivalencia debe ser reflejada por los resultados de simulación *Gate Level*.

Restricciones de tiempo: Dado que algunas de las técnicas a evaluar requieren de simulación *Full Timing Gate Level* es necesario que la implementación del circuito pase la verificación de tiempo o STA (*Static Timing Analysis*).

Verificación de consistencia: Los flujos de extracción y análisis de datos, así como la generación de visualizaciones de datos y figuras de mérito deberá contar con algún tipo de verificación de consistencia para asegurar la integridad de los resultados.

Tabla 3.1: Resumen de fuentes requeridas del diseño de prueba.

Fuente	**Formato**	**Descripción**
Diseño RTL	Verilog/UPF	Modelo lógico del diseño (Verilog sintetizable) y modelo del circuito de alimentación (UPF)
Bibliotecas de referencia	NDM/DB/Verilog	Modelos lógicos y físicos de las celdas estándar y macro celdas con sus modelos de simulación.
Estímulos	Assembly/C/script	Programa o vectores para simular el funcionamiento del diseño.
Banco de pruebas	Verilog	Lógica de soporte que permite estimular y verificar el diseño en un entorno de simulación.
Restricciones de diseño	TCL	Restricciones de velocidad y potencia que la implementación del diseño deberá cumplir.
Modelos de parásitos	TLU	Coeficientes RC para extraer el modelo de parásitos del circuito implementado.
Tech file	TF	Especificaciones de tecnología de fabricación para implementación física.
Floorplan	TCL/DEF	Restricciones para implementación física.

3.2. Elección del diseño de pruebas

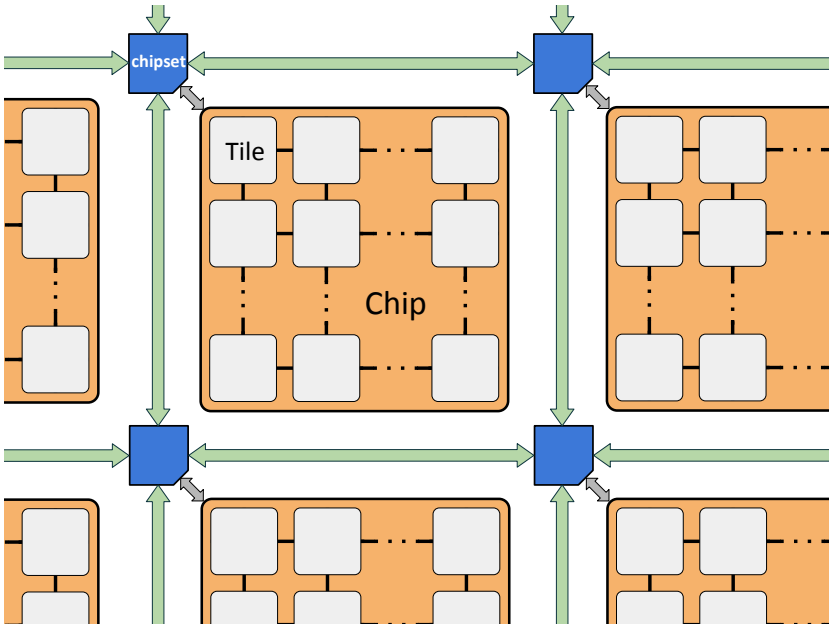


Figura 3.1: Arquitectura OpenPiton. *Chip*: Circuito integrado principal. *Chipset*: Integrado secundario para interconectar múltiples *chips*. *Tile*: Unidad de procesamiento básica.

El diseño de pruebas a utilizar será el procesador OpenPiton [2], creado para usarse como plataforma de investigación por la universidad de Princeton basándose en la CPU de código abierto OpenSPARC de Oracle. Este fue concebido para uso en data centers con una arquitectura optimizada para la ejecución de instrucciones sobre múltiples datos en paralelo (SIMD), además de permitir paralelismo masivo al interconectar varios *chips*, tal como se representa en la figura 3.1. Además, ha sido prototipado en FPGA y fabricado en ASIC, por lo que cuenta con toda la infraestructura necesaria para su implementación y verificación.

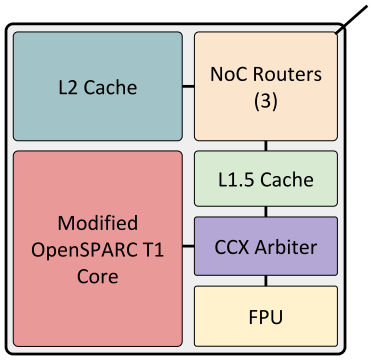


Figura 3.2: Tile de la arquitectura OpenPiton.

La arquitectura del *chip* está formada por múltiples *tiles* (núcleos) cuya configuración esta ilustrada por la figura 3.2. Estos contienen la CPU junto con memorias cache, unidad

de punto flotante (FPU) y un *router* para la interconexión o *Network-on-Chip* (NoC), que en este caso puede ser extendida hacia afuera del *chip* como muestra la figura 3.1.

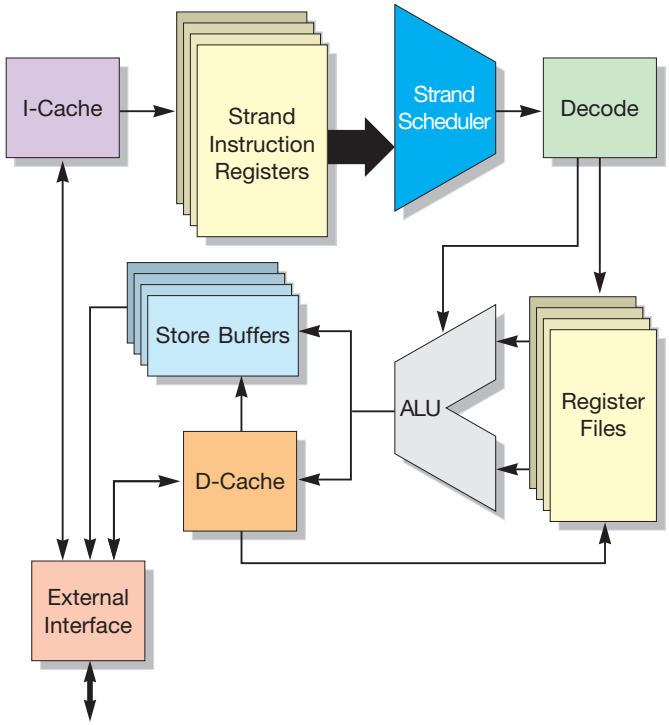


Figura 3.3: Arquitectura del *SPARC core*.

A modo de descripción general la CPU o *SPARC core* es de tipo RISC, soporta 4 hilos y tiene un *pipeline* de 6 etapas, su arquitectura se ilustra en la figura 3.3. En términos de software esta CPU tiene un excelente soporte debido a que ha sido utilizada ampliamente en la industria.

Este diseño se utilizará debido a su complejidad apropiada, que permite realizar diversos tipos de pruebas, y a su estructura jerárquica que permite abordarlo de forma parcelada. Además, en el proyecto original se utilizaron herramientas de Synopsys para la implementación y verificación del *chip*, contándose con lo siguiente:

Código fuente RTL: La base del proyecto es el código Verilog, que consiste principalmente en una versión modificada del OpenSPARC y un sistema de interconexión que permite parametrizar el número de núcleos en el *chip*.

Flujo de diseño: Como se dijo anteriormente, el *chip* fue fabricado con 25 núcleos utilizando un proceso IBM de 32nm. Para conseguir esta implementación se utilizaron las herramientas de Synopsys: Design Compiler para síntesis lógica y IC Compiler para implementación física. Se debe mencionar que el flujo de implementación publicado bajo licencia de código abierto está incompleto pues por motivos de propiedad intelectual no es posible publicar datos sobre las bibliotecas lógicas de IBM.

Infraestructura de verificación funcional: Este consta de infraestructura para correr test de regresión automatizados que en total contiene cerca de 8000 tests. A grandes

rasgos estas pruebas consisten en un programa en lenguaje ensamblador, que podría ser compilado, y una serie de monitores programados en Verilog para verificar que el procesador esté funcionando correctamente a lo largo de la ejecución del programa.

Dada la regularidad presente en la arquitectura del procesador y a su complejidad este debe ser implementado con una metodología *bottom-up*, es decir, se sintetizan primero los bloques que están más abajo en la jerarquía del diseño y luego estos bloques se van juntando para obtener el diseño completo. De lo contrario intentar implementar el procesador completo en un solo flujo de diseño significaría un costo de procesamiento muy elevado y es posible que no se converja a un resultado que cumpla con las restricciones de diseño. En esta memoria se trabajara con el módulo *SPARC core* a una frecuencia de reloj de $100MHz$ y un voltaje de alimentación de $0.85V$.

3.3. Flujo de implementación

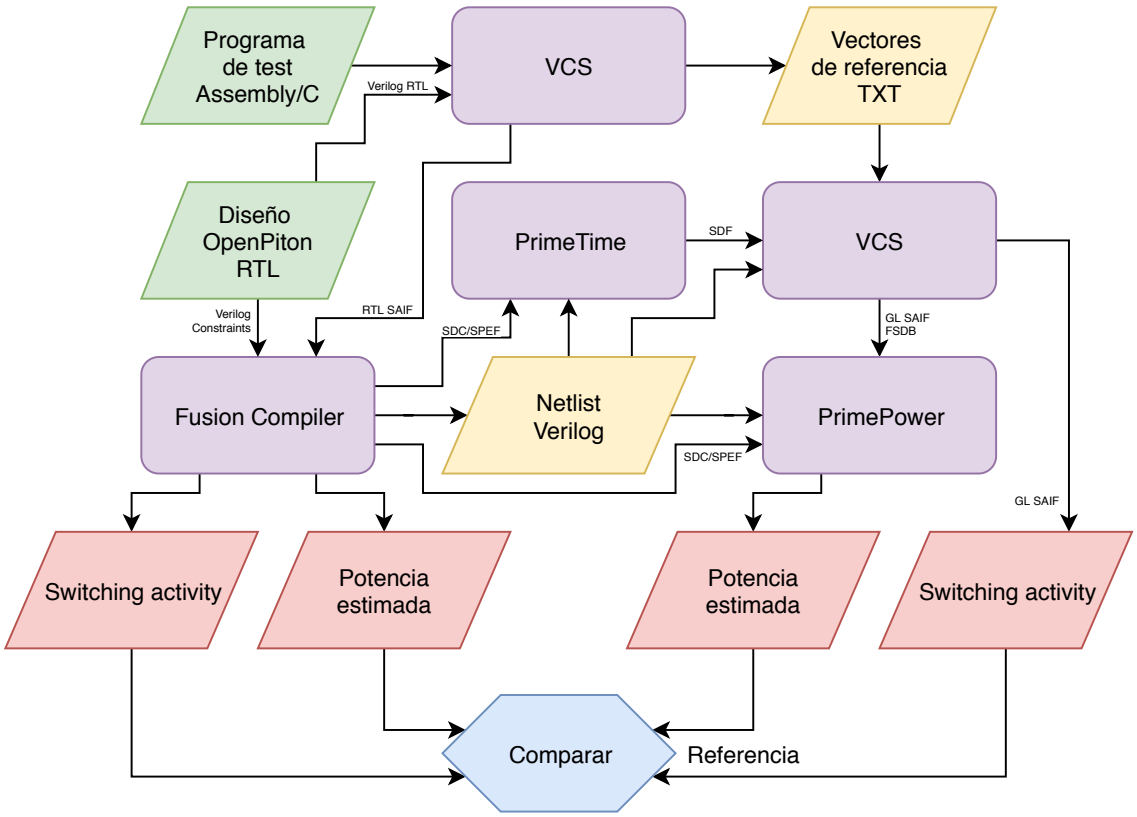


Figura 3.4: Flujo de implementación para la metodología propuesta.

Para cumplir con los requerimientos planteados se propone realizar la implementación de la metodología con el flujo de la figura 3.4, en este se resume la implementación del diseño en Fusion Compiler, las distintas etapas de simulación a nivel RTL y *Gate Level*, así como el análisis de potencia y la comparación entre herramientas que permite evaluar la efectividad de las distintas técnicas de reducción de potencia que se deseen evaluar. El objetivo de las

siguientes secciones del presente documento será desglosar este flujo en sus distintas etapas, para así explicar los requisitos y objetivos de cada una de ellas.

3.3.1. Simulación RTL

El primer paso que se realizara para obtener una implementación del diseño es realizar una simulación a nivel RTL en VCS para obtener el *switching activity* de los nodos invariantes que luego son anotados en un archivo SAIF (*Switching Activity Interchange Format*). Los nodos invariantes corresponden a nodos que están presentes en RTL y en la *netlist* sintetizada como los que se conectan a puertos y salidas de registros. Contar con un archivo SAIF RTL es importante para el flujo de diseño pues la herramienta de síntesis lo utilizará para optimizar el consumo de potencia aplicando las técnicas tratadas en la Sección 2.5. Se debe notar que el *switching activity* resultante dependerá tanto del diseño en sí mismo como del estímulo que se aplica a las entradas del circuito, que proviene del software de prueba que se ejecuta en el procesador. Para obtener un SAIF para la herramienta de síntesis se utilizará un código ensamblador del repositorio de pruebas que calcula la suma de los números del 0 al 9 y revisa que el resultado final sea el esperado (45).

Un segundo objetivo igual importante de simulación RTL es la generación de un archivo de vectores para los puertos de entrada y salida, esto consiste en grabar el estado (0, 1, X o Z) de cada puerto del procesador en cada ciclo de reloj que será requerido como estímulo y valores de referencia para verificación en simulación *Gate Level*. Para realizar esta anotación se utiliza el código Verilog "playback_dump.v" obtenido del repositorio y ligeramente modificado pues existen discrepancias en las rutas de los puertos.

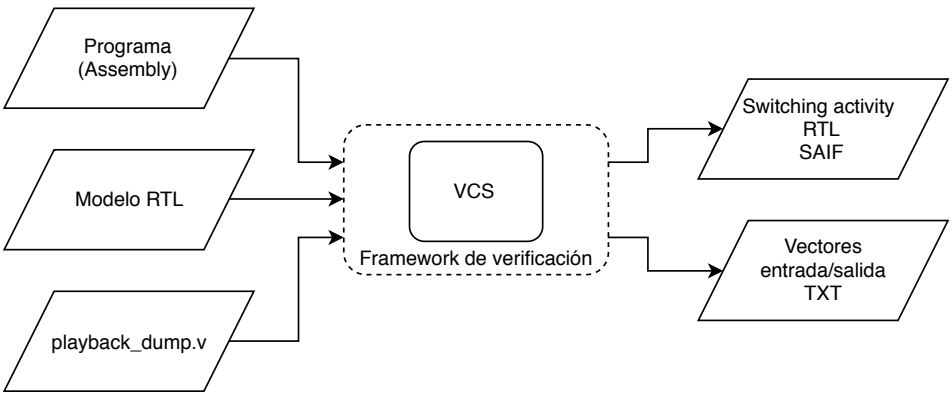


Figura 3.5: Flujo de simulación RTL.

Para llevar a cabo la simulación RTL se usará el script de verificación del proyecto OpenPiton. El funcionamiento de este sigue el esquema de VCS de dos etapas, primero se compila el modelo de simulación binario que luego se puede ejecutar. Para compilar el modelo se requiere especificar la configuración del procesador (cantidad de filas y columnas de núcleos) y posteriormente se puede ejecutar especificando un código fuente en ensamblador.

3.3.2. Síntesis del diseño

Para implementar el diseño se utilizará Fusion Compiler, siguiendo la metodología explicada en el Capítulo 2. Como punto de partida se emplea la metodología de referencia del software (o RM, *Reference Methodology*), que incluye todos los scripts necesarios para implementar un *chip* a partir de un diseño RTL y sus restricciones, archivos de simulación, etc. Para esto es necesario modificar los scripts para cargar los archivos del proyecto, así como para la generación de los datos requeridos por la metodología de evaluación. El flujo de implementación esta resumido en la figura 3.7, a continuación, se revisarán algunos elementos adicionales requeridos para realizarlo.

Como se mencionó anteriormente el proyecto OpenPiton utilizaba las bibliotecas lógicas de IBM pero estas no pueden distribuirse libremente por lo que no se cuenta el diseño del *chip* implementado ni con macro-celdas compiladas, como bloques de memoria RAM. Para llevar a cabo la implementación se emplearán las bibliotecas lógicas DesignWare de Synopsys y se deberá compilar las RAMs requeridas para las memorias cache y archivo de registros de la CPU.

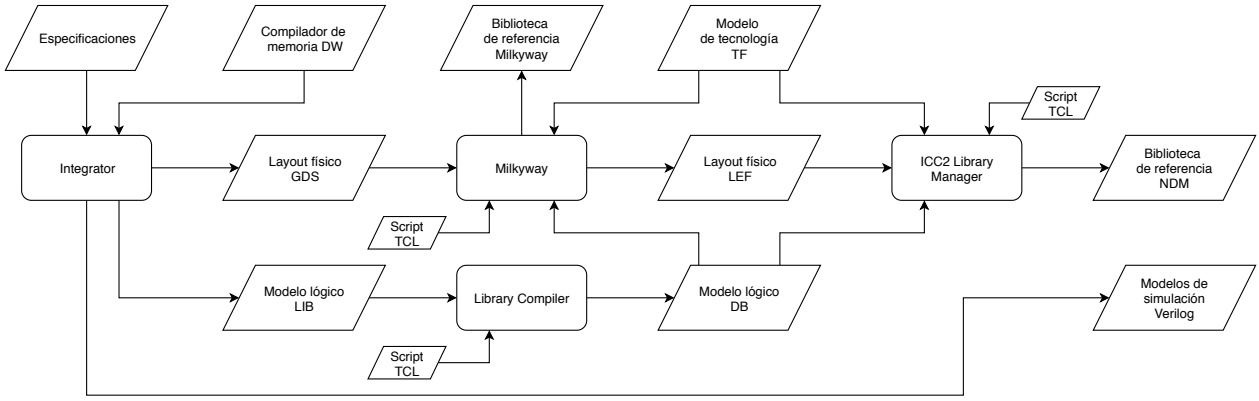


Figura 3.6: Flujo de compilación de memorias RAM.

Para compilar estas memorias se hace uso de los compiladores DesignWare, siguiendo el flujo de la figura 3.6 donde se ingresan las especificaciones de la tabla 3.2 como el tamaño de la memoria en palabras (Words), cantidad de *bits* por palabra (Bits), numero de puertos de escritura/lectura (Puertos) y otras capacidades como Built-In Self-Test (BIST) que consiste en incluir un circuito de test dentro de la memoria entregándole la capacidad de testearse a sí misma, y finalmente un puerto de mascara de escritura (Write mask) que es del mismo ancho que el puerto de escritura y permite escribir ciertos *bits* selectivamente, manteniendo los valores de los *bits* que no son escritos.

Tabla 3.2: Especificaciones para las memorias RAM

Referencia	**Words**	**Bits**	**Puertos**	**BIST**	**Write mask**
sram_1rw_128x78	128	78	1	Sí	Sí
sram_l1d_data_piton	128	288	1	Sí	Sí
sram_l1d_data_piton	128	288	1	Sí	Sí
sram_l1d_tag	128	132	1	Sí	Sí
sram_l1i_data	256	272	1	Sí	Sí
sram_l1i_data	256	272	1	Sí	Sí
sram_l1i_tag	128	132	1	Sí	Sí

Una vez compiladas las memorias estas se deben instanciar en los módulos entregados en la primera columna de la tabla 3.2. Estos son *wrappers* para las macro celdas, en donde dependiendo del caso se añade lógica para adaptar el comportamiento a lo esperado por el procesador. En este caso se debe negar la señal de *reset* y añadir un circuito tipo contador para escribir los valores en memoria a cero al inicializar el procesador, estos elementos son requeridos para pasar las pruebas de verificación. Para facilitar la escritura de lógica adicional se utiliza un banco de pruebas sencillo que instancia el *wrapper* de una memoria dos veces, en versión RTL y macro celda, y se realizan operaciones básicas como *reset*, escritura y lectura comparándose las salidas para verificar que el comportamiento sea exactamente el mismo.

Una vez que se completan los preparativos se cargaran los datos en Fusion Compiler siguiendo el esquema de la figura 3.7. Las entradas están divididas en tres categorías principales, estas son bibliotecas de referencia, el diseño mismo y modelo de tecnología. [18]

La primera corresponde a los modelos de la biblioteca de celdas estándar con las que se implementaran las compuertas lógicas en el diseño, así como a las macro celdas que están compiladas a partir de celdas estándar, pero se tratan como si fueran una sola unidad. Estas bibliotecas están contenidas en un archivo con formato NDM y empaquetan múltiples parámetros como modelos de lógica, tiempo y consumo de potencia para cierto rango de condiciones de operación y su geometría física, lo que incluye posicionamiento y enrutamiento en el caso de las macro celdas.

La segunda contempla importar el diseño se entregan las descripciones RTL del procesador, el archivo SAIF obtenido a partir de simulación RTL como se describe en la Subsección 3.3.1, un modelo del circuito de alimentación en formato UPF que en este caso es bastante simple pues se cuenta con un solo voltaje para alimentar el procesador completo y finalmente las restricciones del diseño en un script en TCL estas incluyen restricciones de tiempo como velocidad del reloj, retardos en los puertos de entrada y salida, excepciones y punto de operación para las celdas entre otros, finalmente se entrega el *floorplan* que incluye las ubicaciones de puertos y macro celdas, y geometrías de los rieles de alimentación para las celdas.

Y la tercera categoría corresponde a los parámetros de la tecnología de fabricación, esto se hace mediante un archivo con formato TLU+ que contiene los coeficientes RC que necesita la herramienta para realizar extracción de parásitos y el *tech file* que contiene las definiciones de las capas del *chip* que utilizaran las herramientas de implementación.

Luego las salidas del flujo son la *netlist*, que representa el circuito implementado en formato Verilog estructural. También se genera un modelo de parásitos en formato SPEF, en este se anotan la capacitancia y resistencia de cada una de las interconexiones presentes en la *netlist*, con esto se pueden calcular los retardos de cada una de ellas. Además, se escribe un archivo SDC (*Synopsys Design Constraints*) con las restricciones de tiempo escritas por la herramienta, esto es útil para facilitar el traspaso del diseño a otros softwares subsecuentes.

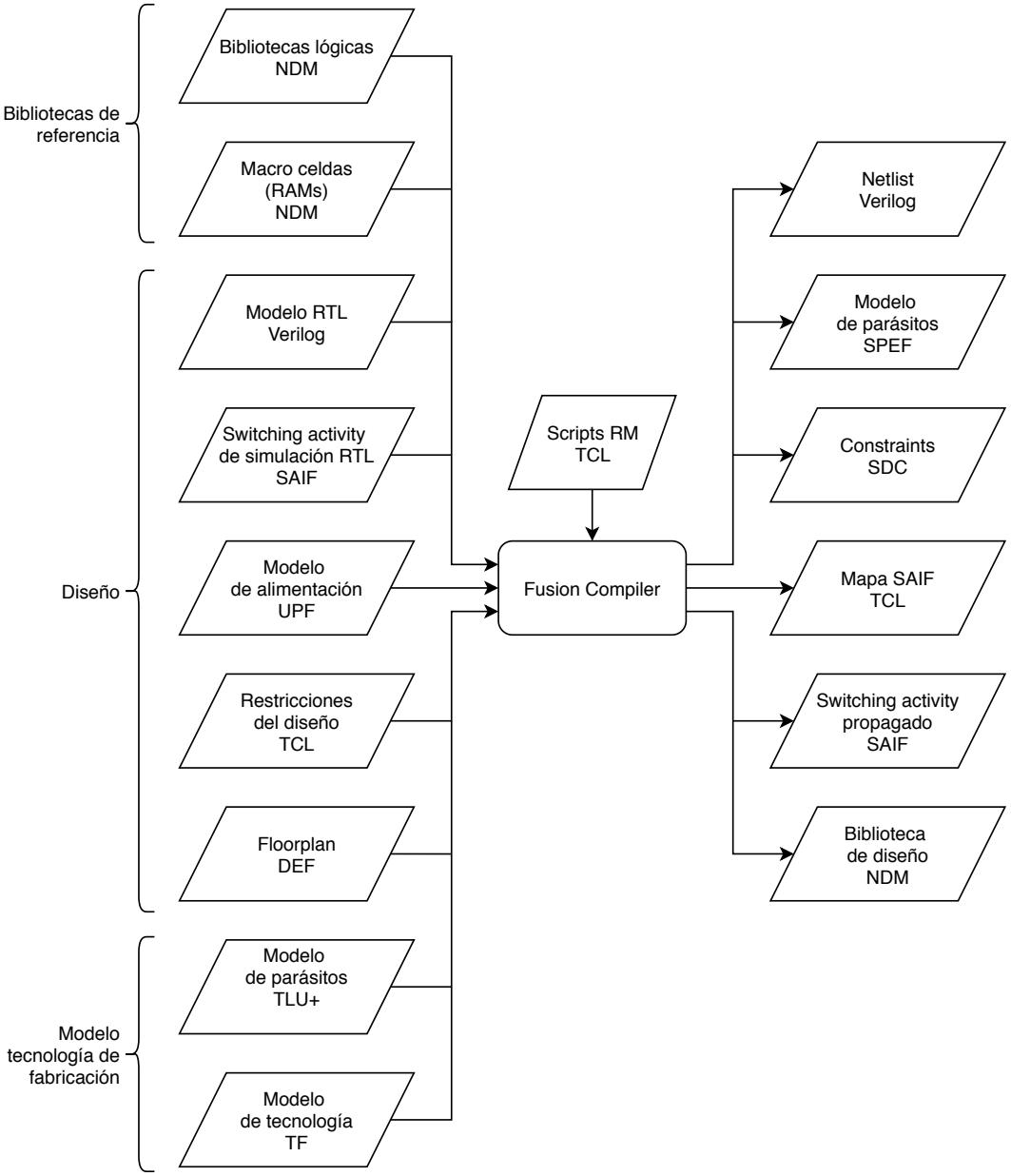


Figura 3.7: Flujo de implementación.

Como parte del análisis requerido para aplicar técnicas de optimización de potencia dinámica la herramienta debe calcular el *switching activity* en el circuito sintetizado reiteradas veces durante el proceso de implementación, esto debido a que se aplican varias transformaciones al mismo. Para realizar esta función se utiliza un motor de propagación que es capaz de estimar el *switching activity* a partir del archivo SAIF de simulación RTL, si bien es claro

que en la *netlist* aparecerán nuevos nodos estos se pueden calcular mediante simulación con valores aleatorios que simulen el *switching activity* anotado en nets que estén presentes en ambas versiones del circuito. Por lo anterior se escribe un SAIF propagado que corresponde a la actividad de la *netlist* estimada y un mapa en formato TCL de correspondencia entre nombres de nodos RTL y *netlist*.

Finalmente se escribe uno o más NDMs del diseño que contienen tanto la *netlist* como información de *switching activity*, parásitos y en general toda la información del diseño. Estos archivos son muy importantes pues se escriben en las distintas etapas del flujo y permiten luego retroceder la herramienta a un estado similar a cuando termina cierta etapa lo que se utiliza para inspeccionar resultados, extraer datos etc.

3.3.3. Cálculo de retardos para simulación *Gate Level*

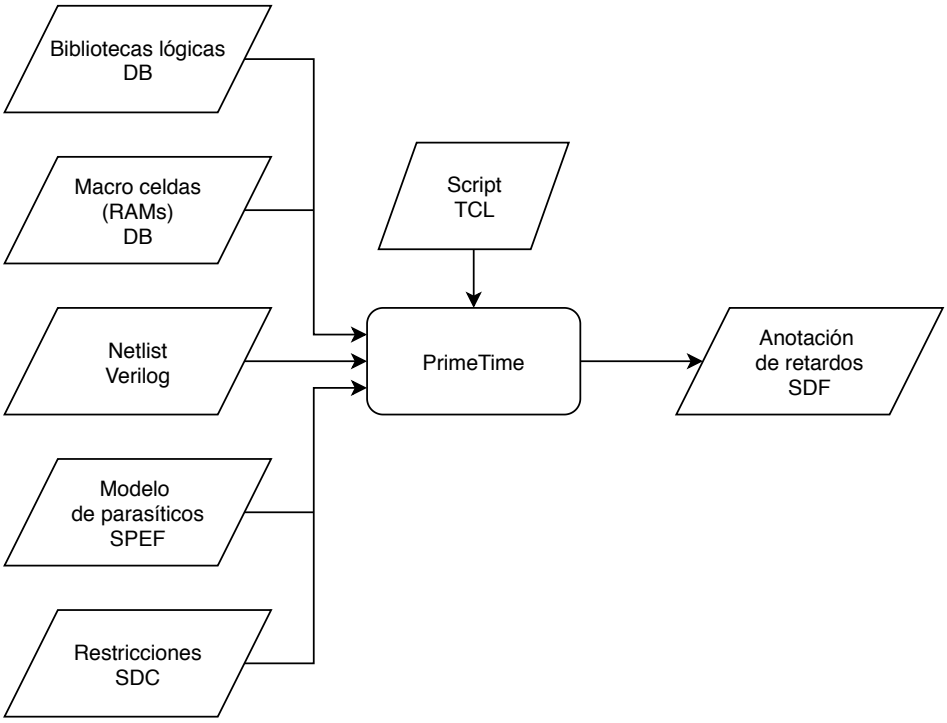


Figura 3.8: Flujo de cálculo de retardos.

Este paso será requisito para realizar simulación de tipo *Full Timing Gate Level*, y se debe a que los retardos tienen una fuerte dependencia con la implementación del circuito, así como de las condiciones de operación, por lo que deben ser anotados en forma posterior a la implementación. Para esto, en los modelos Verilog de las celdas se declaran los arcos de tiempo combinacionales (retardos dentro de la compuerta) y comprobaciones de tiempo secuenciales (umbrales de *setup* y *hold*) con valores nulos por defecto. Luego al momento de realizar la simulación se anotan los valores correctos a partir de un archivo SDF (*Standard Delay Format*). Ademas se incluyen los retardos de las interconexiones (rutas) del circuito.

Para generar un SDF se utilizan herramientas de análisis de tiempo que generalmente son utilizadas para el proceso de *sign-off* del *chip*, es decir en etapas finales del flujo de diseño cuando se requiere comprobar con una alta precisión y confiabilidad si el circuito cumple las restricciones antes de fabricarlo. Para esto se utilizará la herramienta PrimeTime, en la que se cargara el diseño y se ejecutara un análisis de tiempo estático y una vez que se verifique que no hay violaciones de tiempo se escribirán los datos en un archivo SDF para simulación posterior.

El proceso mostrado en la figura 3.8 consiste en cargar las bibliotecas de referencia en formato DB que contiene solo los modelos lógicos y de tiempo/potencia de las celdas. Luego se importa la *netlist*, parásitos y restricciones del diseño sintetizado obtenido en la Subsección 3.3.2 para proceder con el análisis de tiempo o STA.

3.3.4. Simulación *Gate Level*

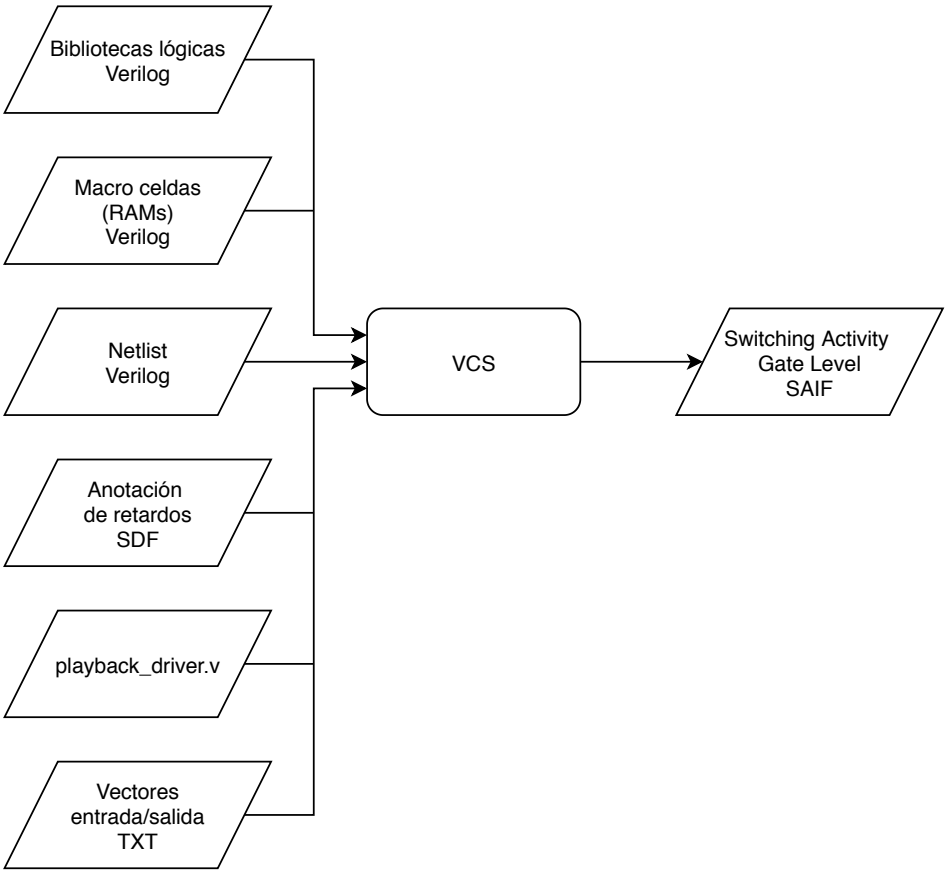


Figura 3.9: Flujo de simulación *Gate Level*.

Para obtener datos de *switching activity* con el mayor nivel de detalle y precisión posible se realizará una simulación en VCS de la *netlist* sintetizada cuya información de retardos es anotada a partir del análisis de tiempos realizado en la Subsección 3.3.3. Para esto se ejecuta el flujo de la figura 3.9 en donde se cargan en VCS los modelos Verilog de las celdas estándar y RAMs compiladas, así como la *netlist* obtenida de Fusion Compiler, el archivo

SDF de PrimeTime y finalmente el banco de pruebas contenido en "playback_driver.v" en donde se leen los vectores anotados en los puertos de entrada/salida del procesador que fueron obtenidos de simulación RTL, como se describe en la Subsección 3.3.1. Además, para esta simulación no se utilizará la infraestructura del proyecto OpenPiton debido a que no es posible utilizar los monitores originales pues los nombres de los nodos entre el código RTL y *netlist* son distintos. Es por esto por lo que para verificación se utilizará un sistema de marcador con vectores de referencia, el cual será descrito más adelante en este capítulo.

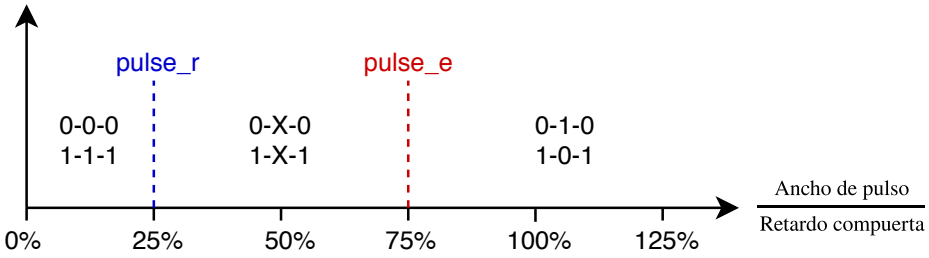


Figura 3.10: Demostración de los parámetros de control de pulsos en simulación con retardos.

Los parámetros más importantes para controlar el efecto de los retardos en simulación son los niveles de control de pulsos *pulse_e* y *pulse_r*, cuyo funcionamiento se ilustra en la figura 3.10. Estos corresponden a niveles umbral definidos como porcentaje del retardo de cada compuerta, por cada pulso que llega a las entradas de una compuerta la herramienta (VCS) evalúa si el ancho porcentual es menor al valor fijado en *pulse_e*, de ser así el pulso se propagará a la salida como un valor X, lo que corresponde a un *glitch* inercial cuando *pulse_e* se fija en 100 %. Luego si el ancho porcentual es menor a *pulse_r* el pulso es suprimido, es decir, no produce un cambio de estado en la salida, por lo tanto, al fijar este parámetro en 100 % se traduce en la eliminación de cualquier *glitch* inercial en la simulación. Un tercer parámetro es la opción delay_mode_zero para controlar la anotación de SDF, mediante la cual se puede desactivar completamente los retardos en la simulación [21].

Modificando los parámetros anteriores se repite la simulación para obtener cuatro variantes del archivo SAIF *Gate Level* cuya principal utilidad es en el cálculo de actividad debida a *glitch*, estas variantes se especifican en la tabla 3.3. Es importante destacar que en lo concerniente a los campos de *glitch* en el formato SAIF el simulador solo tiene la capacidad de anotar IG, es decir, cantidad de *glitches* inerciales y solo cuando los parámetros de control de pulsos son apropiados (ver tabla 3.3). El campo TG, para contar *glitches* de transporte, es problemático para el simulador pues las transiciones de un *glitch* de transporte son completas (0-1-0 o 1-0-1) por lo que no se pueden diferenciar de las funcionales.

Relativo al marcador en el banco de pruebas, el funcionamiento es el siguiente: en cada ciclo de reloj se lee una línea del archivo de vectores, dicha línea contiene dos números escritos en binario, el primero corresponde al valor de los puertos de entrada y el segundo a los valores de las salidas esperadas al final del ciclo. Con esto el banco de pruebas asigna los valores de las entradas correspondientes y al finalizar el ciclo verifica que las salidas sean correctas, si ocurre que uno o más *bits* de la salida es distinto al valor de referencia se incrementa un contador de discrepancias. Si el número de discrepancias excede cierto umbral la simulación

es abortada y se imprime un mensaje indicando que la prueba fallo. Si por el contrario la simulación funciona correctamente durante la cantidad de ciclos requeridos para llegar al final del archivo de vectores se imprime un mensaje de éxito y la prueba se da por pasada. Además, el banco de pruebas contiene los comandos Verilog necesarios para leer el archivo SDF, escribir un archivo VCD con formato propietario FSDB y escribir un archivo SAIF que contiene la información de *switching activity* del circuito, que es el objetivo de este paso en la metodología.

Tabla 3.3: Variantes de SAIF *Gate Level*.

Nombre de variante SAIF GL	**delay_mode_zero**	**pulse_r [%]**	**pulse_e [%]**	**Anotación de glitches**
zd	Sí	n/a	n/a	Actividad ideal anotada. No hay *glitches* en la simulación.
r0e0	No	0	0	*Glitches* inerciales anotados en TC. *Glitches* de transporte anotados en TC.
r0e100	No	0	100	*Glitches* inerciales anotados en IG. *Glitches* de transporte anotados en TC.
r100e100	No	100	100	*Glitches* inerciales suprimidos. *Glitches* de transporte anotados en TC.

3.3.5. Análisis de potencia

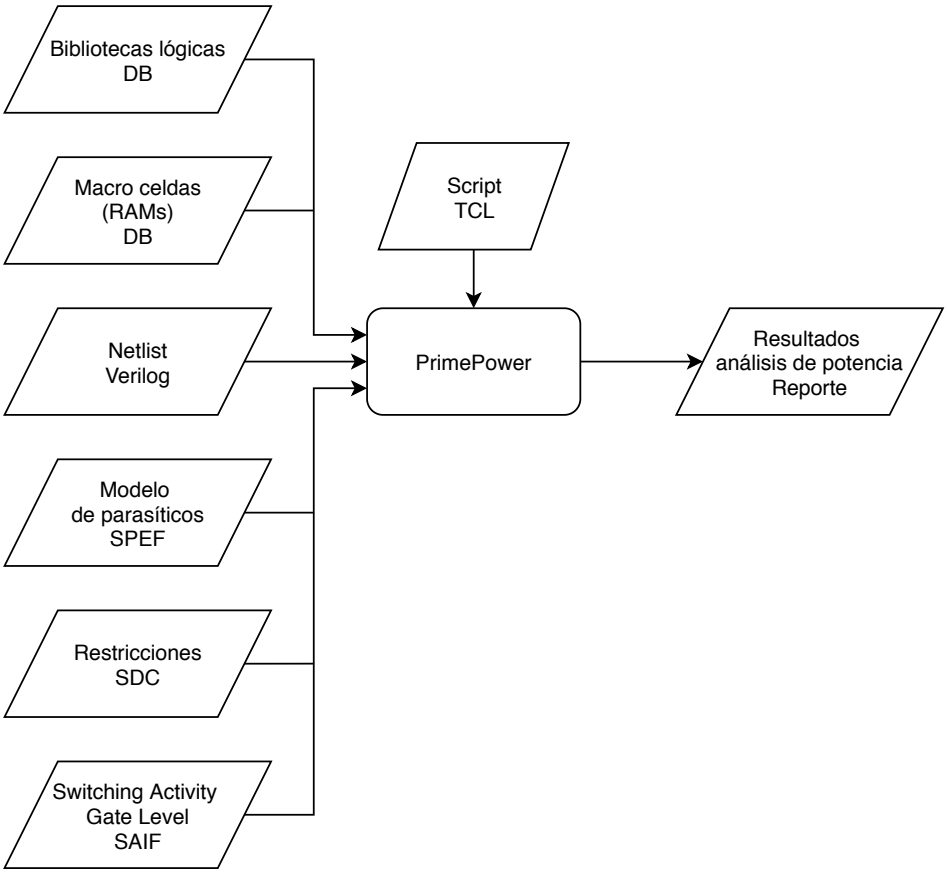


Figura 3.11: Flujo de análisis de potencia.

Ya que el objetivo final de este trabajo es evaluar la efectividad de un conjunto de técnicas de reducción de potencia dinámica es muy importante contar con una estimación precisa del consumo del diseño de pruebas, esto para tener una referencia sobre la cual realizar las evaluaciones, por ejemplo, para calcular la reducción de consumo al aplicar una optimización del circuito o evaluar la correlación de un nuevo estimador para análisis de potencia. Para obtener esta referencia se empleará la herramienta de *sign-off* PrimePower que es la de mayor precisión en cuanto a cálculo de potencia pues se utiliza para analizar el diseño al final del flujo, como parte de una batería de análisis para verificar el cumplimiento de todas las restricciones antes de fabricación.

Para esto se deberá importar el diseño incluyendo un archivo SAIF con el *switching activity* del circuito, esto es esencial para lograr una estimación de potencia representativa del funcionamiento real del circuito. Una vez abierto el diseño y ejecutado el análisis se obtendrá el reporte que contiene la información de potencia interna, de conmutación y fuga, descompuesta en las siguientes categorías: I/O *pads*, memoria, caja negra, red de reloj, registros, secuencial y combinacional. [19]

3.4. Extracción de datos

Toda metodología de reducción de potencia dinámica requiere de una buena estimación de la actividad del circuito, tanto para la ejecución de los algoritmos como para realizar la estimación de potencia, que corresponde a la función objetivo que se busca minimizar. Es por esto por lo que la extracción de datos se enfocara en estos dos elementos, *switching activity* y potencia, para un posterior análisis comparativo.

Para extraer información de actividad del circuito es necesario utilizar el formato SAIF de la IEEE, en particular para obtener los datos que se utilizarán como referencia de simulación como se vio en la Subsección 3.3.1 y Subsección 3.3.4, esto dado que la metodología de evaluación exige evaluar la precisión de los estimadores de actividad en la herramienta de síntesis. Para generar un archivo SAIF de simulación en el estándar [1] se definen dos flujos, representados en la figura 3.12 donde el ubicado en la sección superior genera datos SDPD (*State-Dependent Path-Dependent*), más detallados al utilizar un archivo *forward* SAIF que contiene modelos de celdas para entregar la información descompuesta en los distintos estados y caminos lógicos de cada celda. Dicha información es entregada en un archivo *backward* SAIF que consiste en una lista de nodos y celdas del circuito con sus respectivos datos de actividad anotados. Sin embargo, para este trabajo solo se empleará la forma de la sección inferior en donde solo se anota la actividad de cada nodo del circuito, esto debido a que utilizar *forward* SAIF genera complicaciones en el flujo al requerir la compilación de bibliotecas lógicas y en el análisis pues aumenta el volumen y cantidad de tipos de datos en el *backward* SAIF.

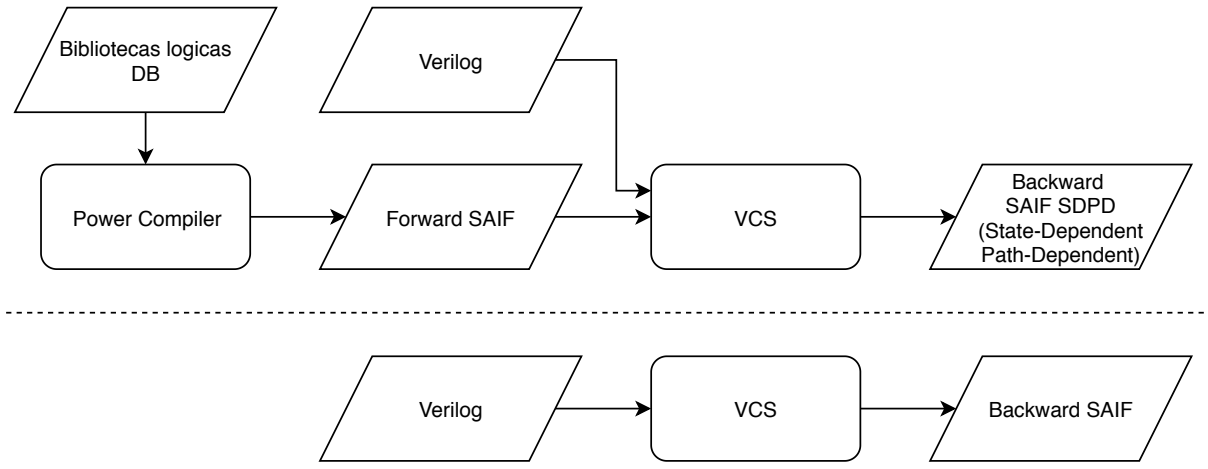


Figura 3.12: Flujos para generar SAIF según el estándar IEEE 1801.

Para realizar los análisis se utilizará Python con la biblioteca Pandas, que provee estructuras y algoritmos de análisis para manejar grandes cantidades de datos. Por lo tanto, se deberá transformar los archivos SAIF al formato DataFrame de Pandas, para esto se usará el flujo de la figura 3.13. Para analizar el archivo SAIF se hace uso de la biblioteca de C++ "cppSaif" de código abierto. Con esto se obtiene una estructura de datos tipo árbol, a partir de la cual se escribe un archivo en formato CSV (*Comma Separated Values*), donde la primera columna es la ruta jerárquica del nodo del diseño y el resto de las columnas corresponden a los datos presentes en nodos hoja del archivo SAIF (ver tabla 2.1). Con los datos en CSV un

script en Python puede fácilmente generar un DataFrame, además de leer los metadatos del archivo SAIF (duración de la simulación, escala de tiempo, entre otros) para luego escribir un archivo con formato Pickle, que es el estándar para guardar objetos de Python en disco duro. Posteriormente el archivo Pickle puede ser leído por otros scripts para procesar los datos y realizar análisis.

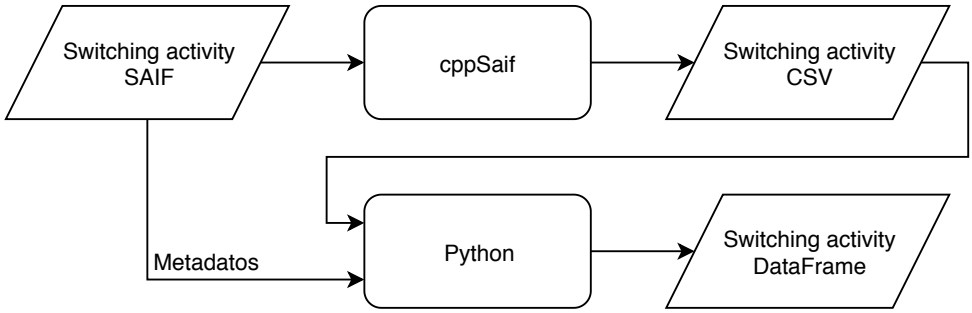


Figura 3.13: Flujo de lectura de archivos SAIF.

Por otro lado se deberá obtener los datos de *switching activity* de la herramienta de síntesis, en este caso Fusion Compiler, para esto se sigue el flujo de la figura 3.14 donde se abre la biblioteca de diseño resultante de cierto paso en el flujo de diseño como por ejemplo post-síntesis, post-posicionamiento, post-enrutamiento, etc. y luego se escriben en un archivo de texto plano una colección de listas que contienen los nombres las nets, así como los valores de los atributos de *switching activity*: *toggle rate*, *static probability* y *glitch rate*. Luego este archivo se analiza en Python para generar un DataFrame con las columnas respectivas. En general estos son los datos que se desea evaluar pues son generados por el motor de propagación de la herramienta y corresponden a una estimación de los valores entregados por el simulador.

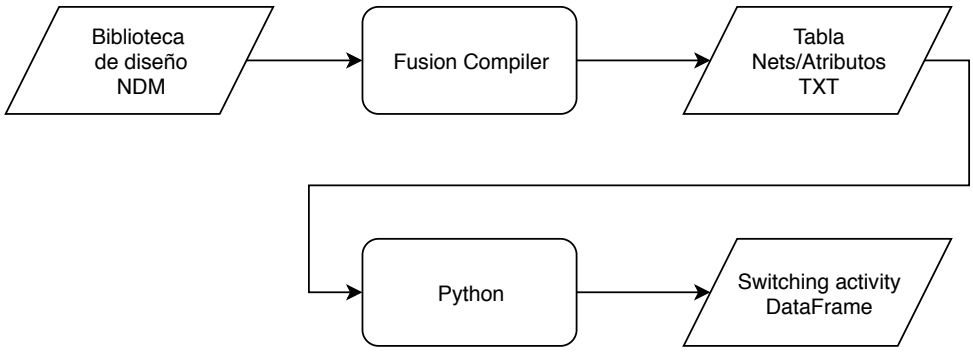


Figura 3.14: Flujo para obtener *switching activity* de Fusion Compiler.

3.5. Flujo de procesamiento de datos

Una vez se cuenta con los datos de actividad de referencia (SAIF) y los de la herramienta (Fusion Compiler o FC en lo que sigue) se debe realizar un preprocesamiento que consiste en calcular una intersección del conjunto de nombres de nets de FC con el conjunto obtenido del SAIF de simulación, esto debido a que la última contiene anotaciones redundantes y esto podría entorpecer el análisis posterior. La causa es mostrada en la figura 3.15 donde se ejemplifica un circuito con dos jerarquías en donde la misma conexión física queda representada por dos nets lógicas, una para cada jerarquía, y ambas son anotadas por VCS a pesar de no ser estrictamente necesario, y en el caso de nets que atraviesen más jerarquías se producen aún más anotaciones. Para resolver esta redundancia al momento de obtener la lista de nets del circuito en FC se utiliza el siguiente comando:

```
1   get_nets -hierarchical -top_net_of_hierarchical_group
```

Con el que se identifican los conjuntos de nodos equivalentes y se elige el que esta más alto en la jerarquía. En el caso de la figura 3.15 solo se tendría a "T/n2" en la lista de nets de FC, y al realizar la intersección la fila de "T/B/n1" seria eliminada del DataFrame SAIF.

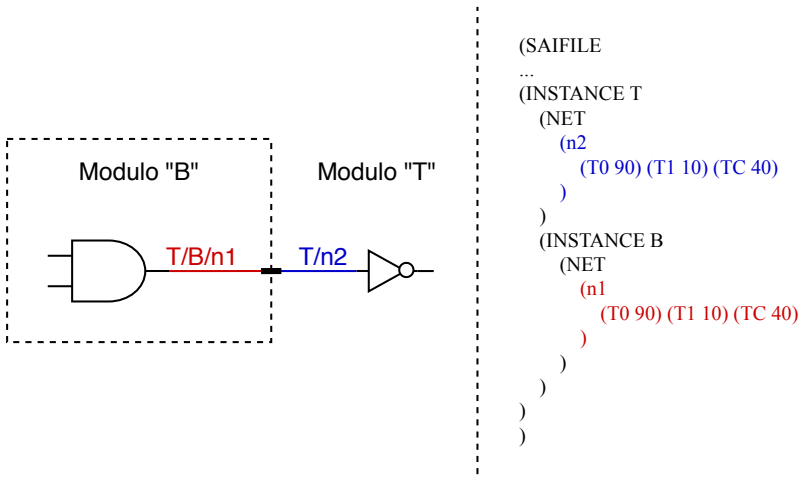


Figura 3.15: Ejemplo de anotación de un nodo que atraviesa múltiples jerarquías en SAIF.

Además, se deberán obtener los atributos de *switching activity* a partir de los datos crudos del archivo SAIF, es decir, transformar las columnas T0, T1, TC y TG en *toggle rate*, *static probability* y *glitch rate* según sea necesario.

Luego se calculan las siguientes ecuaciones para cada fila del DataFrame SAIF añadiéndose las nuevas columnas:

$$\text{Toggle rate} = \frac{\text{TC}}{T} \tag{3.1}$$

$$\text{Static probability} = \begin{cases} \frac{\text{T1}}{\text{T0+T1}} & \text{T0} > 0 \vee \text{T1} > 0 \\ 0.5 & \text{T0} = 0 \wedge \text{T1} = 0 \end{cases} \tag{3.2}$$

$$\text{Glitch rate} = \frac{\text{TG}}{T} \tag{3.3}$$

Donde T corresponde a la duración de la simulación, obtenida de los meta-datos del archivo SAIF y la definición por tramos del *static probability* se debe a que en simulación los registros sin *reset* tienen un valor X hasta que se carga un valor por primera vez, por lo tanto durante parte de la simulación o toda si no son utilizados, existirán nodos para los cuales $\text{T0} + \text{T1}$ no será igual al tiempo de simulación, para estos el cálculo contabilizará solo el tiempo que estuvieron en un estado válido. Y para los nodos que pasan toda la simulación en X se considera que 0.5 es el valor correcto para la probabilidad de estar en 1 pues el estado X representa una probabilidad igual de estar en 0 o 1.

Una vez que se obtienen los datos preprocesados se procederá a calcular el error de correlación entre la actividad de FC con respecto a la referencia que vendrían siendo los datos del SAIF de simulación de acuerdo con la ecuación 3.4.

$$\text{Error } [\,\%] = 100 \times \frac{|x - y|}{x} \tag{3.4}$$

Donde x corresponde al valor de un atributo calculado a partir del SAIF e y corresponde al valor de un atributo extraído de la herramienta (FC), este cálculo se realiza para cada fila (nodo del circuito) en ambos DataFrames. Luego se generan histogramas y se calculan estadísticas descriptivas para calificar la calidad de la correlación. Esta última etapa de obtención de resultados varía según el uso que se le dé a la metodología, pues dependiendo de la técnica de optimización o estimación que se esté evaluando se requerirán distintos tipos de análisis a partir de los datos.

3.6. Verificación de consistencia de datos

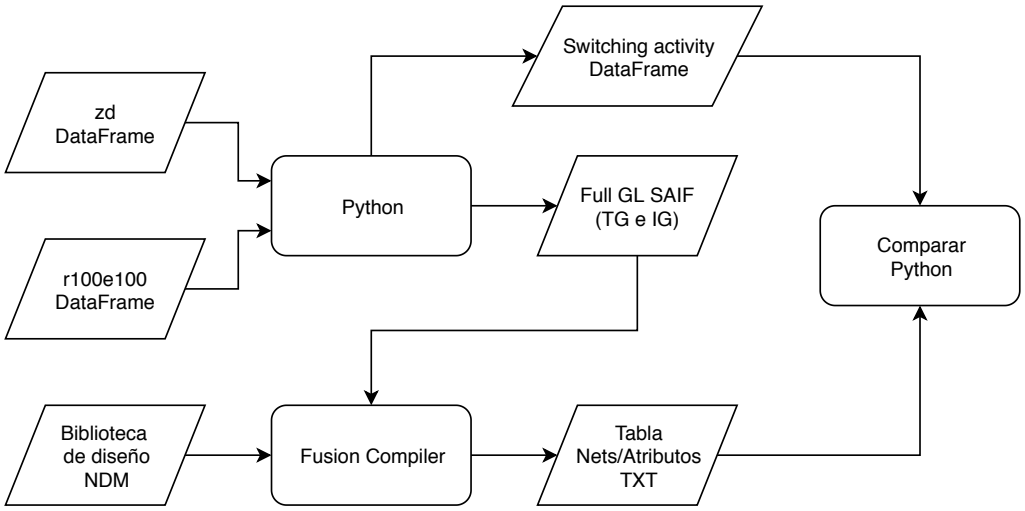


Figura 3.16: Flujo de verificación de consistencia.

Esta última etapa de la metodología cumple el objetivo de verificar la consistencia de los datos de *switching activity* al pasar por cada etapa del flujo de análisis. Específicamente se busca comprobar que las herramientas de Synopsys estén correctamente configuradas para la lectura y escritura de archivos contenedores como SAIF y tablas de atributos, y comprobar que los scripts de Python no contengan errores de código que invaliden los resultados. Para cumplir estos objetivos se emplea el método propuesto en la figura 3.16.

Este flujo se basa en utilizar los datos de *switching activity* de simulación *Full-Timing Gate-Level* para generar un archivo SAIF completamente anotado, es decir, que incluya todos los campos de actividad funcional, así como los de *glitch*, pues como se vio en la Subsección 3.3.4, el simulador no es capaz de anotar el campo TG para *glitches* de transporte. Para generar este campo se utilizan las cuentas de transiciones funcionales (TC) de las variantes *zero delay* y *r100e100* del SAIF *Gate Level* de VCS, esto pues como se ve en la tabla 3.3, el caso *r100e100* contiene los *glitches* de transporte sumados en TC por lo que basta con restar los TC del SAIF *zero delay*, que solo contiene transiciones funcionales, para obtener la cantidad de *glitches* de transporte en cada nodo. Lo anterior se calcula en Python y para generar un nuevo DataFrame que luego se escribe en formato SAIF.

Posteriormente en Fusion Compiler se abre la biblioteca de diseño para cargar la *netlist* sintetizada y se lee el archivo SAIF escrito anteriormente, lo que produce la anotación del circuito completo con todos los datos de *switching activity*. Con la *netlist* anotada se busca emular el caso de uso de la metodología, en donde se tendrían anotados los datos generados por el motor de propagación de la herramienta. Una primera verificación que hace en esta etapa es observar el reporte de anotación de *switching activity*, en el cual se espera una anotación del 100 % de los nodos proveniente de SAIF. Ya que en este caso los datos provienen directamente de VCS se pueden exportar los atributos de *switching activity* y luego compararlos con el DataFrame generado en Python. El resultado esperado es una correlación perfecta que debería reflejarse en cada gráfico y métrica del flujo de comparación.

Capítulo 4

Casos de estudio

En este capítulo se presentarán algunos casos de aplicación de la metodología propuesta en esta memoria. Estos casos corresponden a desarrollos realizados dentro de Synopsys para introducir nuevas capacidades a las herramientas de diseño de circuitos integrados con el objetivo en reducir el consumo de potencia dinámica a la vez que se minimiza el impacto negativo en las otras métricas de calidad de resultados como área y velocidad.

4.1. Resultados preliminares

En la tabla 4.1 se pueden encontrar las principales métricas de potencia, desempeño y área obtenidas para el *SPARC core* producido por el flujo de implementación, tal como se describe en la Sección 3.3. Para el análisis de potencia se utiliza la simulación del programa que calcula la suma de los números del 0 al 9.

Tabla 4.1: Características del *SPARC core* implementado.

Total Power:	6,1352E-03 W
Internal Power:	4,0230E-03 W
Switching Power:	2,0114E-03 W
Leakage Power:	1,0078E-04 W
Setup Slack:	0,2811 ps
Utilization Ratio:	47,96 %
Total Area:	434112,359 μm^2
Combinational Cell Count:	120509
Sequential Cell Count:	68018
Total Cell Count:	188527

La ejecución de los flujos se realizó en una máquina de 8 núcleos y 8 GB. de RAM. El flujo de implementación demora en promedio 1 hora y 30 minutos, mientras que las simulaciones aproximadamente 30 minutos.

4.2. Estimador de glitch

4.2.1. Descripción del caso

Se han registrado casos de diseños en los que el consumo energético debido a *glitch* es de un 70 % de la potencia dinámica y un 30 % de la potencia total [17]. Por lo tanto y considerando el estado de avance de optimización de potencia actual, es necesario considerar la reducción de *glitch* como una oportunidad para reducir el consumo de potencia.

La metodología propuesta en este trabajo se utilizará para evaluar técnicas de estimación y ahorro de energía relacionadas con reducción de *glitch*. En particular la estimación de la actividad de *glitch* en la herramienta de síntesis Fusion Compiler. Si bien la capacidad de estimar *glitch* ya estaba presente en las herramientas de *sign-off* como PrimePower, estas solo se utilizan en etapas finales del flujo de diseño en donde las oportunidades para optimizar el circuito se ven reducidas notablemente pues realizar modificaciones requiere volver atrás en el flujo y esto significa un gran costo en términos de ingeniería pues es necesario repetir una gran cantidad de verificaciones para asegurar que los cambios no afectaron negativamente otros aspectos del desempeño del *chip*.

Por lo tanto, el objetivo del proyecto a evaluar es la implementación de técnicas de estimación de *glitch* en etapas más tempranas del flujo en donde todavía hay suficiente flexibilidad para realizar optimizaciones que apunten a reducir el consumo producido por transiciones espurias.

El funcionamiento del estimador consiste en activarlo mediante una opción de la herramienta al principio del flujo de diseño, luego al compilar la *netlist* el motor de propagación de *switching activity* generara los datos de *glitch rate* para la *netlist*, junto con los datos clásicos de *toggle rate* y *static probability*. Una limitación importante del estimador es que solo es capaz de generar y propagar *glitches* de transporte por lo que el análisis no incluirá los *glitch* inerciales.

4.2.2. Aplicación de la metodología

En este caso la metodología se aplicará para determinar la efectividad del estimador de *glitch* para calcular la generación y propagación de estos en cada nodo del circuito. Como referencia se utilizarán los archivos SAIF generados por simulación *Full Timing Gate Level*. También se comparará el resultado del análisis de potencia generado por la herramienta de síntesis con el de la herramienta de *sign-off* especializada en análisis de potencia, la cual cuenta con la capacidad de incorporar información de *glitches* a la actividad del circuito bajo análisis.

4.2.3. Resultados y análisis

Según se plantea en la metodología lo primero que se realizara es verificar la integridad de los datos al pasar desde Fusion Compiler hacia Python. Para esto se cargó el SAIF de

verificación generado obteniéndose los resultados de la tabla 4.2. Como se puede ver la anotación de tipo simulada, que significa que los datos fueron obtenidos de un archivo SAIF, alcanza un valor muy cercano al 100 % como es el resultado esperado. Existen 7 nodos con anotación de tipo propagada, es decir, no están presentes en el SAIF por lo que son calculados por la herramienta. Estos se detallan en la tabla 4.3 y corresponden a nodos constantes como las fuentes de alimentación y nodos que se usan como constantes lógicas, los cuales no están presentes en la *netlist* escrita en formato Verilog, lo que explica la falta de anotación. Claramente esto no constituye un problema pues la actividad es nula para los 7 nodos, lo que implica que no existe disipación de potencia dinámica por lo que se pueden despreciar para efectos de cualquier análisis que concierna al presente trabajo.

Tabla 4.2: Reporte de anotación para el SAIF de verificación de consistencia.

Tipo de anotación	**Cantidad de nodos**	
Propagado	7	0.003 %
Simulado	234870	99.997 %
Total	234877	100 %

Tabla 4.3: Nodos sin anotación del SAIF de verificación de consistencia.

#	**Nodo**	**Descripción**
1	VDD	Fuente de alimentación principal
2	VSS	Tierra de alimentación principal
3	VCS	Fuente de alimentación RAMs
4	sparc0/ifu/ifu/swl/*Logic0*	Constante 0 lógico
5	sparc0/ifu/ifu/ifqdp/*Logic0*	Constante 0 lógico
6	sparc0/ifu/ifu/invctl/*Logic0*	Constante 0 lógico
7	sparc0/tlu/tlu/intctl/*Logic0*	Constante 0 lógico

Posteriormente se cargan los datos del archivo SAIF en Fusion Compiler y se exportan hacia Python para realizar la comparación contra los datos de referencia (SAIF). El resultado se puede apreciar en la figura 4.1 se ven los histogramas del error de correlación los cuales correctamente indican que todos los nodos tienen un error del 0 %, que es el resultado ideal para la evaluación.

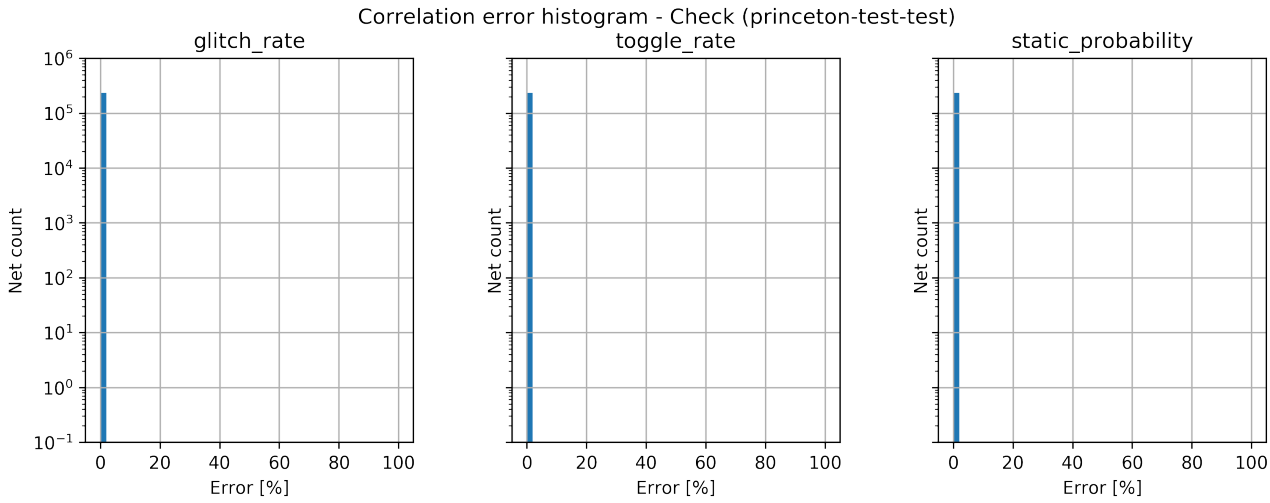


Figura 4.1: Histograma de error de correlación para verificación de consistencia.

Tabla 4.4: Resumen resultados verificación de consistencia

		Error promedio		
Caso	**Programa**	**Glitch rate**	**Toggle rate**	**Static probability**
Check	princeton-test-test	0.0 %	0.0 %	0.0 %

Con estos resultados es posible verificar que el traspaso de datos entre las distintas herramientas que se utilizan para el análisis comparativo de datos de *switching activity* son consistentes, por lo que se puede concluir que la evaluación parte de una buena base.

Para medir la correlación del estimador de *glitch* se utilizarán tres casos distintos al variar la fuente de información de *switching activity* para el motor de propagación. Lo anterior claramente sin incluir datos de *glitch* pues se busca evaluar el estimador y considerando que para dicha propagación se requiere de cierta cantidad de información inicial para generar la anotación de *switching activity* de la *netlist* completa. Luego los casos a analizar se muestran en la figura 4.2 y serán descritos a continuación.

Referencia:

Descripción: Datos del archivo SAIF producido por VCS a partir de simulación *Full-Timing Gate-Level*.

Objetivo: Se utilizarán estos datos como línea base para evaluar los resultados obtenidos a partir del estimador de *glitch*.

RM:

Descripción: Corresponde al *switching activity* anotado al final del flujo de síntesis. Dicho flujo parte con el SAIF de simulación RTL por lo que el motor de propagación calcula la actividad de los nuevos nodos que aparecen al implementar el diseño. Su nombre deriva de *Reference Methodology* que es el flujo de implementación que se utilizó como base.

Objetivo: Evaluar el estimador en un caso de uso durante el diseño de un *chip*.

RTL:

Descripción: Se leerá el SAIF RTL y se anotara directamente en la *netlist* sintetizada, luego se eliminará la anotación de los nodos no esenciales para el estimador y se ejecutara la propagación de *switching activity*.

Objetivo: Caso aislado de las perturbaciones que pudieran ser generadas por el flujo de síntesis, en el que la *netlist* sufre múltiples modificaciones.

GL:

Descripción: Se leerá el SAIF GL y se anotara directamente en la *netlist* sintetizada, luego se eliminará la anotación de los nodos no esenciales para el estimador y se ejecutará la propagación de *switching activity*.

Objetivo: Evaluar el estimador en el mejor caso posible, es decir, con la información esencial más precisa pues proviene directamente del simulador. Notar que este caso es el más sintético de los tres pues realizar simulación *Gate Level* es una práctica poco usual en los flujos de diseño de *chips* dada la complejidad de realizar la simulación misma como el alto tiempo de ejecución requerido.

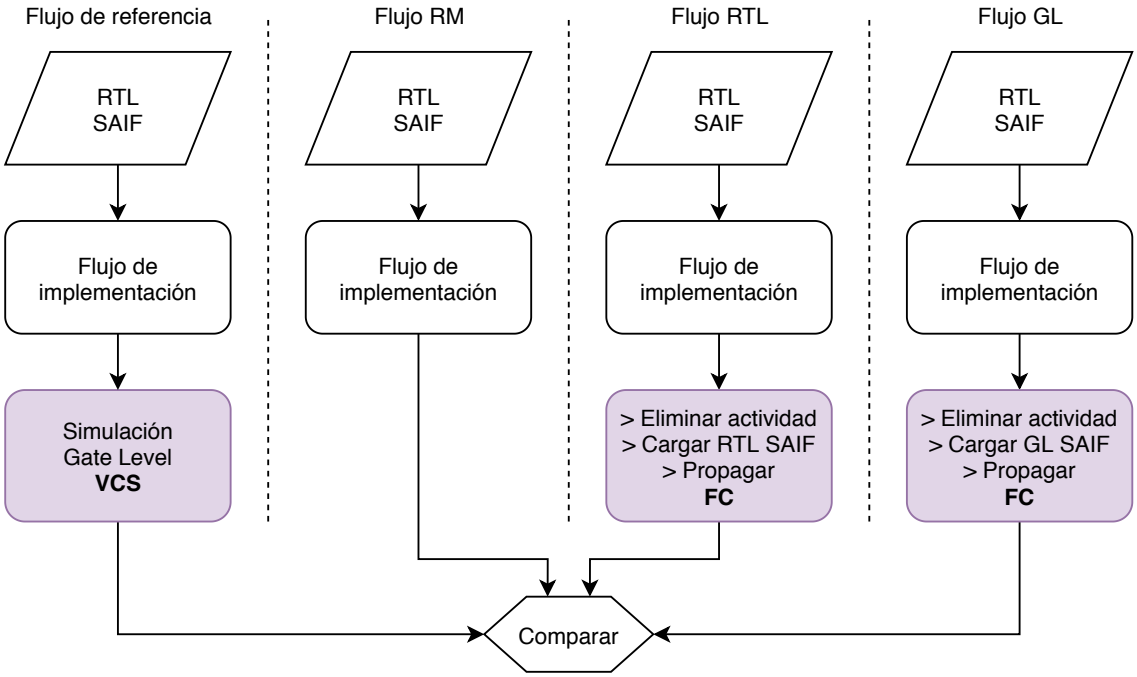


Figura 4.2: Casos para evaluar la correlación de actividad del estimador.

Para cada uno de los casos anteriores se obtienen los reportes de anotación presentados en las tablas siguientes. Estos corresponden a las anotaciones de *switching activity* posteriores a la ejecución del flujo, para el caso RM, y de ejecutar la propagación para los casos RTL y GL.

En general para los resultados mostrados de las tablas 4.5, 4.6 y 4.7, la cantidad de nodos no esenciales o propagados corresponden a aproximadamente un 55 % del total. Para los casos RM y RTL en las tablas 4.5 y 4.6, aproximadamente un 15 % de los nodos presentan actividad por defecto, lo que significa que no están presentes en el SAIF RTL ni fueron propagados por lo que presentan un valor predefinido en la herramienta (*toggle rate* 0.1 y *static probability* 0.5). Además, se observa que las anotaciones son bastante similares lo que comprueba el correcto funcionamiento del mapa entre nodos RTL y de *netlist*. También se debe destacar que para el caso GL en la tabla 4.7 la anotación simulada corresponde a aproximadamente el doble de la de los casos RM y RTL lo que es esperable pues en simulación *Gate Level* se cuenta con todos los nodos tanto esenciales como no esenciales de la *netlist*.

Tabla 4.5: Reporte de anotación para el caso RM.

Tipo de anotación	**Cantidad de nodos**	
Propagado	136898	58.285 %
Defecto	36907	15.7133 %
Simulado	61072	26.0017 %
Total	234877	100 %

Tabla 4.6: Reporte de anotación para el caso RTL.

Tipo de anotación	**Cantidad de nodos**	
Propagado	136898	58.285 %
Defecto	34818	14.8239 %
Simulado	63161	26.8911 %
Total	234877	100 %

Tabla 4.7: Reporte de anotación para el caso GL.

Tipo de anotación	**Cantidad de nodos**	
Propagado	121744	51.8331 %
Simulado	113133	48.1669 %
Total	234877	100 %

Para complementar la representatividad del análisis al considerarse múltiples escenarios también se realizarán las mediciones con un segundo programa de prueba (código para la CPU) distinto al que se utilizó como estímulo para la simulación RTL con la que se obtuvo el SAIF que se cargó en la herramienta de síntesis. Luego los programas a emplearse son:

princeton-test-test Este programa calcula la suma de los números del 0 al 9 en forma iterativa y al finalizar verifica que el resultado sea 45. Utilizado para obtener el SAIF RTL que se utilizó como entrada para el flujo de implementación.

exu_muldiv_stress_1 Programa adicional que se utiliza para realizar un stress test de la unidad de multiplicación del procesador. Se eligió debido a que el *datapath* del multiplicador alberga grandes cantidades lógica combinacional la cual es propensa a generar y propagar *glitches*. Notar que para este programa no es posible obtener resultados para el caso RM pues se requeriría cargar un nuevo SAIF RTL en el flujo de síntesis lo que produciría una *netlist* completamente distinta lo que dificultaría el análisis comparativo.

Una vez ejecutados los flujos especificados en la metodología, se obtuvieron las correspondientes tablas de *switching activity* en los nodos del circuito para cada combinación de caso (RM, RTL y GL) y programa de estímulo (princeton-test-test y exu_muldiv_stress_1) según sea corresponda. Como se mencionó en la metodología y dado el alto volumen de datos obtenidos a continuación se presentarán los vectores por sus principales estadísticas descriptivas. Para tener una mejor idea de la calidad de la salida del estimador en los resultados se incluyen los datos de *switching activity* tradicional, *toggle rate* y *static probability*, esto pues la herramienta cuenta con soporte para propagarlos desde versiones más antiguas por lo que también se pueden usar como referencia para evaluar las métricas de correlación de *glitch*. Los resultados presentados en las tablas 4.8 a 4.14 se normalizaron con respecto al *toggle rate* del reloj para las columnas *glitch rate* y *toggle rate*, mientras que *static probability* se expresa sin normalizar y en porcentaje.

Tabla 4.8: Estadísticas para el caso de referencia, programa princeton-test-test.

	Glitch rate	**Toggle rate**	**Static probability**
mean	0,01 %	0,08 %	33,36 %
std	0,16 %	2,22 %	40,36 %
max	49,43 %	100,00 %	100,00 %
min	0,00 %	0,00 %	0,00 %

Tabla 4.9: Estadísticas para el caso RM, programa princeton-test-test.

	Glitch rate	**Toggle rate**	**Static probability**
mean	0,00 %	0,89 %	32,41 %
std	0,07 %	3,00 %	40,73 %
max	12,21 %	99,26 %	100,00 %
min	0,00 %	0,00 %	0,00 %

Tabla 4.10: Estadísticas para el caso RTL, programa princeton-test-test.

	Glitch rate	**Toggle rate**	**Static probability**
mean	0,00 %	0,85 %	32,41 %
std	0,07 %	2,98 %	40,73 %
max	12,21 %	99,26 %	100,00 %
min	0,00 %	0,00 %	0,00 %

Tabla 4.11: Estadísticas para el caso GL, programa princeton-test-test.

	Glitch rate	**Toggle rate**	**Static probability**
mean	0,01 %	0,09 %	33,39 %
std	0,31 %	2,26 %	40,41 %
max	48,77 %	100,00 %	100,00 %
min	0,00 %	0,00 %	0,00 %

Los resultados para el programa princeton-test-test comienzan con la tabla 4.8 en la que se muestran los resultados obtenidos de VCS y que se utilizan como referencia, es decir, como los valores objetivo que se esperan para los tres casos siguientes de las tablas 4.9, 4.10 y 4.11, los cuales se obtienen del motor de propagación de Fusion Compiler.

Tabla 4.12: Estadísticas para el caso de referencia, programa exu_muldiv_stress_1.

	Glitch rate	**Toggle rate**	**Static probability**
mean	0,03 %	0,17 %	35,37 %
std	0,31 %	2,25 %	39,03 %
max	49,84 %	100,00 %	100,00 %
min	0,00 %	0,00 %	0,00 %

Tabla 4.13: Estadísticas para el caso RTL, programa exu_muldiv_stress_1.

	Glitch rate	**Toggle rate**	**Static probability**
mean	0,00 %	0,95 %	33,58 %
std	0,10 %	3,01 %	39,33 %
max	12,41 %	99,79 %	100,00 %
min	0,00 %	0,00 %	0,00 %

Tabla 4.14: Estadísticas para el caso GL, programa exu_muldiv_stress_1.

	Glitch rate	**Toggle rate**	**Static probability**
mean	0,01 %	0,20 %	35,41 %
std	0,33 %	2,33 %	38,96 %
max	49,56 %	100,00 %	100,00 %
min	0,00 %	0,00 %	0,00 %

Luego se tienen los resultados equivalentes con el segundo programa de prueba, con la excepción del caso RM que no se puede generar con la misma *netlist*. Los datos pueden ser revisados en las tablas 4.12, 4.13 y 4.14.

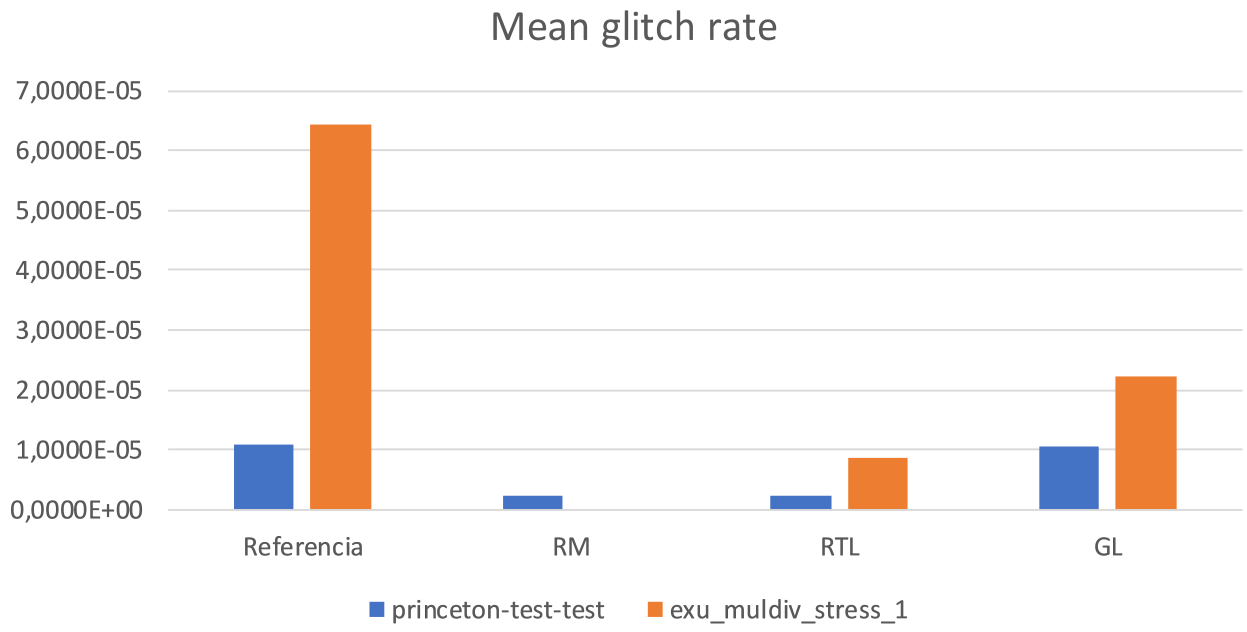


Figura 4.3: *Glitch rate* promedio.

Para facilitar la comparación se presenta el grafico de la figura 4.3 donde se visualiza el promedio del *glitch rate* para cada caso, en donde para la referencia se observa un incremento de 486 % en *glitch rate* al emplear el programa de stress test para el multiplicador. Ademas, es posible notar que el estimador tiende a subestimar el nivel de actividad de *glitch* en los casos basados en SAIF RTL, mientras que en el caso GL la estimación para el primer programa es muy cercana a la referencia, no así con el segundo programa donde se observa una diferencia considerable lo que debe ser investigado para comprobar si existe un problema en el estimador de *glitch*.

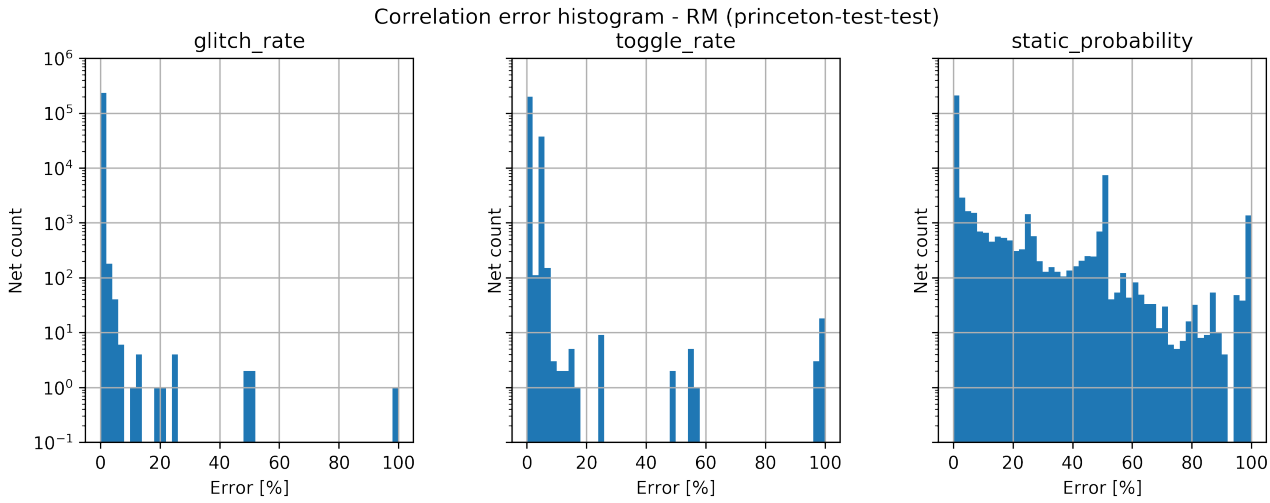


Figura 4.4: Histograma de error de correlación para el caso RM, programa princeton-test-test.

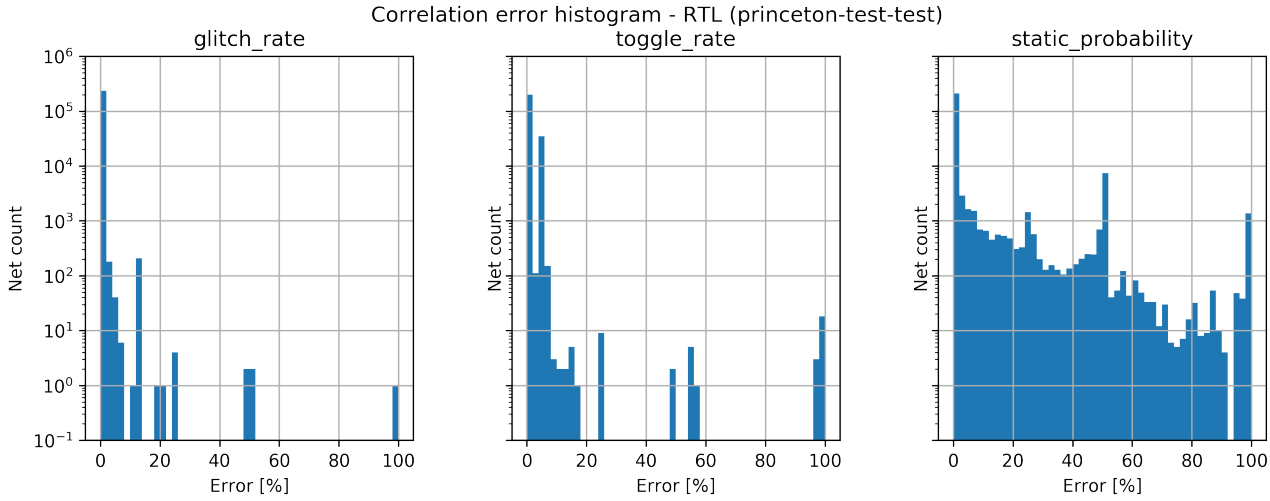


Figura 4.5: Histograma de error de correlación para el caso RTL, programa princeton-test-test.

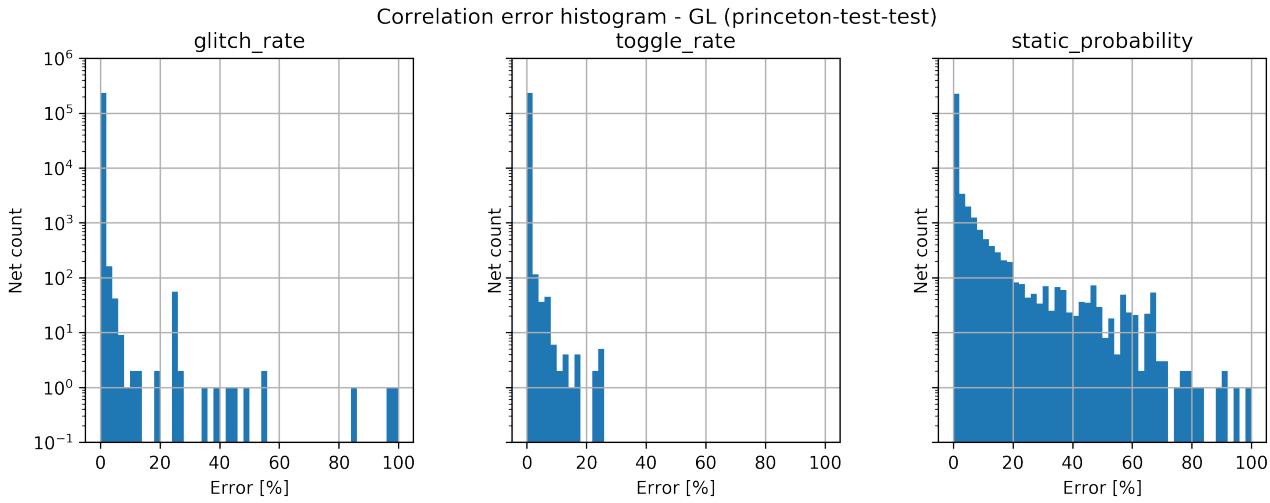


Figura 4.6: Histograma de error de correlación para el caso GL, programa princeton-test-test.

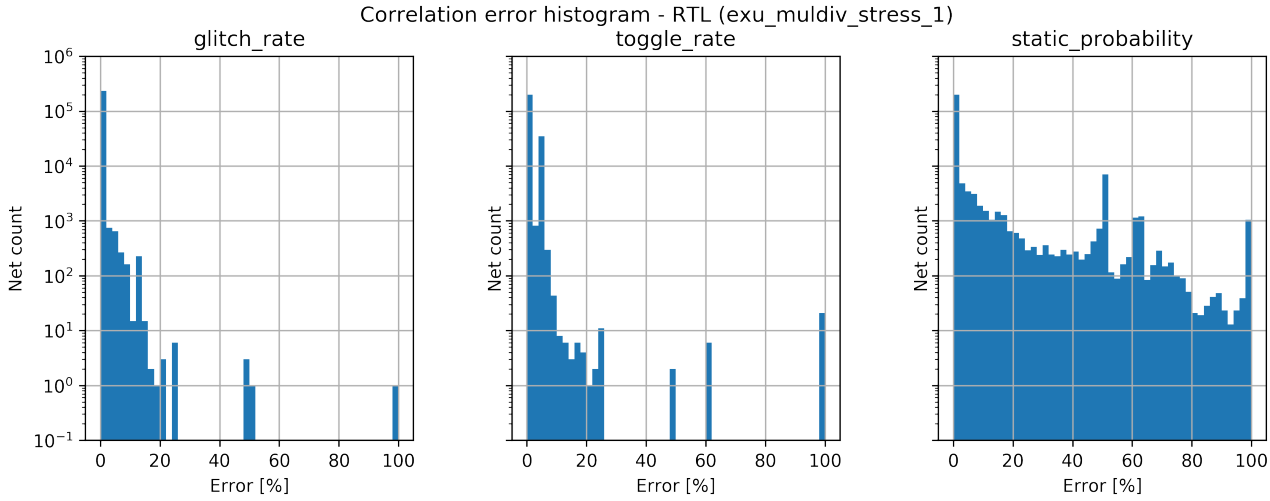


Figura 4.7: Histograma de error de correlación para el caso RTL, programa exu_muldiv_stress_1.

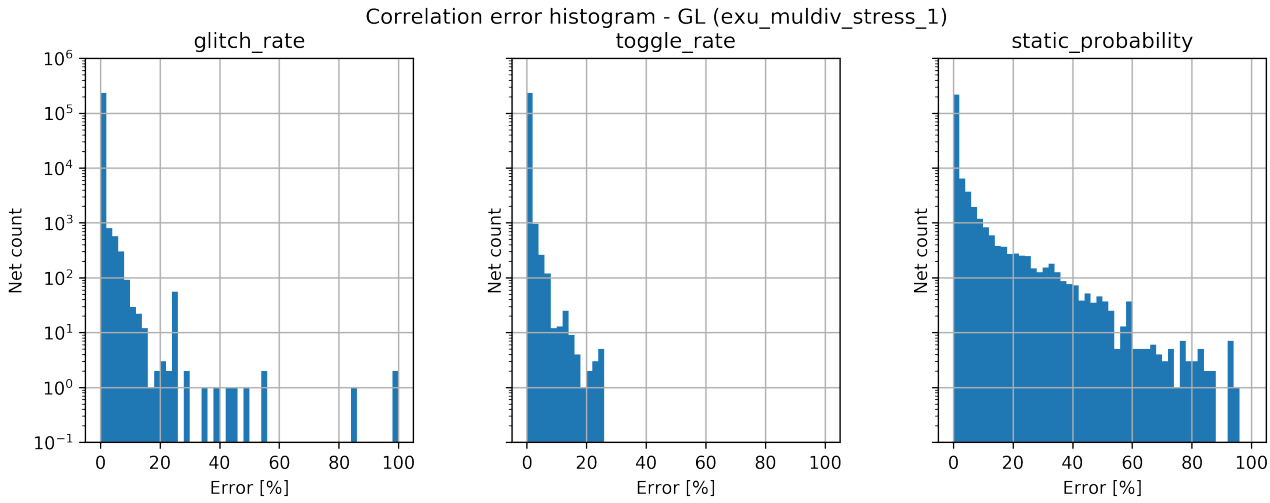


Figura 4.8: Histograma de error de correlación para el caso GL, programa exu_muldiv_stress_1.

Para caracterizar la correlación en forma más detallada o comparando nodo a nodo se presentan los histogramas de las figuras 4.4, 4.5 y 4.6 para el programa princeton-test-test, así como las figuras 4.7 y 4.8 para el programa exu_muldiv_stress_1. Estos se obtienen de calcular el error como el valor absoluto de la diferencia entre el valor de referencia y el estimado para cada nodo. Como se puede apreciar las distribuciones en general presentan una clara tendencia hacia el error 0 % en donde se encuentran la mayoría de los nodos, con la excepción del campo *static probability* que cuenta con una distribución mucho más distribuida entre 0 % y 100 %.

Tabla 4.15: Resumen resultados programa princeton-test-test

	Error promedio		
Caso	**Glitch rate**	**Toggle rate**	**Static probability**
RM	0.0126 %	0.8201 %	3.4298 %
RTL	0.0244 %	0.7756 %	3.4298 %
GL	0.0189 %	0.0173 %	0.4389 %

Tabla 4.16: Resumen resultados programa exu_muldiv_stress_1

	Error promedio		
Caso	**Glitch rate**	**Toggle rate**	**Static probability**
RTL	0.0759 %	0.8051 %	4.6123 %
GL	0.0678 %	0.0601 %	0.7937 %

Para finalizar la comparación de *switching activity* se cuantifica la información contenida en los histogramas mediante el error del vector de error, el resultado puede consultarse en las tablas 4.15 y 4.16, en donde no está presente el caso RM para el programa exu_muldiv_stress_1 pues como se mencionó anteriormente se requeriría generar una nueva implementación y los resultados no serían comparables. En estas tablas puede confirmarse que el caso GL fue en general el más cercano a los resultados de referencia en los tres campos de *switching activity*.

Tabla 4.17: Reportes de potencia [Watts], programa princeton-test-test.

Caso	**Glitch Power**	**Internal Power**	**Leakage Power**	**Switching Power**	**Total Power**
Referencia	4,3572E-06	3,2135E-03	3,9662E-05	2,4835E-04	3,5015E-03
RM	1,1904E-06	4,2491E-03	1,0085E-04	4,4796E-04	4,7991E-03
RTL	1,1903E-06	4,2491E-03	1,0085E-04	4,4797E-04	4,7991E-03
GL	7,3236E-06	2,3178E-03	1,0082E-04	2,6250E-04	2,6885E-03

Tabla 4.18: Reportes de potencia [Watts], programa exu_muldiv_stress_1.

Caso	**Glitch Power**	**Internal Power**	**Leakage Power**	**Switching Power**	**Total Power**
Referencia	2,3755E-05	3,4124E-03	3,9772E-05	3,2624E-04	3,7784E-03
RTL	4,0514E-06	4,5029E-03	1,0095E-04	5,2680E-04	5,1347E-03
GL	1,3777E-05	2,4539E-03	1,0094E-04	3,4133E-04	2,9099E-03

Dado que la métrica de calidad del *chip* que interesa realmente a los diseñadores es la potencia dinámica disipada, y en este sentido los datos de *switching activity* corresponden a una herramienta para realizar análisis de potencia, se procede a evaluar la correlación en términos de potencia con respecto a los resultados obtenidos de la herramienta especializada PrimePower. Para cada programa se obtuvieron los reportes de potencia de las tablas 4.17 y 4.18, en los que se detalla la potencia total consumida, así como su desglose en potencia de *glitch*, interna, fuga y conmutación. De entre estos datos es destacable que para los valores de referencia se observa un incremento de 445 % en potencia consumida por *glitch* al aplicar el stress test del multiplicador, lo cual es consistente con el incremento de *switching activity* visto en la figura 4.3.

Tabla 4.19: Error de análisis de potencia, programa princeton-test-test.

Caso	**Glitch Power**	**Internal Power**	**Leakage Power**	**Switching Power**	**Total Power**
RM	-72,68 %	32,23 %	154,27 %	80,38 %	37,06 %
RTL	-72,68 %	32,23 %	154,26 %	80,38 %	37,06 %
GL	68,08 %	-27,87 %	154,20 %	5,70 %	-23,22 %

Tabla 4.20: Error de análisis de potencia, programa exu_muldiv_stress_1.

Caso	**Glitch Power**	**Internal Power**	**Leakage Power**	**Switching Power**	**Total Power**
RTL	-82,95 %	31,96 %	153,81 %	61,48 %	35,90 %
GL	-42,01 %	-28,09 %	153,81 %	4,63 %	-22,99 %

En las tablas 4.19 y 4.20 se muestra el error con respecto al valor de referencia para las potencias reportadas. En estas el signo negativo representa el caso en el que el valor estimado es menor a la referencia y el signo positivo el caso contrario, además los porcentajes son relativos al valor de referencia. En cuanto a los resultados se observa que el caso GL es el más exacto lo que se explica por la mayor precisión de los datos de *switching activity*. Finalmente se obtiene una diferencia prácticamente nula entre los casos RTL y RM.

4.3. Técnicas de optimización y análisis para Self Gating

4.3.1. Descripción del caso

En el presente caso de estudio se evaluará la aplicación de metodologías de optimización lógica y análisis de potencia para inserción de *self gating*, cuya revisión se hizo en la Subsubsección 2.5.1.2. *Self gating* es una técnica derivada de la aplicación de *clock gating* a registros cuyo valor conmuta una cantidad reducida de veces durante el funcionamiento normal del *chip*.

Respecto a esta técnica en primer lugar se evalúa la selección automática o basada en optimización de la compuerta detectora de cambios en el valor del registro. Esto significa utilizar una compuerta distinta de XOR en ciertos casos, como compuertas AND y OR, según sea conveniente de acuerdo con la actividad de cada registro. Con esto se logra reducir el impacto en área, pues la compuerta XOR es la compuerta primitiva más grande en cantidad de transistores por lo que además presentara el mayor consumo de potencia asociado por lo que incluso podría obtenerse una mayor cantidad de *self gates* insertadas.

En segundo lugar, se evalúa la aplicación de un flujo alternativo de propagación de *switching activity* con el objetivo de incrementar la cobertura de *self gating* en el circuito, donde cobertura se entiende como el porcentaje de los registros con *self gating* al finalizar el flujo de síntesis. Típicamente el flujo de análisis de potencia consiste en anotar la actividad de simulación RTL transferida a través de un archivo SAIF en la *netlist* mediante un mapa entre los nodos RTL y los de la *netlist* que se va transformando a lo largo de las diversas transformaciones ejecutadas por síntesis manteniéndose vigente la actividad de simulación tanto como sea posible y propagando el resto de los nodos. Sin embargo, existen técnicas como *self gating* que requieren un flujo de propagación distinto para incrementar la precisión, el cual consiste en forzar la propagación de nodos específicos incluso si tienen actividad anotada a partir de SAIF. La precisión reducida de la propagación de *switching activity* puede ser perjudicial pues introduce pesimismo en la evaluación de factibilidad de inserción de *self gates*.

4.3.2. Aplicación de la metodología

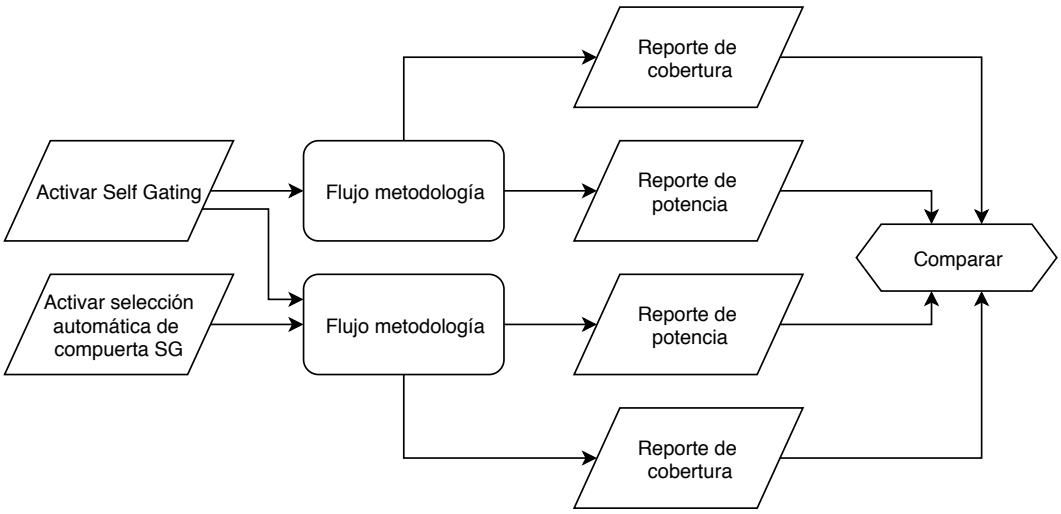


Figura 4.9: Aplicación de la metodología para evaluar selección automática de compuerta de compuerta detectora en *self gating*.

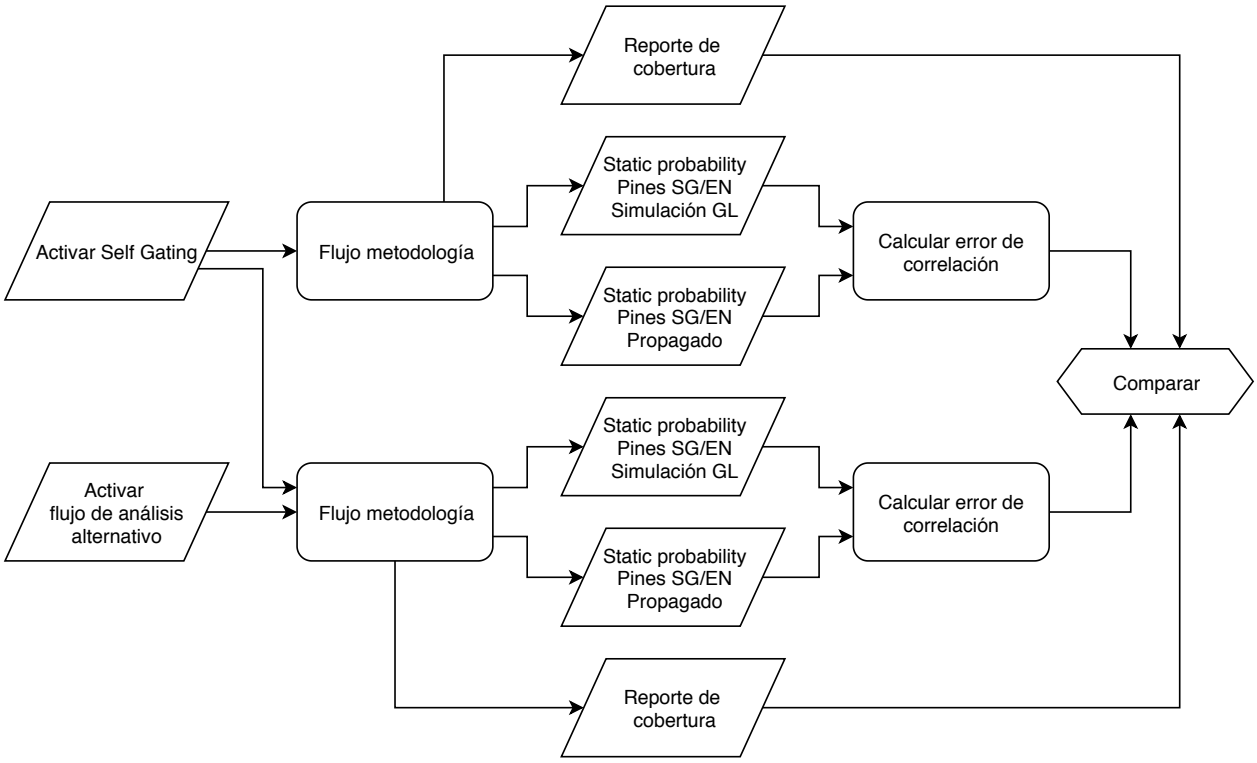


Figura 4.10: Aplicación de la metodología para evaluar flujo de propagación alterno de *self gating*.

Para cada una de las evaluaciones cuyos flujos quedan representados en las figuras 4.9 y 4.10, se aplicará la metodología para obtener dos versiones de la CPU, en donde una utilizará

el flujo regular de inserción de *self gating* y servirá como referencia, mientras que la otra utilizara una de las técnicas bajo evaluación, activadas mediante opciones de la herramienta de síntesis. Por lo tanto, se utilizará el flujo de la metodología hasta el punto en que se obtiene la *netlist* sintetizada por Fusion Compiler, no siendo necesario ejecutar el análisis de potencia en PrimePower. Ademas, para ambas evaluaciones se compara la cobertura, y en particular para la primera se analiza el ahorro de potencia, mientras que, para la segunda, el error de correlación entre simulación GL y propagación para los nodos de interés, pues el objetivo es verificar si existe una reducción de error.

4.3.3. Resultados y análisis

Debido a que se utilizaran los mismos métodos para transferir datos desde la herramienta de síntesis hacia Python para realizar el análisis de datos, se considera que las pruebas de validación de consistencia realizados para el caso de estudio de la Sección 4.2 siguen siendo válidos para este análisis.

Tabla 4.21: Resultados de PPA obtenidos al activar la selección automática de compuerta detectora en *self gating*.

	Referencia	**Selección automática**	**Variación**
Total Power	5,1923E-03 W	4,7023E-03 W	-9,44 %
Internal Power	3,8665E-03 W	3,3475E-03 W	-13,42 %
Switching Power	1,2243E-03 W	1,2536E-03 W	2,39 %
Leakage Power	1,0151E-04 W	1,0128E-04 W	-0,23 %
Setup Slack	0,5038 ps	0,1048 ps	-79,20 %
Utilization Ratio	52,04 %	50,75 %	-1,29 %

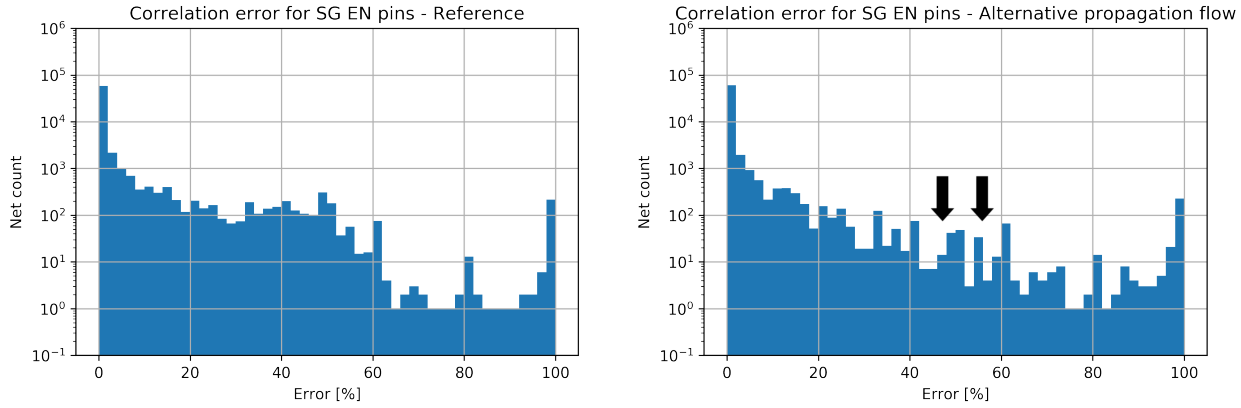
En la tabla 4.21 se entregan los resultados en términos de PPA al sintetizar con selección automática de compuerta asociada a *self gating*, donde se utiliza como referencia el caso cuyas *self gates* utilizan la compuerta XOR exclusivamente. Se produjo un descenso en la potencia total del 9.44 %, del cual la mayor parte proviene una mejora en el consumo de potencia interna, lo cual es consistente con la técnica bajo evaluación pues el principio es que las compuertas AND y OR consumen una menor potencia que la XOR. Desde el punto de vista de la velocidad del circuito se produjo una degradación de 79.2 % y en área se produjo una reducción del 1.29 % la cual nuevamente podría explicarse dado el menor tamaño de las compuertas elegidas. Ademas, la cobertura de *self gating* paso de 22.92 % a 24.23 %, es decir, un incremento del 1.31 %.

Luego, como primera prueba del flujo alternativo de análisis se hizo un par de corridas de síntesis como se aprecia en la figura 4.10, ambas utilizando opciones de configuración de *self gating* tal como las que se utilizarían en un diseño real, donde se obtuvo una cobertura de 22.92 % para el caso de referencia y de 27.26 % para el caso bajo evaluación, obteniéndose así un aumento de 4.34 % en la cobertura por efecto de activar el flujo alterno. Todos los porcentajes son relativos a un total de 67056 registros en el diseño.

Tabla 4.22: Estadísticas de *static probability* en los *pines* de *enable* de las *self gates*.

	Referencia		**Flujo alternativo**	
	Simulación GL	**Propagado**	**Simulación GL**	**Propagado**
mean	0,0250	0,0377	0,0250	0,0245
std	0,0981	0,1249	0,0981	0,1047
min	0,0000	0,0000	0,0000	0,0000
max	1,0000	1,0000	1,0000	1,0000

Luego para evaluar el método en términos de reducción de error en la propagación de *static probability* para el *enable* de las *self gates*, se ejecutó un segundo par de flujos de síntesis forzando la inserción de *self gating* en todos los registros del circuito. De esta forma los resultados al calcular el error de correlación tendrán una mayor representatividad. Para estos casos los vectores de *static probability* en los *pines* de interés están descritos por la tabla 4.22 donde ya se puede apreciar una mejora pues el error calculado como diferencia entre los promedios de los datos de simulación menos los propagados pasa de $50.62\,\%$ a $-2,03\,\%$.



(a) Caso de referencia.

(b) Caso con flujo de propagación alternativo.

Figura 4.11: Histogramas de error de correlación.

En de la figura 4.11 se muestran los histogramas de error de correlación, en los cuales se utilizan los datos de simulación GL como valores de referencia para cada caso. En estos se observa claramente una disminución del error en los nodos con error cercano al $50\,\%$, y el valor promedio para cada vector de error pasa de $2.49\,\%$ a $1.53\,\%$, es decir, una reducción de $0.96\,\%$.

Tabla 4.23: Reportes de potencia [Watts] para los casos de referencia y flujo alternativo de *self gating*.

	Referencia	**Flujo alternativo**
Cell Internal Power	7,53E-03	7,05E-03
Net Switching Power	2,19E-03	2,04E-03
Total Dynamic Power	9,72E-03	9,10E-03

Para finalizar, de la tabla 4.23 se observa que el flujo alternativo produce una reducción de potencia dinámica de 6.42 % respecto del caso base. Con estos resultados se concluye que ambos métodos son efectivos en mejorar la cobertura y efectividad de *self gating*. Para el caso de selección automática de compuerta se observan un ahorro de potencia considerable al aplicar la técnica, así como una leve disminución en área. La degradación de potencia debe ser estudiada en detalle, sin embargo, existen muchos factores adicionales a la aplicación de esta técnica que podrían estar afectando esta métrica, pues las restricciones de tiempo están muy ligadas al proceso de síntesis física en donde podría haber diferencias considerables en la ubicación y enrutamiento de las celdas. Para el flujo alternativo de análisis se encontró que se produjo la disminución esperada en el error de propagación, y se incrementan las posibilidades de inserción de *self gating* lo que se observa en la disminución del 35.01 % en el *static probability* propagado para el pin de *enable* de las *self gates*.

Capítulo 5

Conclusiones

Con esta memoria se logró obtener una metodología completa para evaluar el desempeño de técnicas de reducción de potencia dinámica en circuitos VLSI a partir de un diseño de pruebas junto con la infraestructura necesaria para implementarlo y simularlo utilizando las herramientas de Synopsys. La metodología desarrollada también contempla flujos de traspaso de información desde las herramientas de implementación hacia el sistema de procesamiento de datos de Python que cuenta con amplias capacidades y es de uso extendido en la industria. Por lo tanto, se consigue un flujo de evaluación flexible, aplicable a diversas técnicas de reducción y análisis de potencia en la herramienta de síntesis. A partir del cual se podría expandir el alcance a otros aspectos no relacionados con potencia dinámica del flujo de diseño.

Con relación al desarrollo práctico de la metodología se pudo elegir y obtener un diseño de pruebas de complejidad similar a los diseños actuales el cual cuenta con todo lo necesario para generar un flujo de diseño completo. También se pudo sintetizar el diseño en las herramientas de nueva generación que se requiere evaluar y ejecutar exitosamente simulación *Full Timing Gate Level* del diseño sintetizado, a partir de la cual es posible obtener datos de actividad del circuito de la mayor precisión posible sin fabricar y medir el *chip* físicamente. Finalmente contar con un diseño en RTL y un flujo de implementación completo permite obtener una metodología con capacidad de evaluar una gran cantidad de técnicas pues es posible modificar partes del diseño, así como del flujo para facilitar optimizaciones e incluir nuevos tipos de análisis según se requiera.

Sobre la aplicación de la metodología propuesta se realizaron dos casos de estudio en donde el primero consistió en evaluar el desempeño de un estimador de *glitch* para la herramienta de síntesis con el objetivo de soportar técnicas de reducción de *glitch* y por lo tanto de la potencia asociada. La evaluación dio resultados positivos en términos de la calidad de la estimación de actividad producida por *glitch*, lo que se demuestra en una sólida correlación entre la cantidad de *glitches* estimados y los que se obtuvieron del simulador, con un error promedio de 0.04 % para todos los casos. A pesar de lo anterior los resultados cuando se compara la potencia reportada por la herramienta de síntesis basada en el estimador con la reportada por la herramienta de *sign-off* basada en los datos de simulación, no fueron consistentes con un 67.68 % de error promedio considerando todos los casos. Esto podría estar relacionado con la baja cantidad de *glitches*, relativo a la bibliografía [17], que se obtuvieron del circuito. Utilizando los datos de actividad del simulador para el programa exu_muldiv_stress_1 se

obtuvo un 0.63 % de potencia de *glitch* con respecto a la potencia dinámica total, por lo que no es claro si este resultado es realmente representativo y se necesitaría otro diseño, o bien contar con una mayor porción del OpenPiton dentro de la metodología, para obtener una mayor cantidad de *glitches*.

Luego sobre el segundo caso de estudio en el que se aplicó la metodología se evaluaron dos técnicas asociadas a *self gating* en donde los resultados fueron concluyentes en que ambas mejoran la cobertura y por lo tanto incrementan el ahorro de energía al aplicar *self gating*. Para la técnica de selección automática de compuerta detectora se pudo demostrar que efectivamente hay una reducción de potencia de 9.44 % cuando se elige otro tipo de compuerta distinto al XOR para detectar los cambios en los valores de los registros. Además, se incrementa la cobertura en 1.31 %. En el segundo caso se probó que el flujo alternativo para propagar actividad del pin de *enable* de las *self gates* reduce el error de correlación en un 52.65 %, de tal forma que los resultados son más favorables a la inserción y por lo tanto se incrementa la cobertura en 4.34 %. Esto se traduce en una reducción de 6.42 % de la potencia dinámica total.

5.1. Propuestas para trabajos futuros

El trabajo futuro que se deriva de esta memoria consiste principalmente en extender la metodología de evaluación propuesta, para esto se sugieren los siguientes objetivos:

- Continuar desarrollando el flujo de implementación del *chip* hasta sintetizar el procesador *manycore* completo. Esto con el objetivo de obtener una mayor cantidad de *glitches* y contar con un caso de pruebas más grande sobre el que ampliar la metodología.
- Obtener y compilar una prueba de rendimiento utilizada en la industria para generar un mejor estímulo para el procesador. Dado que el estímulo ejerce una gran influencia en el diseño sintetizado y en el análisis de potencia que se basa en la actividad del circuito, es importante contar con estímulos representativos del funcionamiento práctico del diseño.
- Incorporar otros diseños que cumplan los requerimientos de la Sección 3.1 a la metodología. Una batería de diseños produciría resultados más representativos de los distintos casos en los que se utilizan las herramientas en la práctica. Además, permite cruzar datos para verificar la integridad de los resultados.
- Automatizar la metodología propuesta completamente, de forma que pueda ser ejecutada en forma periódica para obtener estadísticas en función del tiempo a medida que se desarrolla el código de las herramientas.
- Continuar el estudio de potencia consumida por *glitch* debido a que existen pocas fuentes de investigación recientes al respecto, en particular sobre potencia consumida por *glitch* inercial y su modelamiento en VLSI.

Glosario

Bit Digito binario.

Bottom-up Metodología de síntesis jerarquizada a partir de los módulos más simples.

Chip Circuito integrado.

Clock gating Técnica de reducción de potencia dinámica.

Clock skew Desfase temporal con que el reloj llega a distintas celdas.

Clock tree Red de distribución de reloj.

Datapath Subconjunto del diseño en el que se procesan los datos.

Driver Dispositivo que se conecta a un determinado *pin* de entrada de una celda lógica.

Fan out Conjunto de dispositivos alimentados por cierto *pin* de salida de una celda lógica.

Floorplan Conjunto de restricciones físicas de un *chip*.

Full Timing Gate Level Simulación a nivel de celda lógica con retardos.

Glitch rate Conmutaciones debidas a *glitch* por unidad de tiempo.

Hold slack Holgura en el tiempo mínimo de propagación.

IP Propiedad intelectual.

Integrated Clock Gate Celda lógica especializada para realizar *clock gating*.

Latch Celda secuencial asíncrona.

Layout Diseño físico de un chip utilizado para fabricar máscaras litográficas.

Netlist Representación de la conectividad de un circuito eléctrico.

Pad Contacto metálico mediante el que un *chip* se conecta a otros dispositivos.

Pin Puerto lógico en el que se conectan salidas o entradas de un módulo.

Pipeline Conjunto de etapas de lógica combinacional separadas por registros.

Placement Posicionamiento de las celdas en ubicaciones optimas.

RISC Arquitectura de procesador con instrucciones reducidas.

RTL Simulación o diseño a nivel de transferencia de registros.

Router Dispositivo encargado de enviar paquetes de datos por una red de comunicaciones.

SAIF Formato definido por la IEEE para intercambiar datos de *switching activity*.

SDC Formato definido por Synopsys para intercambiar restricciones de diseño.

SDF Formato definido por la IEEE para intercambiar información de retardos de un diseño.

STA Verificación estática de cumplimiento de restricciones temporales.

Self gating Técnica de reducción de potencia basada en *clock gating*.

Sequential gating Técnica de reducción de potencia basada en *clock gating*.

Setup slack Holgura en el tiempo máximo de propagacion.

Sign-off Etapa de verificación final antes de fabricar un *chip*.

Slew rate Tasa de cambio de voltaje en un intervalo de tiempo.

Static probability Probabilidad de que una señal se encuentre en 1 lógico.

Switching activity Caracterización de actividad para estimar potencia eficientemente.

Tech file Archivo que contiene restricciones de la tecnología de fabricación.

Toggle rate Conmutaciones funcionales por unidad de tiempo.

Wrapper Módulo que se utiliza para adaptar a otro, manteniendo la interfaz del original.

Bibliografía

[1] *IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems*. IEEE. doi: 10.1109/ieeestd.2019.8686430. URL https://doi.org/10.1109/ieeestd.2019.8686430.

[2] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. Openpiton: An open source manycore research framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 217–232, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4091-5. doi: 10.1145/2872362.2872414. URL http://doi.acm.org/10.1145/2872362.2872414.

[3] Abdellatif Bellaouar and Mohamed I. Elmasry. *Low-Power Digital VLSI Design*. Springer US, 1995. doi: 10.1007/978-1-4615-2355-0. URL https://doi.org/10.1007/978-1-4615-2355-0.

[4] Rakesh Chadha and J. Bhasker. *An ASIC Low Power Primer*. Springer New York, 2013. doi: 10.1007/978-1-4614-4271-4. URL https://doi.org/10.1007/978-1-4614-4271-4.

[5] B. Chen and I. Nedelchev. Power compiler: a gate-level power optimization and synthesis system. In *Proceedings International Conference on Computer Design VLSI in Computers and Processors*. IEEE Comput. Soc. doi: 10.1109/iccd.1997.628852. URL https://doi.org/10.1109/iccd.1997.628852.

[6] Yongseok Cheon, Pei-Hsin Ho, A.B. Kahng, S. Reda, and Qinke Wang. Power-aware placement. In *Proceedings. 42nd Design Automation Conference, 2005.* IEEE, 2005. doi: 10.1109/dac.2005.193924. URL https://doi.org/10.1109/dac.2005.193924.

[7] Tiansong Cui, Qing Xie, Yanzhi Wang, Shahin Nazarian, and Massoud Pedram. 7nm FinFET standard cell layout characterization and power density prediction in near- and super-threshold voltage regimes. In *International Green Computing Conference*. IEEE, November 2014. doi: 10.1109/igcc.2014.7039170. URL https://doi.org/10.1109/igcc.2014.7039170.

[8] Masanori Hashimoto, Hidetoshi Onodera, and Keikichi Tamaru. A power optimization method considering glitch reduction by gate sizing. In *Proceedings of the 1998 international symposium on Low power electronics and design - ISLPED '98*. ACM Press, 1998. doi: 10.1145/280756.280907. URL https://doi.org/10.1145/280756.280907.

[9] Gordon E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(3):33–35, September 2006. doi: 10.1109/n-ssc.2006.4785860. URL

https://doi.org/10.1109/n-ssc.2006.4785860.

[10] T. Mudge. Power: a first-class architectural design constraint. *Computer*, 34(4):52–58, April 2001. doi: 10.1109/2.917539. URL https://doi.org/10.1109/2.917539.

[11] David Money Harris Neil H. E. Weste. *CMOS VLSI Design: A Circuits and Systems Perspective (4th Edition)*. Addison Wesley, 4th edition, 2010. ISBN 0321547748,9780321547743.

[12] M.J.M. Pelgrom, A.C.J. Duinmaijer, and A.P.G. Welbers. Matching properties of MOS transistors. *IEEE Journal of Solid-State Circuits*, 24(5):1433–1439, October 1989. doi: 10.1109/jssc.1989.572629. URL https://doi.org/10.1109/jssc.1989.572629.

[13] Aviral Shrivastava Krishnaiah Gummidipudi (auth.) Preeti Ranjan Panda, B. V. N. Silpa. *Power-efficient System Design*. Springer US, 1 edition, 2010. ISBN 1441963871,9781441963871.

[14] A. Raghunathan, S. Dey, and N.K. Jha. Register transfer level power optimization with emphasis on glitch analysis and reduction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(8):1114–1131, 1999. doi: 10.1109/43.775632. URL https://doi.org/10.1109/43.775632.

[15] J.A. Ryu, S.C. Kim, J.D. Cho, H.W. Park, and Y.H. Chang. A new lower power viterbi decoder architecture with glitch reduction. In *AP-ASIC'99. First IEEE Asia Pacific Conference on ASICs (Cat. No.99EX360)*. IEEE. doi: 10.1109/apasic.1999.824034. URL https://doi.org/10.1109/apasic.1999.824034.

[16] Pallavi Saxena. Design of low power and high speed carry select adder using brent kung adder. In *2015 International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA)*. IEEE, January 2015. doi: 10.1109/vlsi-sata.2015.7050465. URL https://doi.org/10.1109/vlsi-sata.2015.7050465.

[17] J. G. Sudhakar, K. Tirupathi Rao, and B. Suresh. Glitch power minimization techniques in low power vlsi circuits. 2012.

[18] *Fusion Compiler User Guide*. Synopsys, 2019.

[19] *PrimePower User Guide*. Synopsys, 2019.

[20] *SiliconSmart User Guide*. Synopsys, 2019.

[21] *VCS User Guide*. Synopsys, 2019.

[22] Senthil Kumaran Varadharajan and Viswanathan Nallasamy. Low power VLSI circuits design strategies and methodologies: A literature review. In *2017 Conference on Emerging Devices and Smart Systems (ICEDSS)*. IEEE, March 2017. doi: 10.1109/icedss.2017.8073688. URL https://doi.org/10.1109/icedss.2017.8073688.