"El saber de mis hijos hará mi grandeza"

UNIVERSIDAD DE SONORA

División de Ciencias Exactas y Naturales

Departamento de Investigación en Física

Doctorado en Ciencias en Electrónica

Identificación de rutas lógicas influenciadas por acoplamientos capacitivos severos

T E S I S

Presentada en cumplimiento de los requisitos para obtener el grado de **Doctorado en Ciencias en Electrónica**
presenta:

MGTI Iván Dostoyewski Meza Ibarra

Director: **Dr. Roberto Gómez Fuentes**
Co-Director: **Dr. Victor Hugo Champac Vilela**

Hermosillo, Sonora, México, Mayo de 2022

Universidad de Sonora

Repositorio Institucional UNISON

"Con fundamento en los artículos 21 y 27 de la Ley Federal del Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada "**Identificación de rutas lógicas influenciadas por acoplamientos capacitivos severos**", otorgo de manera gratuita y permanente a la **Universidad de Sonora**, la autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una contraprestación".

MGTI IVÁN DOSTOYEWSKI MEZA IBARRA

19 DE MAYO DE 2022

FIRMA: _____

Dedicatoria

A mi familia, esposa e hijos, el motor de mi vida...

Agradecimientos

Tengo una "gran familia", a la que debo agradecer, el dejarme soñar, intentar, y ponerme retos, siempre depositaron su confianza en mí, además de su apoyo, muchas veces haciendo grandes sacrificios, esto es por ustedes. Abuela (Amá), mejor fuente de inspiración y motivación no pude tener en la vida. Papá, Mamá, sigan cosechando triunfos, todo ha válido la pena, gracias por enseñarme que siempre hay algo que aprender. Esposa e hijos, misión cumplida, siempre estuvimos juntos en la buenas y en las malas, una experiencia más en la bitácora de la vida.

A la coordinación y profesores del Posgrado en Electrónica, un gran grupo, siempre dispuesto a guiar, asesorar y apoyar a los estudiantes.

A mi Co-director de tesis, Dr. Victor Champac, debí ser el estudiante más complicado que haya tenido, pero su apoyo y orientación, siempre estuvo, es un placer ser parte del grupo.

A mi Director de tesis, Dr. Roberto Gomez, por su paciencia, apoyo y "pláticas motivacionales", como me lo planteó desde un principio, esto tiene un fin, mientras trabajemos, seremos amigos y enemigos. Solo usted sabe, que esto fue más allá de un proyecto académico, hubo muchos más eventos alternos, y los que siguen.

A la Universidad de Sonora, por fin mi nombre está registrado en la base de datos de estudiantes, desde que aprendí andar, he rondado, por rincones de esta gran institución. Aquí planeé muchos proyectos de mi vida, parte de ellos, los realicé aquí.

A todos, muchas gracias...

Resumen

El desarrollo de circuitos integrados es una industria propositiva en el uso de la tecnología, y por lo tanto partícipe en la generación de cambios. El desarrollo puede tener etapas generales: 1) diseño de circuito; 2) análisis de circuito; 3) simulación de circuito; 4) fabricación y 5) pruebas. Realmente en cada etapa hay una evaluación de lo que se espera que haga el circuito. No considerar ciertos factores impacta en el éxito de cada etapa, sobre todo en el aspecto de funcionamiento real y de mercadeo.

Se puede mejorar bastante el proceso de desarrollo si en la etapa de simulación se tiene un modelo que permita considerar factores, variables y valores que puedan ser alterados en un circuito integrado, que a su vez refleje el comportamiento apegado a la realidad. Esto hará que en todas las etapas, incluyendo la simulación, se pueda tener mejor certeza en su implementación, pero sobretodo impactar en las fases de evaluación, disminuyendo costos y tiempos.

La tendencia a la tecnología nanométrica ha hecho que se tengan que desarrollar nuevos procesos de simulación, así como estrategias de diseño, lo cual ha derivado a considerar más aspectos de evaluación, además de los ya establecidos.

En este trabajo de investigación se presenta la metodología para identificar rutas lógicas que presentan una mayor influencia de acoplamientos capacitivos con otras interconexiones. La metodología se basa en el algoritmo de Dijkstra, que se utiliza para buscar las rutas más cortas, que en este caso se modificó para buscar las rutas más largas. Por el impacto de los valores de acoplamiento capacitivos que se tiene en las interconexiones de los circuitos, las rutas críticas se consideran las más largas.

Índice general

Capítulo 1

Introducción

1.1. Introducción

El presente documento presenta una propuesta de trabajo, enfocada al perfil de la Ingeniería de *Testing*. Esta área desarrolla estrategias de análisis en circuitos electrónicos VLSI antes del proceso de fabricación con el fin de tener en cuenta todos los parámetros que afectan en el material y proceso.

1.2. Defectos, errores y fallas

En [1] los defectos, errores y fallas se pueden definir de la siguiente forma:

Defecto: En un circuito electrónico un defecto se define como la diferencia no deseada entre el dispositivo fabricado y el diseñado. Entre los defectos más comunes en los circuitos VLSI se encuentran:

- Defectos en el proceso: Ausencia/adición de materiales no esperados, transistores parásitos, rupturas en el óxido.
- Defectos de material: Defectos del cuerpo (imperfecciones en la red cristalina), impurezas en la superficie, etc.
- Defectos por el uso: Rupturas en el dieléctrico, electromigración.
- Defectos del encapsulado: Degradación de contactos, hoyos o aberturas en el sellado.

Error: Se llama así a la señal de salida producida por un sistema con defectos.

Falla: Es la representación de un defecto a un nivel abstracto de operación.

Esta tesis se enfoca en defectos de interconexiones abiertas. Una abertura de rutas de interconexión hace que no haya conducción desde una compuerta.

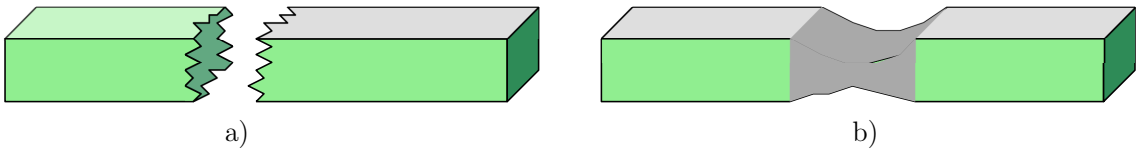


Figura 1.1: Concepto de abertura completa (a) y resistiva (b) en una línea de interconexión.

Débido a la rotura, los transistores pMOS y nMOS de una compuerta, flotan (es decir, quedan aislados o separados). Las interconexiones abiertas, pueden estar abiertas completamente o en estado de resistencia. Una abertura total, Figura 1.1 a), es cuando hay ausencia de material en una sección de interconexión. La distancia entre los puntos desconectados, es lo suficientemente grande para que no exista influencia de la señal de entrada sobre el segmento de línea desconectada. Una abertura resistiva, es cuando el material conductor no está completamente formado, Figura 1.1 b), incrementando la resistencia en la interconexión afectada.

Los contactos, también conocidos como vías son un lugar probable para que ocurra una abertura [2], [3], [4]. Los contactos se han convertido en un importante detractor del rendimiento en las tecnologías modernas que tienen un alto número de contactos debido a los muchos niveles de metal utilizados [2].

1.3. Prueba de circuitos integrados

El objetivo de la prueba (*testing*) de circuitos integrados (CI), es identificar aquellos circuitos fabricados que no satisfacen con las especificaciones iniciales. La figura 1.2 muestra el flujo de pasos de una prueba de un CI, que consiste en los siguientes pasos [5]:

1. Aplicar vectores de prueba a entradas primarias de los circuitos. Los vectores de entrada sensibilizan el defecto y propagan el posible error a un resultado (salida) observable.

2. Se realiza una medición con un resultado/salida observable.

3. El valor medido se compara con un valor de referencia para determinar si el CI se acepta como libre de fallas o se rechaza.

1.4. Modelo de fallas

Para probar los CI's, los defectos físicos se presentan adecuadamente en un nivel superior de abstracción, a esto se le llama Modelo de fallas. El modelo de fallas se puede realizar en diferentes niveles de abstracción como eléctrico, lógico o funcional. Los modelos [6], [5], [7], son definidos para describir el efecto que puede presentar un defecto físico en el desarrollo de un circuito. La

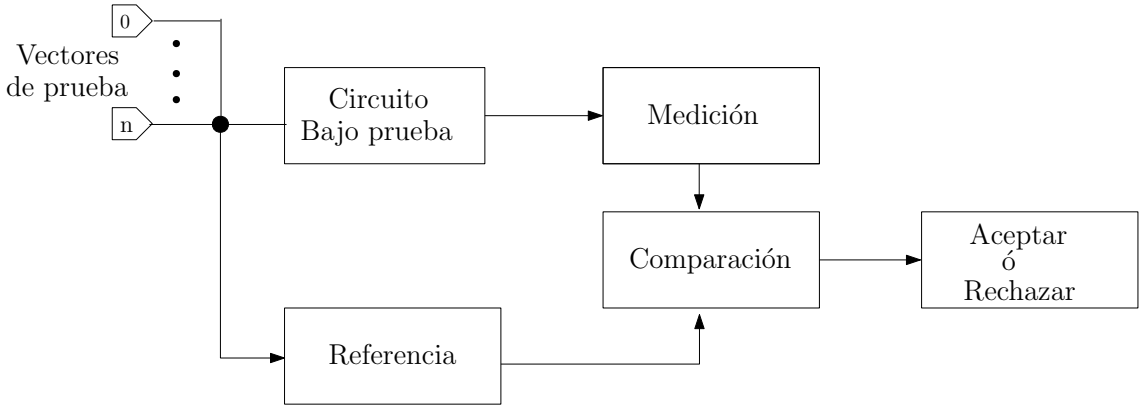


Figura 1.2: Proceso de prueba (*testing*).

generación de patrones de prueba se lleva a cabo en base a un modelo de falla. Un buen modelo de falla puede tener las siguientes propiedades [8], [9]:

- Debe coincidir con el tipo de circuito que se va a analizar.
- La complejidad de las fallas no debe implicar un esfuerzo de cálculo excesivo.
- El modelo de falla debe reflejar el comportamiento de las fallas físicas con suficiente precisión.

1.4.1. Modelo *stuck-at*

El modelo de falla más utilizado es el modelo de falla *stuck-at*. El modelo de falla *stuck-at*, abstrae la implementación y detalles tecnológicos de un circuito, asociando la ocurrencia de fallas directamente en el nivel de compuerta.

El modelo de falla *stuck-at* asume que un nodo defectuoso se comporta como un nodo conectado a uno de los voltajes de suministro, ya sea V_{DD} o *GND* de forma permanente, respectivamente denominados como SA-1 (*stuck-at*-1) y SA-0 (*stuck-at*-0), y se utilizan para describir un nodo que presenta una falla [10]. En el nivel de compuerta, el número de fallas que pueden ocurrir para una compuerta combinacional con *n* entradas y 1 salida es de *2n + 2* [11]. Cada uno de los *n* nodos de entrada puede sufrir fallas SA-0 ó SA-1. Lo mismo ocurre en los nodos de salida. En el modelo de falla *stuck-at*, el conjunto de vectores es aplicado a las entradas primarias del circuito para sensibilizar la falla. El error se propaga a una salida primaria (PO - *Primary Output*). En un circuito, pueden ocurrir varias fallas *stuck-at* simultáneamente. Un circuito con n líneas tendría 3^n - 1 posibles estados *stuck-at*, que es un número alto y computacionalmente costoso. Por lo tanto, es común modelar sólo una falla *stuck-at* al mismo tiempo (no fallas múltiples). De esta forma, un circuito con *n* líneas tiene *2n* fallas *stuck-at*. Este número se reduce aún más por el proceso de compactación de fallas debido a que existen fallas equivalentes.

Algunas de las características de este modelo se pueden resumir en:

- Muchos defectos físicos se pueden modelar mediante la misma lógica.
- La complejidad se reduce enormemente.
- El modelo *stuck-at* es independiente de la tecnología.
- Una prueba (*test*) *stuck-at* simple cubre un gran porcentaje de múltiples *stuck-at*.
- Una prueba (*test*) *stuck-at* simple cubre un gran porcentaje de defectos físicos no modelados.

A pesar de las grandes ventajas del modelo de fallas *stuck-at*, se ha encontrado que este modelo no es adecuado para representar algunos defectos en las tecnologías CMOS [12], [5], [13], [14]. Debido a esto, se han propuesto otros modelos de fallas.

1.4.2. Defectos tipo puente

Se ha demostrado que los defectos tipo puente, son una fuente importante de fallos en VLSI (Very Large Scale Integrated) [15], [16]. Estos tipos de defectos son definidos en [17] como una conexión involuntaria entre dos o más nodos de circuito. Ferguson *et. al.* [18] definieron estos tipos de defectos en conexiones eléctricas no deseadas entre dos o más líneas, resultado de material conductor adicional o falta de material aislante. Para crear la condición de falla de un defecto tipo puente, el vector de prueba debe fijar los nodos en cortocircuito a polaridades lógicas opuestas, el vector de prueba también propaga el valor lógico incorrecto hasta una salida primaria [18], [19].

1.4.3. Aberturas como defectos

Las aberturas o roturas en los circuitos CMOS son fallas difíciles de diagnosticar, al utilizar cualquiera de las técnicas de prueba de dispositivos actuales. Se ha detectado una amplia gama de comportamientos defectuosos. Las aberturas pueden ser causadas por material conductor faltante o por material aislante adicional, por lo que un solo nodo eléctrico se divide en múltiples nodos [18]. Un circuito abierto puede ocurrir en cualquiera de los materiales de interconexión que afecten a la compuerta, drenaje (drain) ó fuente (source). El comportamiento de falla causado por la presencia de una abertura depende de su ubicación, resistencia, valores de capacitancias acopladas parásitas y las fugas de corriente asociadas con el nodo flotante.

En [17] se definen seis clases generales de aberturas. Estas clases son las siguientes:

- Transistor encendido.
- Par de transistores encendidos.
- Par de transistores encendido/apagado.

- Retardo.
- Memoria (transistor apagado).
- Secuencial.

Las primeras cinco categorías de aberturas se presentan en circuitos lógicos combinacionales y en ciertos casos se desarrollan defectos de abertura en circuitos secuenciales.

En general, cuando ocurren aberturas amplias en compuertas, el comportamiento de las fallas difiere dependiendo de su ubicación, resistencia y valores de acoplamientos capacitivos parásitos. Cuando ocurren aberturas que causan que los pares de transistores floten, es probable que un transistor conduzca con *stuck-at* y el otro no [20], [21]. En el caso de que un solo transistor de la compuerta esté flotando, el transistor defectuoso puede estar en *stuck-at*, en este caso la compuerta puede funcionar correctamente pero conmutar a velocidades más lentas [20]. Un transistor flotante de una compuerta también puede ser susceptible a influencias de acoplamientos de metales conductores adyacentes [22], [20], [21]. Sin embargo, si la función lógica se altera, depende de los rangos de ancho y longitud, la topología del circuito y las variaciones de proceso de fabricación [18]. Cuando el ancho de la rotura es lo suficientemente estrecha, la fuga de corriente puede jugar un papel importante en un comportamiento defectuoso [20].

1.5. Tipos de pruebas

1.5.1. Pruebas lógicas

Una prueba lógica [8] se usa para monitorear los niveles lógicos (valores booleanos) de los circuitos bajo prueba. El nodo de salida de un circuito bajo prueba muestra un valor lógico definido para una combinación dada de las entradas. La prueba lógica compara la respuesta del nodo de salida con la respuesta sin falla esperada. Si ambos resultados no son iguales, el circuito bajo prueba se considera defectuoso. En las pruebas lógicas, se asume un tiempo de espera para que los vectores aplicados a las entradas se establezca en niveles estables.

1.5.1.1. Pruebas funcionales

En los primeros años de la tecnología de CI una estrategia de prueba exhaustiva ó completamente funcional se empleó para circuitos integrados de pequeña escala (SSI - *Small Scale Integration*) debido a que la complejidad de circuitos se limitó a compuertas sencillas [23]. Los nodos internos eran de fácil acceso directamente a través de paquetes de pines de IO (*Input-Output*) y la generación del proceso de prueba fue fácil. Sin embargo, el método sólo es aplicable a circuitos pequeños, ya que el tamaño del conjunto de prueba se relaciona exponencialmente con el número de entradas. Para un circuito combinacional con n entradas, un conjunto de prueba exhaustivo consta de 2^n vectores de prueba de entrada [13]. Para un circuito lógico secuencial con m registros de un bit

(elementos de memoria) y una relación de entrada-a-salida, en la que, las salidas dependen tanto de las entradas como de los valores de los registros, un conjunto de prueba exhaustivo constan de 2^{n+m} vectores de prueba.

1.5.1.2. Pruebas estructurales

A medida que los niveles de integración evolucionaron de SSI y MSI a LSI, las pruebas funcionales completas ya no eran posibles, debido al costo de aplicar un conjunto grande de prueba al dispositivo [23]. Un conjunto de prueba cuyo tamaño es lineal al número de nodos en el circuito tendría una clara ventaja sobre la estrategia de prueba funcional completa, en caso de buscar cumplir con los objetivos de la prueba de dispositivo. El costo adicional requerido para encontrar un conjunto de prueba apropiado podría amortizarse durante el tiempo ahorrado al aplicar un reducido conjunto de prueba a cada CI. En consecuencia, para circuitos LSI, el objetivo de la generación de procesos de prueba es determinar el número mínimo de vectores de prueba necesarios para realizar una verificación estructural del CI.

Roth presenta un método simple para derivar un conjunto de prueba que cumpla con este objetivo [24]. En este método, se construye una tabla de verdad para el circuito correcto y para cada uno de los circuitos *p* defectuosos. Un proceso iterativo compara la tabla de verdad correcta con cada una de las tablas de verdad con defectos, el vector de entrada es guardado. Cuando se encuentra una discrepancia entre los valores de salida de las tablas de verdad correcta y con defectos, el vector de entrada se guarda. Cada tabla de verdad con defectos se procesa de esta forma hasta que se encuentre un vector de entrada o las entradas estén agotadas. El resultado de vectores de prueba representan el conjunto de prueba del dispositivo para estas fallas.

Cada tabla de verdad contiene 2^n líneas. Si el circuito contiene *p* nodos, entonces se requerirían *p + 1* tablas de verdad para llevar a cabo el análisis. Claramente, esto tipo de enfoque no se puede utilizar para circuitos grandes. Investigaciones de métodos alternativos fueron considerados en la década de 1960's [24], [25].

1.6. Organización de esta tesis

En el Capítulo 2, se describe el modelo eléctrico, en el que se basó para el desarrollo de simulaciones, análisis de parámetros, así como el planteamiento de la metodología. Los circuitos de prueba utilizados son los ISCAS'85. Los ISCAS'85 son un conjunto de circuitos combinacionales y secuenciales que desde su introducción han sido empleados para evaluar el rendimiento de algoritmos ATPG (*Automatic Test Patron Generation*), así como simulación de fallas, capacidad de prueba de análisis, verificación formal, síntesis lógica y de diseño [26].

En el Capítulo 3, se describe la metodología de trabajo, la que cual inicia con la lectura de archivos ISCAS'85 y de capacitancias de acoplamiento, que a partir de esto, se generará un grafo equivalente, del cual se realizarán diversos análisis utilizando algoritmos de recorrido, como el de la *ruta más corta primero* (Dijkstra) modificado para que busque la ruta más larga, así como el

algoritmo de *búsqueda primero en profundidad*, con recursividad. Se muestra el proceso del trabajo desarrollado en está tesis.

Los resultados se muestran en el Capítulo 4, los cuales abarcan cantidad de segmentos y rutas analizadas. Los tiempos de proceso también son mostrados, considerando varios factores. Esto se da como resultado del proceso aplicado al análisis de los circuitos ISCAS'85, incluyendo sus acoplamientos capacitivos.

Capítulo 2

Defectos en interconexiones

2.1. Circuitos integrados CMOS

El desarrollo de circuitos integrados a muy alta escala (VLSI - *Very Large Scale Integration*) es un campo de la ingeniería que está enfocado en crear circuitos electrónicos de alta complejidad en geometrías reducidas [27].

La industria microelectrónica ha evoluacionado y crecido a gran ritmo durante las últimas décadas, en gran parte debido a la capacidad de integración en los procesos de fabricación de los circuitos de tipo semiconductores de óxido de metal complementario (Complementary Metal Oxide Semiconductor - CMOS). Se ha presentado un aumento considerable en las velocidades de reloj, así como en la reducción en los costos de dispositivos que lo implementan, cuya aplicación está en la mayoría de las áreas de tecnologías [28]. En la Figura 2.1 se puede observar la reducción de las dimensiones de los transistores en el transcurso de los años.

La tecnología CMOS ha evolucionado, tal como se muestra en la Figura 2.1, en dimensiones, así como también en tiempos de conmutación, requerimientos de energía y otros factores, generando un gran desarrollo industrial, esto ha ido impulsando diferentes procesos como el de manufactura, entre otros. Pero este desarrollo ha tenido inconvenientes que se han atendido mediante prácticas de investigación y pruebas debidamente establecidas.

Esta tecnología, de manera básica opera con transistores de efecto de campo pMOS y nMOS. Se deriva de la familia MOS, pero CMOS se diferencia de la anterior, dado que utiliza dos tipos de transistores en su circuito de salida, se usan conjuntamente un MOSFET (MOS Field-Effect Transistor, transistor de efecto campo MOS) de canal *n* (nMOS) y de canal *p* (pMOS) en el mismo circuito, para obtener varias ventajas sobre las familias pMOS y nMOS [29], [30]. La tecnología CMOS es ahora la dominante debido a que es más rápida y consume aún menos potencia que otras familias MOS.

Un transistor del tipo nMOS consta de cuatro terminales: *gate* (compuerta), *source* (fuente), *drain* (drenaje) y *bulk* (sustrato) [31]. La Figura 2.2, muestra la representación de un transistor nMOS y un transistor pMOS, cada terminal está indicada con las letras G (*gate*), S (*source*), D

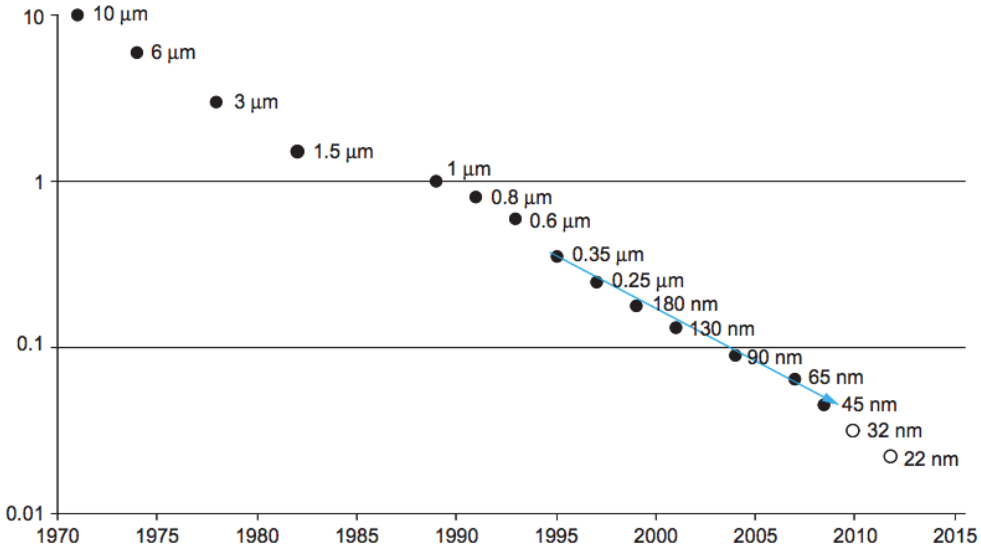


Figura 2.1: Dimensiones de transistor CMOS en el trancurso de los años [29].

(*drain*) y B (*bulk*).

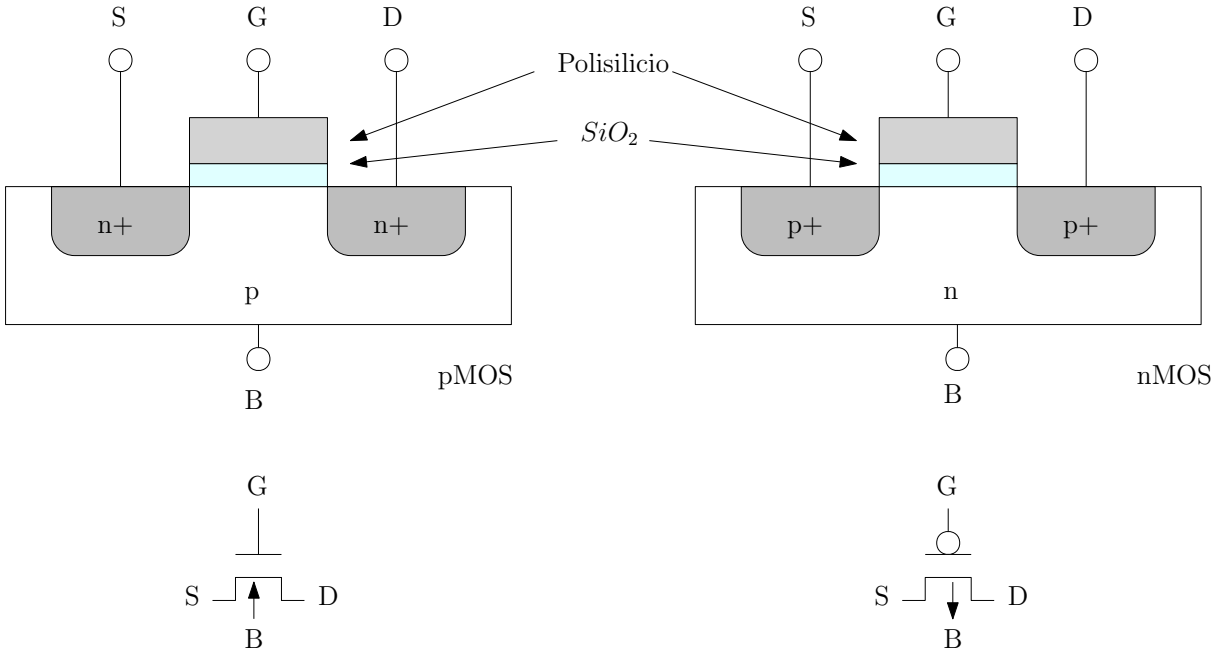


Figura 2.2: Vista transversal de transistores pMOS y nMOS [29].

Hoy en día, por confiabilidad y razones legales, los fabricantes presentan datos y modelos de producción y prueba del peor de los casos para sus productos, incluido un período de envejeci-

miento de 10 o 15 años y condiciones extremas de (85 °C para dispositivos comerciales y 100 °C para dispositivos industriales, carga de trabajo más baja y fuente de alimentación más baja o V_{DD} al final) [32].

2.2. Proceso de *testing* en VLSI

El proceso de fabricación de circuitos VLSI es complejo, integra una serie de pasos mecánicos y químicos que pueden presentar un grado de inexactitud, además de imperfecciones, en los chips o circuitos integrados, por lo que es posible que no funcionen de acuerdo a lo especificado en el diseño inicial del mismo [33]. Lo anterior hace que se deban realizar pruebas que garanticen la funcionalidad de los circuitos integrados, que serán comercializados.

El *testing* es un proceso implementado para identificar imperfecciones o defectos de fabricación en los circuitos integrados. El proceso de *testing* puede considerar 3 características negativas: defecto, error y fallas. Un defecto se describe como una anomalía no intencionada entre el circuito que se está fabricando y el diseño inicial de éste. Los defectos se pueden presentar por falta de contactos, impurezas, cortocircuito en el óxido de una compuerta, abertura o puenteo de metales, degradación de contacto, entre otros [33]. Bajo ciertas circunstancias un dispositivo defectuoso puede generar resultados diferentes a los esperados.

2.3. Impacto de acoplamientos

2.3.1. Acoplamientos en tecnología CMOS

La tecnología CMOS puede presentar defectos en su proceso de fabricación, que consiste en características no deseadas, como variabilidad en densidades, en el grosor del óxido de la compuerta, o profundidad en uniones que impactan en el comportamiento del circuito [28], como por ejemplo retraso en la conmutación de la señal, así como fugas.

Los circuitos integrados modernos están compuestos de estructuras de interconexión complejas, que incluyen algunas capas de metal. La cercanía de estas interconexiones pueden hacer que en ocasiones que se presente un efecto de acoplamiento, que se refleja en señales falsas [34].

La capacitancia (C) es un parámetro que describe la capacidad de almacenamiento de carga de un dispositivo. En un circuito electrónico se presentan diferentes valores de capacitancia.

La capacitancia de una línea de segmento, es decir, la interconexión entre cada compuerta, se compone por la capacitancias a *GND*, V_{DD} y la suma de las capacitancias de compuerta conectadas a la línea del segmento.

Las compuertas basadas en tecnología CMOS presentan capacitancias intrínsecas parásitas (capacitancias de traslape) en cada transistor MOSFET, que deben ser consideradas en su diseño: capacitancia de *gate* a *source* (C_{gso}), capacitancia de *gate* a *drain* (C_{gdo}), capacitancia de *drain* a *bulk* (C_{dbo}), y capacitancia de *source* a *bulk* (C_{sbo}), la Figura 2.3, muestra el esquema de un inversor

involucrando capacitancias de compuertas, las capacitancias de entrada (C_{in}) y salida (C_{out}) están dadas por:

$$C_{in} = C_{gsop} + C_{gdop} + C_{gson} + C_{gdon} \tag{2.1}$$
$$C_{out} = C_{dbop} + C_{sbop} + C_{dbon} + C_{sbon} \tag{2.2}$$

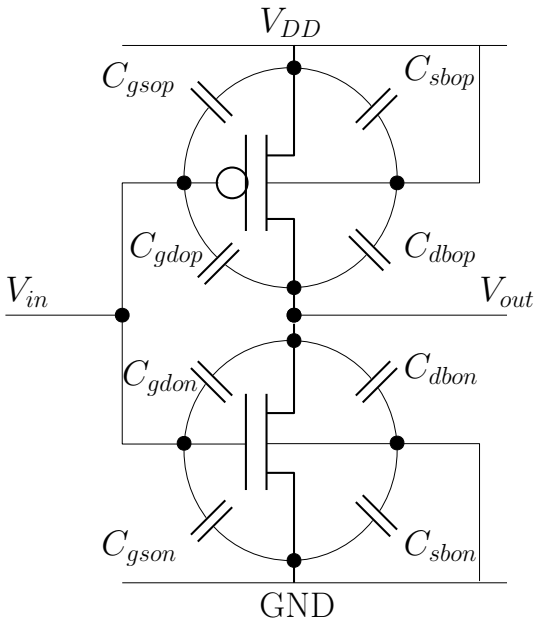


Figura 2.3: Capacitancia de compuerta: Inversor.

La Figura 2.4 muestra una estructura de interconexión, la cual puede estar acoplada por señales que viajan en otras líneas de interconexión. Las líneas en el metal 1 presentan un acoplamiento adyacente (C_C; M1-M2), una capacitancia de acoplamiento interfiriendo se puede observar entre los metales 1 y 2 (C_C; M1-M2), además, el metal 1 presenta una capacitancia a los canales de tierra y alimentación (C_{r_0} y C_{r_1}) [35].

Un acoplamiento capacitivo se presenta cuando una línea de interconexión es influenciada por señales que se transmiten en líneas de acoplamiento adyacentes, ya sea que estén ubicadas encima ó debajo de ésta, va a estar presente en circuitos VLSI debido a la cercanía de las líneas de interconexión. Su impacto depende de la cantidad de capacitancia acoplada en una línea. La interferencia entre la líneas de interconexión se le conoce como *crosstalk*, lo cual impacta en el desempeño del circuito.

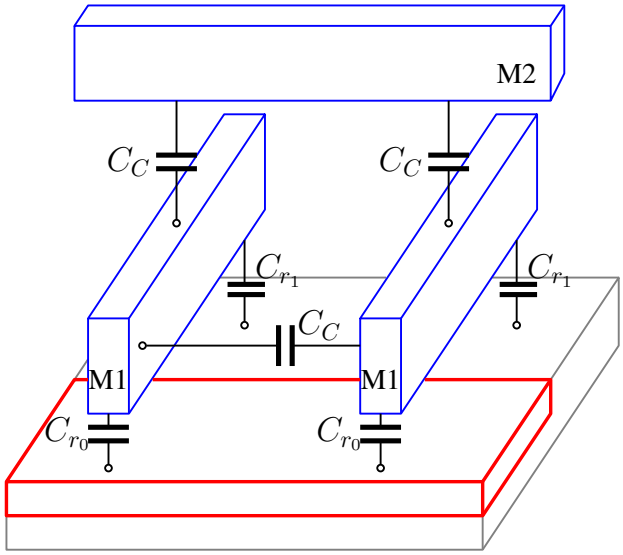


Figura 2.4: Acoplamientos capacitivos entre líneas de conexión.

2.3.2. Acoplamientos capacitivos

Una señal puede ser afectada de manera importante por efectos de *crosstalk* influenciados por acoplamientos capacitivos que se encuentran entre conexiones cercanas. Los efectos de *crosstalk* influenciados por acoplamientos capacitivos entre una conexión, a la cual se le llama víctima e interconexiones adyacentes, llamadas agresoras, pueden aumentar o disminuir los retardos entre segmentos de víctimas y agresores, dependiendo de la transición, tiempo de llegada y acoplamientos capacitivos entre las líneas víctima y agresor(es) [33], [34], [35], [36].

El aumento en frecuencia y la disminución de los tiempos de transición de las señales en diseños actuales provoca más actividad de conmutación simultanea en intervalos de tiempo muy cortos, causando un incremento en la densidad de corriente y caída de voltaje a través de las líneas de interconexión [33], [34], [35], [36].

Debido al escalamiento en la tecnología y el aumento de la velocidad de conmutación, el problema de los retrasos en el modelado de compuertas se vuelve aún más difícil. En diseños VLSI, el factor del tiempo de retardo que contribuyen las compuertas se reduce, mientras que el retardo de interconexión se vuelve dominante. Esto se debe al hecho de que el escalado de las longitudes de las interconexiones no es proporcional a la reducción del tamaño de los transistores que forman las compuertas [37], [33], [34], [35], [36].

Hay dos tipos principales de efectos de *crosstalk* [37]: pulsos inducidos y retardo inducido. Dado un par de nodos que involucran acoplamientos capacitivos, siempre hay uno que actúa como nodo agresor y el otro como nodo afectado o víctima. Una afectación de *crosstalk* es por un pulso inducido por efectos de acoplamiento entre líneas interconectadas. La amplitud y el ancho de la falla dependen del tiempo de conmutación relativo de los agresores, la cantidad de capacitancia de acoplamiento, capacitancia de línea a tierra y los tiempos de transición relativos de los agresores.

Cuando a los nodos agresores se les aplica una señal de transición y las señales estables se aplican al nodo víctima, la señal estable puede experimentar ruido de acoplamiento debido a la transición en los nodos agresores [37].

El *crosstalk* puede causar efectos no deseados, como pueden ser fallas funcionales o cambios en el tiempo de propagación de señal, es decir retardo [38]. El efecto de falla es causado por una señal constante de una víctima debido al acoplamiento por la actividad de conmutación de los agresores vecinos.

Si las líneas de interconexión víctima y agresor están conmutando hacia el mismo estado lógico, entonces el retardo en la línea víctima será menor, a esto se le conoce como *crosstalk* negativo. Si la interconexión del agresor conmuta en dirección opuesta a la línea de la víctima, el aumento en el retardo se presenta en la víctima, esto es conocido como *crosstalk* positivo.

El retardo por *crosstalk*, es un retardo de señal inducido por efectos de acoplamiento que se presentan en transiciones simultáneas entre líneas de interconexión. El *crosstalk* provoca un retraso además del retraso normal de la compuerta y el propio retraso de interconexión. Por lo tanto, es difícil estimar el tiempo de retardo correcto del circuito, lo que puede provocar graves problemas de retardo de la señal. Estos cambios inesperados en los retardos de propagación hacen que el retardo de las rutas en un chip sea mayor de lo esperado, lo que hace que la salida de un chip cambie el comportamiento esperado, a pesar de que el chip sea funcionalmente correcto [37].

El acoplamiento capacitivo puede causar retardo en una señal que viaja a través de las líneas de interconexión de un circuito [39]. En circuitos VLSI las interconexiones presentan acoplamiento a V_{DD}, *GND* y otras líneas que afectan el comportamiento, incrementando, disminuyendo o retrasando el voltaje. La Figura 2.5 muestra un modelo de cómo interviene la capacitancia de acoplamiento.

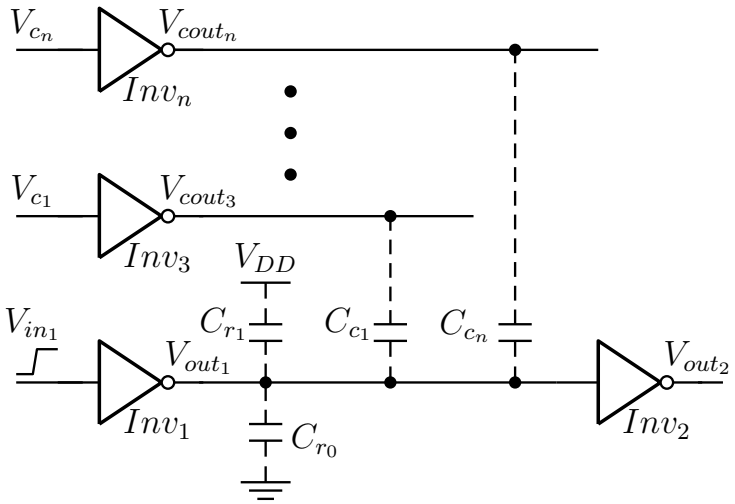


Figura 2.5: Modelo de acoplamiento capacitivo.

En la Figura 2.5 se observa un esquema de inversores, representados por Inv_1 e Inv_2, afectados por varias capacitancias de acoplamiento, las cuales provienen de las salidas de inversores Inv_3,

hasta Inv_n. Para las simulaciones se tomó como base un modelo de tecnología de 65 nm. El esquema se simula para ilustrar el impacto del acoplamiento capacitivo en el retardo del circuito. Se aplican señales de transición en $V_{in_1} \uparrow (0 \rightarrow 1)$ V_{c_1}, y $V_{c_n} \downarrow (1 \rightarrow 0)$; y se mide la respuesta en la salida V_{out_2} del inversor Inv_2. Se puede observar que el retardo de los inversores aumenta debido a la presencia de la capacitancia de acoplamiento (C_C). Por lo tanto, las capacitancias de acoplamiento deben tenerse en cuenta de manera adecuada, ya que pueden afectar significativamente la integridad de la señal y también juegan un papel esencial en la prueba de integración de circuitos.

Para poder comprobar este fenómeno, se realizaron simulaciones en HSPICE, y se verificó el impacto en un circuito, principalmente en el retardo. Un ejemplo se puede observar en la Figura 2.6, los inversores son simulados con acoplamientos capacitivos y sin éstos también.

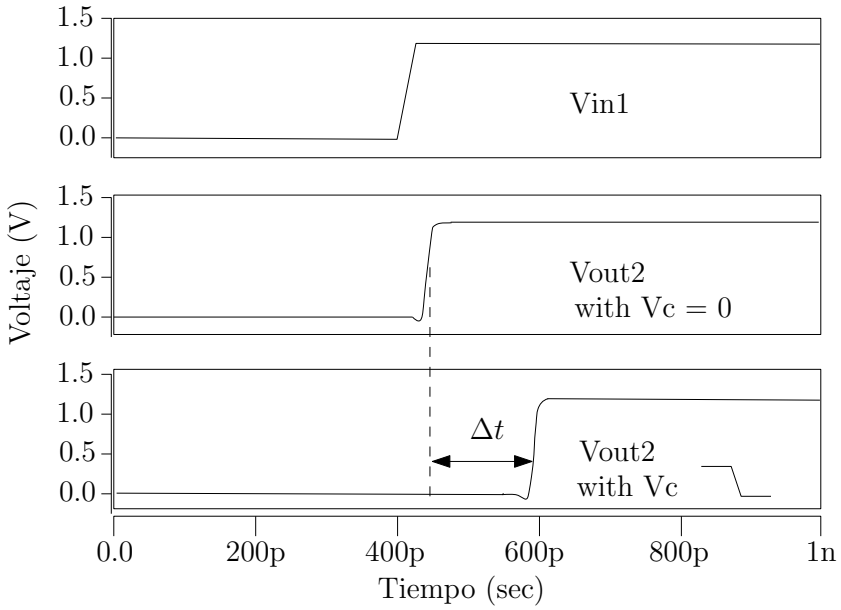


Figura 2.6: Simulación de efectos de acoplamiento capacitivo entre dos líneas de conexión basado en el circuito de la Figura 2.5.

Para asociar el impacto del acoplamiento capacitivo, se establece un factor de selección f, que define la sensibilidad de una línea de segmento a un acoplamiento capacitivo. A f se le pueden asignar valores como 0.2, 0.3, ..., 0.8, 0.9.

Al asignar un valor bajo al factor de selección (f), como 0.2, 0.3, las capacitancias acopladas con valores bajos condicionan a una línea de segmento a ser considerada como crítica, por lo tanto, un conjunto grande de segmentos será considerado como crítico.

Una línea de segmento crítica en un circuito, implica que al menos presenta un acoplamiento capacitvo, representado por C_{c_i}, mayor o igual que el total de la capacitancia de la línea del segmento, representada por C_{CT}, multiplicada por un factor de selección (f), como lo muestra la ecuación siguiente:

$$C_{c_i} \geq f \times C_{CT} \tag{2.3}$$

donde

$$C_{CT} = C_{r_0} + C_{r_1} + C_{g1} + C_{g2} + \cdots + C_{gn} \tag{2.4}$$

La capacitancia total de una línea de segmento (C_{CT}), expresada en la Ecuación (2.4), está compuesta por la capacitancia a *GND* (C_{r0}), V_{DD} (C_{r1}), así como la suma de todas las capacitancias de las compuertas C_{g1}, C_{g2}, hasta C_{gn}, conectadas a la línea del segmento.

Las líneas de interconexión con un acoplamiento significativo, son suceptibles a degradar el desempeño de un circuito y eso puede impactar en la eficacia de las pruebas (*testing*) de los circuitos integrados, debido a esto, sólo las líneas susceptibles a ser consideradas críticas por señales de acoplamiento van ser consideradas, éstas son llamadas *líneas de segmento críticas*. Una línea que interconecta dos compuertas, se denomina línea del segmento, ésta puede ser influenciada por señales adyacentes de acomplamientos capacitivos. Una línea de segmento crítico se define como aquella que presenta al menos una señal de acomplamiento capacitivo representado por C_{C_i}. En la Figura 2.7, se puede identificar la representación del conjunto de C_{C_i}, en C_{C_1},..., C_{C_4}.

En la Figura 2.7, se puede identificar la representación del conjunto de C_{C_i}, en C_{C_1},..., C_{C_4} del Segmento 3.

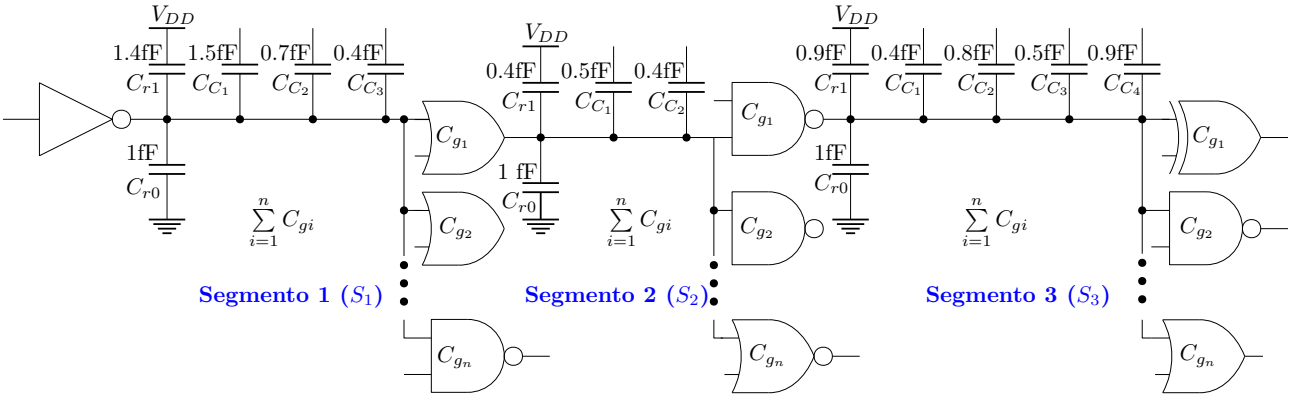


Figura 2.7: Ejemplo de circuito para ilustrar el impacto del factor de selección

Las líneas de segmento crítico se determinan utilizando la Ecuación (2.3). Este valor ayuda a determinar si el segmento es crítico, si se cumple la condición establecida en la ecuación, esto contrubuirá al valor del peso del segmento. El valor de peso que será asignado es aquel que cumple la condición, de otra manera el valor asignado al segmento será 0. La Tabla 2.1 y la Figura 2.7 describen el análisis realizado en la ecuación.

En la Tabla 2.1 se puede observar en la primera columna los segmentos S_1, S_2 y S_3 correspondientes a la Figura 2.7, con sus correspondientes acomplamientos capacitivos en las columnas

Tabla 2.1: Identificación de segmentos críticos.

Segmento	(C_{CT}) (fF)	$f \cdot (C_{CT})$ (fF)	Acoplamientos Capacitivos C_{C_1}	C_{C_2}	C_{C_3}	C_{C_4}
Factor de selección *f = 0.2*						
S_1	2.5	0.5	1.5	0.7	0.4	
S_2	1.5	0.3	0.5	0.4		
S_3	2	0.4	0.4	0.8	0.5	0.9
Factor de selección *f = 0.4*						
S_1	2.5	1	1.5	0.7	0.4	
S_2	1.5	0.6	0.5	0.4		
S_3	2	0.8	0.4	0.8	0.5	0.9

C_{C_1}, ..., C_{C_4}. En la segunda columna se encuentra la capacitancia total, mientras que en la tercera columna, se muestra el resultado de multiplicar el factor de selección por la capacitancia total. En una primera sección se presenta el análisis para un factor de selección de 0.2, en la que se puede observar, en S_1, tiene un acomplamiento capacitivo en C_{C3} de 0.4, en base a la Ecuación (2.3), no cumple la condición, es decir, no es mayor a 0.5, pero dado que en C_{C_2} y C_{C_1} cumplen con la condición de la ecuación, este segmento se considera crítico.

En una segunda sección de la Tabla 2.1, se pueda ver un análisis para un factor de selección de 0.4, que para S_1, en la tercera columna $f \cdot (C_{CT})$, tiene un valor de 1, en el caso de C_{C_1}, sólo cumple la condición establecida por la Ecuación (2.3), esto hace que este segmento se considere como crítico. Pero analizando el caso de S_2, que tiene un valor de 0.6 en $f \cdot (C_{CT})$, ningún acoplamiento capacitivo cumple la condición, ya que C_{C_1} es 0.5 y C_{C_2} es 0.4, por lo que este segmento no se considera crítico.

Capítulo 3

Algoritmos y metodología de detección de fallas

La metodología propuesta consiste en buscar y determinar rutas que pueden llegar a ser consideradas como críticas debido al número de acoplamientos capacitivos que presentan. El primer paso consiste en representar el circuito bajo prueba en un grafo ponderado. El valor de los acoplamientos capacitivos entre cada interconexión está asociado al peso o costo a considerar en la ponderación. Estos grafos son analizados para obtener rutas según especificaciones que podamos plantear, por algoritmos de recorridos como Dijkstra y búsqueda primero en profundidad (*Depth First Search* - DFS). Los circuitos de prueba tomados en cuenta son los ISCAS'85. A partir de los *netlist* de cada circuito se obtienen las conexiones de cada grafo, que se forman a partir de las entradas, compuertas y salidas. También se consideran archivos con acoplamientos capacitivos, para asignar un peso a cada conexión.

Un grafo se compone de un conjunto de vértices o nodos (n), que están interconectados mediante aristas o segmentos, que forman un par de nodos (origen, destino) [40] que pueden tener peso (w) a este tipo de grafos se le conoce como grafo ponderado.

El diagrama de flujo de la metodología se muestra en la Figura 3.1. Las entradas al método propuesto son una lista de conexiones que describen el circuito en Verilog y un archivo que contiene las capacitancias extraídas de las líneas de interconexión con herramientas CAD (Cadence). A partir del diseño del circuito, se extrajeron las capacitancias a tierra y la fuente de alimentación de cada línea y las capacitancias de interconexión.

Primero, se utiliza un algoritmo de búsqueda de ruta para encontrar todas las rutas topológicas del circuito. Luego, se analiza la severidad de las capacitancias acopladas a cada línea. El próposito de este paso es descartar rutas que no impactan en este tipo de análisis, que en este caso son las consideradas cortas o con pocos nodos interconectados. Una vez hecho esto, queda un grafo con menos rutas para que un algoritmo de búsqueda de rutas críticas como Dijkstra sea más efectivo.

Se puede observar en la Figura 3.1, que como entrada, se requiere de dos archivos, el *netlist* y su respectivo archivo con capacitancias de acoplamiento. De éstos se obtiene un grafo ponderado para

obtener rutas topológicas y poder realizar un primer análisis de las líneas de cada segmento, con el fin de determinar qué segmentos serán considerados en el grafo final que analizará el algoritmo de Dijkstra.

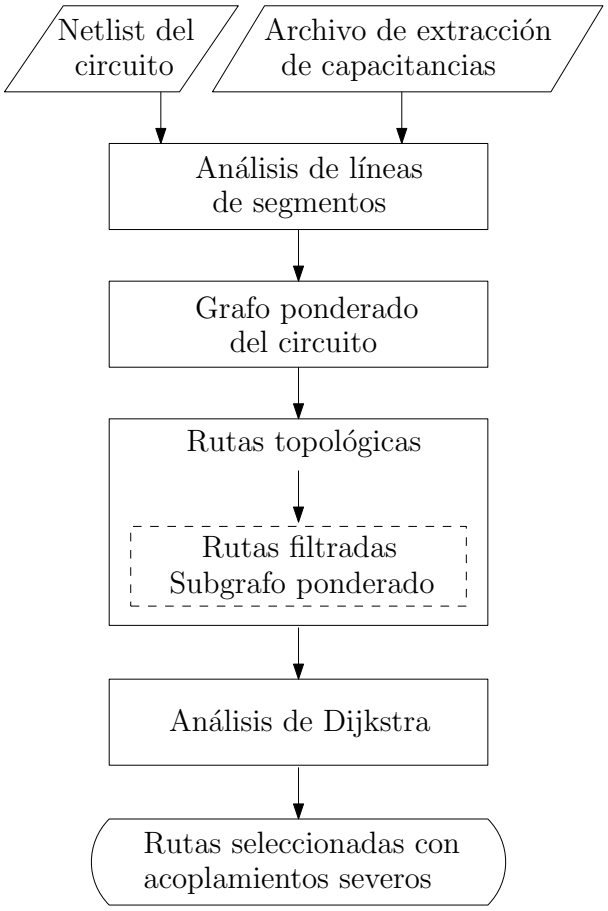


Figura 3.1: Flujo de proceso para detectar rutas lógicas críticas

El objetivo es identificar y seleccionar aquellas rutas afectadas, que debido a los acoplamientos severos son consideradas como críticas, haciendo uso de un algoritmo para recorridos de grafos. Un circuito electrónico es convertido a grafo, tomando como referencia un *netlist*, programado en lenguaje Verilog. Un segmento considerado como crítico en un circuito implica que al menos contiene una línea acoplada capacitivamente, representado por C_{c_i}. El valor en Faradios (F) de la capacitancia C_{c_i} deberá cumplir la condición descrita en la Ecuación (2.3).

Una ruta crítica en un circuito electrónico puede convertirse en la ruta más lenta durante su ciclo de vida. Una combinación de factores pueden hacer cumplir la condición para que una ruta sea considerada como crítica [41]. Tal impacto puede producir desgaste en la línea del circuito. Una selección de ruta crítica eficiente debe identificar sólo aquellas rutas que pueden convertirse

en la ruta más lenta del circuito a lo largo del periodo de vida del circuito, ya que estas rutas fallan primero que otras.

Al factor de selección f, se le puede asignar valores como *0.2, 0.3, ..., 0.8, 0.9*, que define la sensibilidad de una línea de segmento a un acoplamiento capacitivo. Cuando se asigna un valor bajo al factor de selección (f), las capacitancias acopladas con valores bajos condicionan a una línea de segmento a ser considerada como crítica, por lo tanto un conjunto grande de segmentos será considerado como crítico.

3.1. Circuitos de prueba

3.1.1. *Netlist* de circuitos ISCAS'85

En un inicio, como se muestra en la Figura 3.1, el proceso requiere conocer las conexiones del circuito. Como base se tomaron los circuitos ISCAS'85, que son un conjunto de circuitos combinacionales y secuenciales, empleados como circuitos de prueba estándar para investigaciones y desarrollos de VLSI [26]. En la Tabla 3.1, se muestran los circuitos combinacionales utilizados en la primera columna; en la segunda columna se lista el número de compuertas que contiene; la tercera y cuarta columna muestran el número de entradas, así como salidas, respectivamente, en la última columna hace una descripción breve o función del circuito.

Tabla 3.1: Lista de circuitos ISCAS'85 analizados en este trabajo [26].

Circuito	Compuertas	Entradas	Salidas	Descripción
C17	6	5	2	Circuito de prueba básico
C432	160	37	7	Controlador de interruptor de canal
C499	202	41	32	Corrector de error simple
C1908	880	33	25	Corrector de error simple/doble
C1355	546	41	32	Corrector de error simple
C3540	1669	50	22	ALU
C5315	2406	178	123	ALU
C7552	3512	207	108	Sumador/comparador

En la Figura 3.2 se puede observar un diagrama de bloques de un circuito ISCAS'85 utilizado, en este caso el C432, como se describe en la Tabla 3.1, el cual es un controlador de interruptor, específicamente de 27 canales agrupados en 3 grupos de bus de 9 bits (A, B y C). Se compone de 5 módulos M1, M2, M3, M4 y M5.

El primer circuito ISCAS'85 analizado, es el C17, que es sencillo, pero tiene un grado de complejidad al contar con varias entradas que presentan varias rutas con diferentes salida. El código

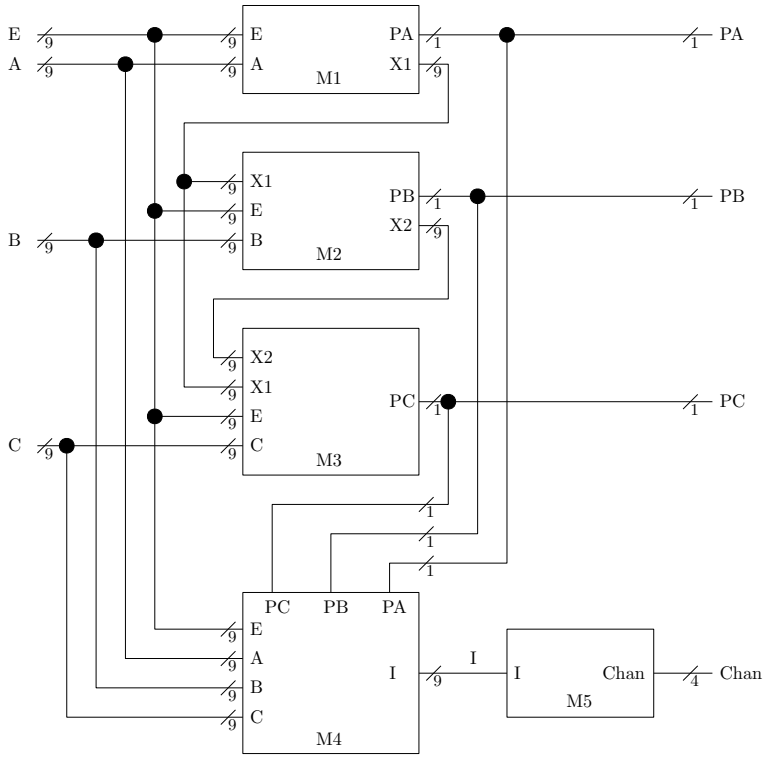


Figura 3.2: Diagrama de circuito ISCAS'85 C432.

en Verilog es mostrado en el Listado 3.1, donde observan las entradas (*inputs*), salidas (*outputs*), conexiones (*wires*) y las compuertas, especificando el tipo, además de la red del circuito o interconexiones:

Listado 3.1: *Netlist* del circuito ISCAS'85 C17 en Verilog

```
1  module c17 ( N1, N2, N3, N6, N7, N22, N23 ) ;
2
3      input N1 ;
4      input N2 ;
5      input N3 ;
6      input N6 ;
7      input N7 ;
8      output N22 ;
9      output N23 ;
10
11     wire nx88, nx90, nx92, nx96 ;
12
13     nand02 ix87 (.Y (N23), .A0 (nx88), .A1 (nx92)) ;
14     nand02 ix89 (.Y (nx88), .A0 (N2), .A1 (nx90)) ;
```

```
15      nand02 ix91 (.Y (nx90), .A0 (N3), .A1 (N6)) ;
16      nand02 ix93 (.Y (nx92), .A0 (nx90), .A1 (N7)) ;
17      nand02 ix95 (.Y (N22), .A0 (nx96), .A1 (nx88)) ;
18      nand02 ix97 (.Y (nx96), .A0 (N1), .A1 (N3)) ;
19
20  endmodule
```

El código del Listado 3.1 muestra la lista de conexiones del circuito integrado, donde se describen todas las compuertas de cada módulo indicando su entrada y su salida. Se requiere identificar cada entrada, salida, conexión, compuerta, así como las interconexiones entre compuertas. Por esta razón a partir del archivo original en Verilog se generaron 3 archivos: 1) Compuertas, 2) Entradas y 3) Salidas, con el fin de analizar de manera detallada el circuito y poder generar un grafo con la información requerida, además de organizada. Este proceso es ilustrado en la Figura 3.3.

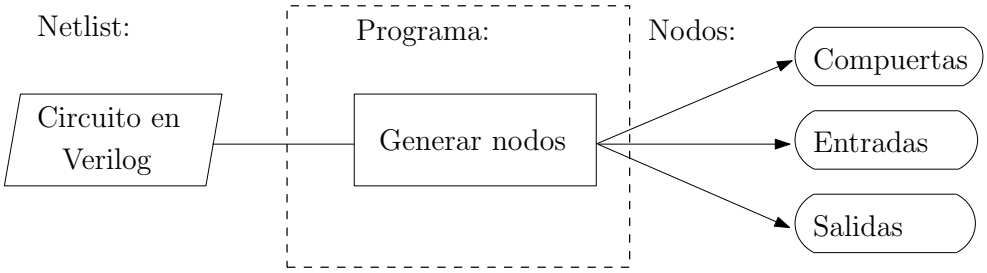


Figura 3.3: Generación de nodos con base en la *Netlist* del circuito

En los siguientes listados se puede observar cómo quedó cada archivo por separado. Inicialmente el Listado 3.2 muestra el archivo con las compuertas que conforman el circuito ISCAS'85 C17, en el cual se puede observar al inicio de cada renglón el identificador de cada una, es decir se cambió el formato del renglon respecto al archivo en Verilog de las interconexiones, con el fin de facilitar la creación del grafo representativo del circuito. Este proceso se llevó a cabo con la implementación de un programa en C++, el cual solicita el nombre del archivo en Verilog e inicia un proceso de análsis de líneas para depurar palabras o caracteres no útiles, e identificar las líneas del archivo que necesitamos para nuestros fines.

Listado 3.2: Compuertas del circuito ISCAS'85 C17 en Verilog

```
1 ix87 nx88 nx92 N23 NAND02
2 ix89 N2 nx90 nx88 NAND02
3 ix91 N3 N6 nx90 NAND02
4 ix93 nx90 N7 nx92 NAND02
5 ix95 nx96 nx88 N22 NAND02
6 ix97 N1 N3 nx96 NAND02
```

Para facilitar la creación del grafo, la sintaxis del renglón que describe las conexiones de cada compuerta (id, entradas, salida, tipo), consiste en colocar al final el tipo, ya que un dato que varía es la cantidad de entradas, con este esquema, facilita su identificación. En cuanto a la sección correspondiente a entradas y salidas no se modificó la sintaxis, por lo que sólo se hizo la identificación y se hizo una copia de cada línea correspodiente, con el fin de estandarizar, identifcar y facilitar la representación del circuito como un grafo. El archivo de entradas se muestra en el Listado 3.3.

Listado 3.3: Entradas del circuito ISCAS'85 C17 en Verilog

```
1 input N1
2 input N2
3 input N3
4 input N6
5 input N7
```

En el Listado 3.4, se muestra el contenido del archivo de salidas obtenidas del circuito ISCAS'85 C17, se puede observar, que tanto el Listado 3.3, así como 3.4, son archivos más simples, ya que cuentan con el tipo de terminal, y su identificador. El archivo original se mantiene, y los nuevos archivos generados son debidamente identificados, y éstos son los que sirven de entrada para cualquier proceso de este trabajo, ya que prácticamente se consideran los nodos de un grafo que representa el circuito.

Listado 3.4: Archivo de salidas del circuito ISCAS'85 C17 en Verilog

```
1 output N22
2 output N23
```

El objetivo de separar el archivo en Verilog, es detectar los nodos que formarán un grafo que represente un circuito electrónico. Analizando una *netlist* en Verilog, se dedujo que éste se forma a partir de conexiones que tiene cada compuerta, por lo que el primer archivo del que se obtienen datos es de compuertas, cuya identificación de la compuerta en turno se encuentra al principio de cada línea, de ahí el archivo de entradas es procesado y finalmente el de salidas.

En el programa desarrollado en C++, el grafo se compone por nodos que constan de cada compuerta, entrada y salida, todas con su identificador. Ya que se tiene la base de los nodos que formarán el grafo que se va a analizar, es posible agregar los valores que tendrá cada segmento de conexión entre cada nodo que se forma a partir de capacitancias.

3.1.2. Archivo de extracción de capacitancias

En la Figura 2.7, que muestra el impacto de la elección del factor de selección, también se describe la Ecuación (2.3), que se aplica a cada una de las líneas de segmento, que presentan afectaciones por capacitancias de acoplamiento. Si se presentan valores de capacitancias de acoplamiento que satisfacen la Ecuación (2.3), esa línea de segmento hará que la ruta a la que pertenece, sea considerada potencialmente como crítica.

Los valores más bajos del factor de selección favorecen la mejora de la cobertura de los defectos de la interconexión y también mejoran el rendimiento del circuito, pero a la vez, este proceso aumenta el tiempo de cálculo. Debido a esto, existe una compensación entre el costo del tiempo de cálculo y la mejora de la cobertura de defectos por rendimiento del circuito.

En cuanto a los valores de las capacitancias de acoplamiento que se encuentran en cada línea de segmento, están contenidos en un archivo en el Listado 3.5, conocido como archivo de extracción de capacitancias parásitas, que tiene el siguiente formato (ver Listado 3.5):

Listado 3.5: Archivo de capacitancias del circuito ISCAS'85 C17

```
1 cc\_1 N7 N23 0.0290312e-15
2 cc\_2 GND N23 1.88438e-15
3 .
4 .
5 .
6 cc\_55 nx90 nx92 0.662803e-15
7 cc\_56 nx96 nx90 0.015656e-15
```

Para generar un grafo ponderado, una vez que se tienen los nodos que lo componen, es decir, se pueden identificar los segmentos, se le van asociando cada línea acoplada que formará el peso de ese respectivo segmento.

Las líneas de interconexión con un acoplamiento significativo, son suceptibles a degradar el desempeño de un circuito y eso puede impactar en la eficacia de las pruebas ($testing$) de los circuitos integrados. Debido a esto, sólo las líneas susceptibles a ser consideradas críticas por señales de acoplamiento van a ser consideradas, éstas son llamadas *líneas de segmento críticas*. Una línea que interconecta dos compuertas, se denomina línea del segmento, y ésta puede ser influenciada por señales adyacentes de acomplamientos capacitivos. Una línea de segmento crítico se define como aquella que presenta al menos una señal de acomplamiento capacitivo representado por C_{C_i}.

3.1.3. Análisis de líneas de segmentos

El impacto relativo de las señales acopladas sobre una línea de segmento se obtiene usando la Ecuación (3.1), se asume que todas las señales acopladas en la misma, afectan. El peso (w) de una línea de segmento considera los siguientes valores de capacitancia:

- Acoplamientos en las líneas de interconexión (C_{c_i}).
- Capacitancia total (C_{CT}, Ecuación (2.4)), que se compone de:
  - Capacitancia a V_{DD}.
  - Capacitancia a GND.
  - El valor capacitivo de las compuertas asociadas a la línea de salida.

$$w = \sum_{i=1}^{n} \frac{C_{c_i}}{C_{CT} + \sum_{i=1}^{n} C_{c_i}} \tag{3.1}$$

Un grafo ponderado del circuito es construído usando los pesos de las líneas de segmento, tal como se muestra en la Figura 3.4, basado en el circuito ISCAS'85 C17.

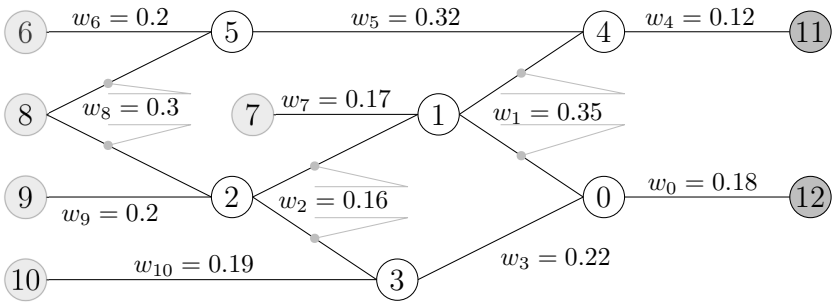


Figura 3.4: Grafo ponderado, basado en el circuito ISCAS'85 C17.

Aplicando un algoritmo para el recorrido de grafos, podemos obtener todas las posibles rutas desde cualquier nodo de entrada, también se pueden elegir rutas críticas basándose en algún criterio. Con esta información podemos elegir rutas que por su magnitud pueden ser críticas y otras que no son relevantes para el análisis. En esta etapa se procedió a descartar rutas que son demasiado cortas, es decir con pocos segmentos. Como resultado se obtuvo un grafo con menos segmentos. El impacto de esto se refleja en los tiempos de cómputo.

3.1.4. Rutas topológicas

Una vez integrada la información del *netlist* del circuito y sus acoplamientos capacitivos se genera un grafo ponderado. Representar un circuito mediante un grafo permite aplicar algoritmos de recorrido o de busqueda de rutas, cortas o largas. El grafo se forma a partir de parejas de nodos origen - destino, y un valor asociado a esa relación, denominado peso. En este caso, los nodos del grafo obtenidos de un circuito electrónico son: entradas, compuertas y salidas. El peso de cada conexión del grafo es el valor de los acoplamientos capacitivos explicados en la sección anterior (Ecuación 3.1). La eficiencia de los algoritmos pueden ser valorada por su complejidad, que está asociada a la cantidad teórica de tiempo de CPU que consume el algoritmo, así como los ciclos involucrados, con sus instrucciones.

El orden de complejidad de un algoritmo (O), se relaciona con las instrucciones necesarias que ocupa para resolver un problema, ya sea operaciones elementales de tipo de aritméticas, comparativas, lógicas entre otras, que son ejecutadas en un tiempo dado [42]. Dado que se tienen variables

como el tiempo, número de datos, así como el tipo de datos, la complejidad en el sentido del tiempo se representa como la función T(n), que representa el número de unidades de tiempo tomadas por un programa o algoritmo para cualquier entrada de tamaño n, que es el número de elementos que son procesados. Por ejemplo un algoritmo de complejidad líneal O(n), conforme aumenta el número de datos, aumenta en forma proporcional el número de operaciones. Un algoritmo de complejidad O(n^2), conforme aumenta el número de datos, el número de operaciones crece en forma exponencial, esto es porque este tipo de algoritmos tiene ciclos anidados, O(n^2) tiene 2 ciclos anidados, O(n^3), tiene 3 ciclos anidados. Un algoritmo de complejidad, O(log n), en el caso de que los datos a ser procesados aumenta, el número de operaciones aumenta pero no de forma proporcional a los datos, éste tipo de algoritmos presentan diferentes condiciones que hacen que ciertas instrucciones u operaciones sean o no ejecutadas. El algoritmo de Dijkstra, tiene una complejidad de tipo O(n^2), ya que presenta 2 bucles anidados de orden n, que es el número de nodos.

La Tabla 3.2 muestra el orden de complejidad para medir la eficiencia de algoritmos. El número de operaciones n es mostrado en cada fila (10 - 10^6) para el tiempo estimado T(n), tomando como referencia un procesador que ejecuta instrucciones a una velocidad de reloj de 1 MHz, en cada columna se presenta el grado de complejidad.

Tabla 3.2: Descripción entre la complejidad de algoritmos y el tiempo estimado de cómputo.

T(n) / n	Log n	n	n Log n	n^2	2^n	$n!$
10	3.3×10^{-6}	1×10^{-5}	3.3×10^{-5}	1×10^{-4}	0.001	3.63
50	5.6×10^{-6}	5×10^{-5}	2.8×10^{-4}	0.0025	indef	indef
100	6.6×10^{-6}	1×10^{-4}	6.6×10^{-4}	0.01	indef	indef
10^3	1×10^{-5}	0.001	0.01	1	indef	indef
10^4	1.3×10^{-5}	0.01	0.13	100	indef	indef
10^5	1.6×10^{-5}	0.1	1.6	$1x10^4$	indef	indef
10^6	2×10^{-5}	1	19.9	1×10^6	indef	indef

Se puede realizar un análisis de todas las rutas el algoritmo que se implementó para obtener todas las rutas de cada circuito fue el de búsqueda primero en profundidad (*Depth First Search* - DFS), complementado con *Backtracking* (recursión), con el fin de tener cada ruta a partir de un origen especificado. La Figura 3.5 muestra el proceso para obtener las rutas del grafo ponderado.

Figura 3.5: Proceso para detectar rutas lógicas

DFS es un algoritmo utilizado para recorrer los nodos de un grafo, a partir de un nodo inicial, tiene como base un recorrido preorden [43]. Una vez establecido el nodo inicial n como parámetro, va recorriendo (procesando) recursivamente cada nodo adyacente al nodo referenciado n. El algoritmo no mantiene registro de cada ruta recorrida, por lo que da como resultado, una sola lista de todos los nodos que recorrió a partir de un nodo origen, es decir, muestra un listado con rutas parciales, por lo que es necesario integrarlo con una técnica *Backtracking*, que consiste en la recuperación de nodos en forma recursiva (hacia atrás), para que de como resultado cada ruta individual, desde un nodo origen a cualquier destino establecido como final o salida.

Básicamente no se requiere que el grafo sea ponderado para realizar el recorrido, pero en este caso se aprovecha el recorrido que hace, y se va realizando una suma acumulada de los pesos de los segmentos que hay en cada ruta. Para ejemplificar el proceso, se tomará el grafo obtenido del circuito ISCAS'S C17, que se muestra en la Figura 3.6.

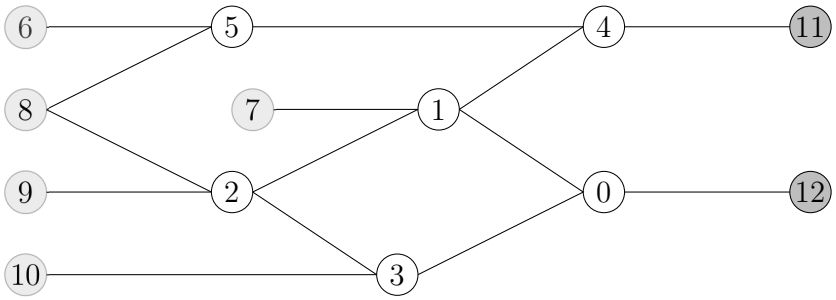


Figura 3.6: Grafo obtenido del circuito ISCAS'85 C17.

En referencia a la Figura 3.6, si se aplica el algoritmo DFS, como se explicó anteriormente, el resultado que arroja es una lista de todos los nodos recorridos, tomando como nodo origen el nodo 9, y procesando cada nodo subsecuente, se muestra a continuación el resultado:

$$9 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow 12 \rightarrow 1 \rightarrow 4 \rightarrow 11$$

Se puede observar, una ruta desde el nodo 9 al 12, pero inmediatamente muestra una ruta parcial que va del nodo 1 al 11, porque los nodos, 9 y 2 fueron procesados inicialmente.

Pero se puede observar en la Figura 3.6, que desde el nodo 9, se pueden tener 3 rutas hacia los nodos 11 y 12. El algoritmo, al complementarse con *Bactracking* (recursivo), da como resultado una ruta única a cada nodo final (11 y 12), el resultado se muestra a continuación:

$9 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow 12$
$9 \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow 12$
$9 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 11$

Algoritmo 1: $DepthFirstSearch$ - DFS

```
1 Función DFS(Nodo n) {
2 n.visitado = true;
3 for cada Nodo m adyacente a n do
4 |   if !m.visitado then
5 |   |   DFS(m);
6 |   |   n.visitado = false;
7 |   end
8 end
9 }
```

El algoritmo 1 realiza un recorrido desde un origen especificado (Nodo n - entrada primaria), encontrando los nodos adyacentes en forma sucesiva hasta encontrar el último (salida primaria), es decir, encuentra cada ruta. A partir del resultado de este algoritmo se realiza un filtrado de rutas, el cual consiste en eliminar aquellas que contienen un menor número de segmentos. El criterio utilizado para descartar las rutas consiste en mantener alrededor del 80 % de las rutas de cada circuito ISCAS'85 descartando las rutas más cortas. Lo que se logra al filtrar las rutas es reducir su cantidad, impactando en el tiempo de cómputo empleado por el algoritmo de Dijkstra modificado.

Al descartar rutas se genera un nuevo grafo, lo que da como resultado un subgrafo ponderado. Ya que el algoritmo de Dijkstra requiere como entrada un grafo ponderado, este puede ser completo o parcial (subgrafo).

3.1.5. Análisis con el algoritmo de Dijkstra

Mediante un gráfo se pueden representar diferentes situaciones que ayudan a resolver problemas o definir procesos. En nuestro caso la situación consiste en detectar las rutas que pueden ser críticas en un circuito electrónico. Como los segmentos de conexión tienen un parámetro que debe ser considerado para detectar si una ruta es crítica, requerimos representar al circuito como un grafo ponderado, como el que se muestra en la Figura 3.7.

En problemas aplicados en soluciones con grafos, si se plantea determinar la longitud del camino o ruta más corta entre un par de vértices, se debe considerar un grafo dirigido y valorado o ponderado, es decir, cada segmento (vi, vj) del grafo tiene asociado un peso w_{ij}, la longitud del camino más corto es la suma de los pesos de las aristas o segmentos que forman el camino o ruta [44]. Se puede realizar un análisis desde diferentes perspectivas, como buscar la ruta que tome el menor tiempo, la que sume un menor costo o la ruta que tenga menos segmentos. La ecuación matemática es:

$$w_r = \sum_{i=0}^{k-1} w_i \tag{3.2}$$

Donde w_r, es la sumatoria de los pesos w_i de cada segmento en la ruta desde el nodo i hasta el nodo $k-1$.

El tiempo de ejecución del algoritmo de Dijkstra es cuadrático, *O* (n^2), debido a los dos bucles anidados de orden n, [43], [42]. Este factor está relacionado con el número de vértices en el circuito bajo prueba.

Para el análisis de rutas se tienen algoritmos que normalmente buscan la ruta más corta, tal es el caso del algoritmo de Dijkstra, el cual requiere como parámetros de entrada un grafo (o subgrafo) ponderado y el nodo inicial (origen), a partir del cual encontrará el menor peso (suma de pesos que resulte menor) por cada nodo que se encuentre el camino, desde el origen establecido hasta cualquier destino correspondiente. Es decir, prácticamente Dijkstra encuentra el menor costo (distancia) desde un origen hasta cualquier destino.

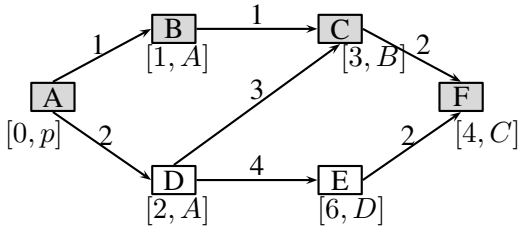


Figura 3.7: Grafo para describir la ruta más corta, usando el algoritmo de Dijkstra.

Con base en la Figura 3.7, teniendo como referencia de entrada el nodo A, Dijkstra encuentra la distancia (suma de pesos) a cada nodo tal como lo muestra la Tabla 3.3. Ya que el nodo A, es el origen, este tiene un peso 0. Para llegar al nodo B, se tiene un costo de 1, y al nodo D, el costo es de 2. Continuando el análisis de costos de rutas desde el nodo A como origen se puede observar que para llegar al nodo C se tienen 2 rutas: $A \rightarrow B \rightarrow C$ con peso de 2 y $A \rightarrow D \rightarrow C$ con peso de 5. El algoritmo de Dijkstra debe decidir qué ruta tiene menor costo, por lo cual establece que la ruta $A \rightarrow B \rightarrow C$ es la mejor ruta, por esa razón es el costo que se registra en la Tabla 3.3, de los costos asociados al nodo C.

3.1.6. Algoritmo de Dijkstra modificado

El algoritmo de Dijkstra utiliza como entrada un grafo ponderado, basado en un circuito. Para cada entrada primaria a cada destino (salida primaria), el algoritmo de Dijkstra selecciona la ruta crítica más afectada por las capacitancias de acoplamiento. El grafo ponderado se crea con la lista de conexiones que describe el circuito y el valor de acoplamiento capacitivo total de las líneas de segmento (aristas). Cada vértice o nodo en el grafo representa una compuerta lógica y la conexión entre las compuertas lógicas está representada por la línea de segmento con un peso. Por lo tanto, se obtiene un grafo ponderado del circuito. La información desde un nodo de origen (*entrada primaria*) a cualquier nodo, llamado valor de distancia, se almacena en una tabla. Se obtienen todos los destinos posibles de cada nodo fuente. El valor de la distancia es la suma de pesos entre un nodo de origen y un nodo de destino.

Tabla 3.3: Aplicación del algoritmo de Dijkstra para el grafo de la Figura 3.7, tomando como origen el nodo A

Nodo	Costo
A	0
B	1
C	2
D	2
E	6
F	4

Por lo general, el algoritmo de Dijkstra se aplica en problemas donde se desea encontrar el camino más corto. El algoritmo encuentra la ruta con el costo (también conocido distancia - *d*) más bajo entre el vértice de origen y todos los demás vértices del grafo. Las entradas y salidas principales del circuito se utilizan como vértice de origen y vértice de destino, respectivamente. El esquema que se muestra en la Figura 3.7 se utiliza para ilustrar cómo funciona el algoritmo de Dijkstra.

En primer lugar, el algoritmo de Dijkstra asigna un valor de distancia cero al nodo A (ver Figura 3.7) y lo etiqueta como permanente (el estado del nodo A es $[0, p]$). Todos los demás nodos tienen un estado temporal $[\infty, t]$. A continuación, se analizan los nodos B y D, ya que se pueden alcanzar desde el nodo actual A. Se actualizan los valores de distancias para estos nodos, $d_B = 1$ y $d_D = 2$. Entre los nodos B y D, el nodo B tiene el valor de distancia más pequeño desde $A[1, A]$. Ahora el nodo actual es B y se puede llegar al nodo C desde el nodo actual B. Los valores de los nodos de distancia se actualizan $d_C = 1 + 1 = 2$. A continuación, el nodo C tiene el valor de distancia más pequeño; tiene estatus permanente. Este proceso se repite para cada nodo. La ruta $A \rightarrow B \rightarrow C$ tiene el valor de distancia más pequeño $d_C = 2$. Finalmente, se puede alcanzar el nodo F y su valor de distancia se actualiza a $d_F = 4$. Como resultado, se obtiene la ruta más corta $A \rightarrow B \rightarrow C \rightarrow F$.

En nuestro caso, el algoritmo de Dijsktra, fue modificado para que ahora busque la ruta más larga, es decir, aquella cuya distancia presenta el valor más alto. Regresando al ejemplo del párrafo anterior, el nodo inicial A se etiqueta como permanente (el estado del nodo se establece como $[0, p]$], los demás nodos tienen un estado temporal $[\infty, t]$, pero ahora al analizar los nodos salientes de A, que son B y D, el nodo que tiene mayor distancia es D, por lo que se establece como nodo actual, y para poder ir al nodo C, la ruta sería a través de D, ahora los valores de distancia se actualizan $d_C = 2 + 3 = 5$. En la Figura 3.8 se puede observar la ruta mencionada.

Al buscar una ruta hacia el nodo F, se puede encontrar que, tomando la ruta a través del nodo E, se tiene una mayor distancia, por lo que ahora, la ruta con mayor distancia desde A, hacia F, es $A \rightarrow D \rightarrow E \rightarrow F$, cuya distancia es $d_F = 8$. En la Figura 3.9, se puede observar una nueva ruta

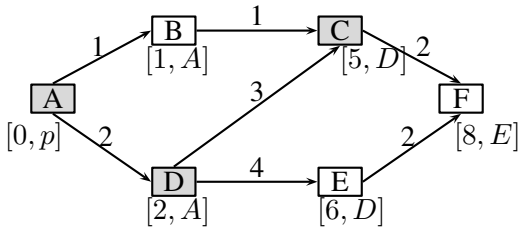


Figura 3.8: Grafo de la ruta más larga, del nodo A, al nodo C, usando el algoritmo de Dijkstra modificado.

con el costo más alto, desde el nodo A, al nodo F.

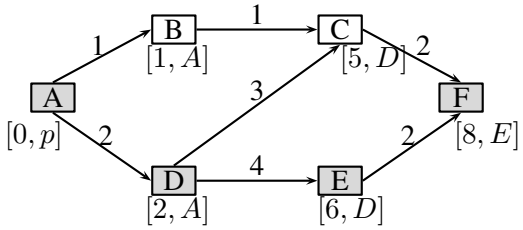


Figura 3.9: Grafo para detectar la ruta más larga, de A a F, usando el algoritmo de Dijkstra modificado.

En la Tabla 3.4 se puede observar el resultado que arroja el algoritmo de Dijkstra modificado, en el cual se busca la ruta más larga, es decir, desde un origen especificado, busca la ruta con valor de peso más alto para llegar a cada posible destino.

Tabla 3.4: Análisis con el algoritmo Dijkstra modificado de la Figura 3.7, tomando como origen el nodo A

Nodo	Costo
A	0
B	1
C	5
D	2
E	6
F	8

La eficiencia del algoritmo depende del número de elementos a procesar y del tipo de bucle involucrado. El tiempo de ejecución del algoritmo de Dijkstra es cuadrático, $O\ (n^2)$, debido a los

dos bucles anidados de order *n*, [43,45,46]. Este factor está relacionado con el número de nodos en el circuito bajo prueba.

Algoritmo 2: Pseudocódigo modificado del algoritmo de Dijsktra para obtener la ruta más larga.

Entrada: Grafo ponderado, Nodo origen.
Salida : Ruta crítica, costos de rutas

```
1  Pseudocódico del algoritmo de Dijkstra modificado:
2  function dijkstra(Grafo g, int origen) {
3  integer i;
4  float distancia;
5  nodo n, w;
6  n = source;
7  while No seleccionado(n) do
8      distance = nodoDistanciaConocida(n);
9      nodosConocidos[origen] = true;
10     for w = 0 a cadaNodoAdyacente do
11         if distancia + g.value(n,w) >distancia then
12             nodeDistanciaConocida[w] = distancia + g.values[n,w];
13             rutaNodo[w] = n;
14             nodoSeleccionado[n] = true;
15             n = w;
16 }
```

3.1.7. Funcionamiento del algoritmo de Dijkstra bajo efectos de acoplamiento en circuitos

La Figura 3.10 muestra el esquema del circuito de referencia ISCAS'85 C17, que se compone de seis puertas Nand de 2 entradas, cinco entradas primarias y dos salidas primarias. En la Figura 3.11 se muestra una representación simplificada de todas las rutas lógicas del cirtuito. Las entradas N1 y N2 presentan una única ruta lógica a la salida N22. Otras entradas presentan más de una ruta lógica desde su entrada hasta una salida determinada. Aún más, una entrada puede tener rutas lógicas a más de una salida. Por ejemplo, la entrada N3 tiene cuatro posibles rutas lógicas, dos rutas lógicas de N3 a N22 y otras dos rutas lógicas de N3 a N23.

La Figura 3.10 ilustra las dos rutas lógicas desde la entrada N3 hasta la salida N22 con pesos asignados a cada línea de segmento. Los pesos de las líneas de segmento (w) se calculan usando la Ecuación (3.1). Para la entrada N3 a la salida N22, el algoritmo de Dijkstra analiza y compara los valores w asociados con cada línea de segmento de las rutas y selecciona la ruta con los valores más altos. Nuestra metodología propuesta basada en el algoritmo de Dijkstra analiza las rutas lógicas

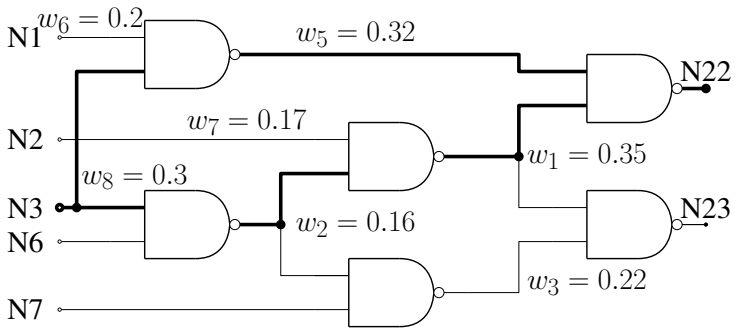


Figura 3.10: Esquemático del circuito de referencia ISCAS'85 C17.

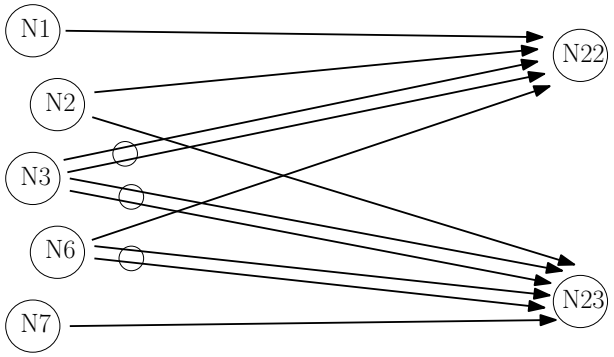


Figura 3.11: Rutas Totales en el circuito C17.

desde una entrada a una única salida del circuito, y entrega la ruta más significativamente influenciada por las capacitancias de acoplamiento. El pseudocódigo del algoritmo de Dijkstra modificado utilizado en este trabajo se muestra en el Algoritmo 2.

3.1.8. Ejemplo de la aplicación del algoritmo de Dijkstra

La Figura 3.12 muestra un gráfo del circuito de referencia ISCAS'85 C17. Las entradas del circuito, denominadas como nodos de origen, se indican en círculos de color gris claro. Las salidas del circuito, denominados nodos de destino, se muestran con un círculo gris oscuro. Los círculos de color blanco representan compuertas lógicas. Una arista (segmento) se utiliza para representar cómo se conectan las compuertas, entradas y salidas. Cada arista tiene asociado un peso que representa el valor de capacitancia de acoplamiento de la línea del segmento crítico.

Los subcircuitos de C17 que se muestran en las Figuras 3.13 a 3.17 se utilizan para ilustrar cómo Dijkstra realiza la identificación de rutas lógicas bajo acoplamiento severo. Los valores de las distancias asociadas con los nodos se almacenan en una tabla. En otras palabras, se almacenan los pesos acumulados desde un nodo de origen seleccionado a un nodo de destino seleccionado.

El proceso comienza con la selección de un nodo de origen y finaliza en un nodo de destino.

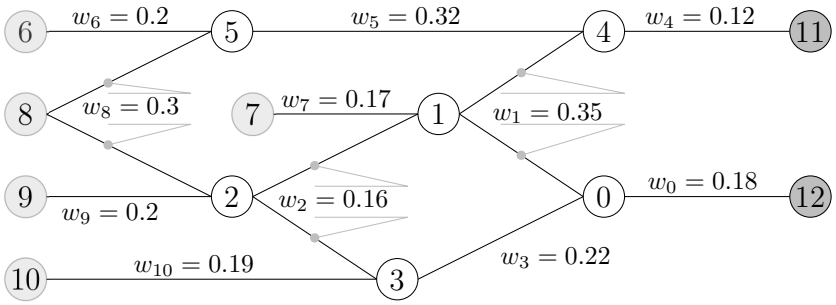
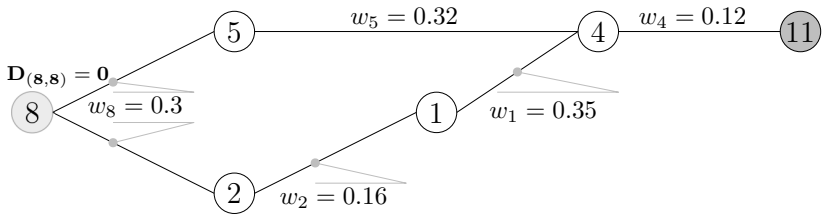


Figura 3.12: Grafo ponderado analizado por el algoritmo de Dijkstra.



Nodo	Distancia $D_{(8,8)}$
8	**0**

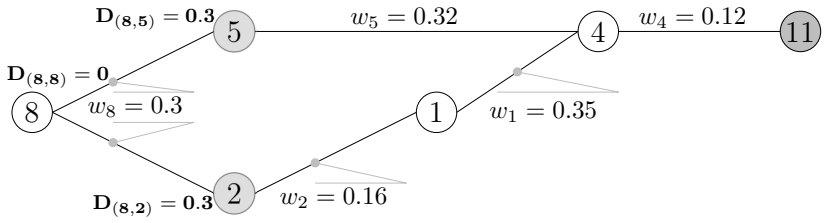
Figura 3.13: Nodo origen del grafo ponderado.

Para el circuito analizado, el nodo 8 y el nodo 5 se seleccionan como los nodos de origen y destino (ver Figura 3.13), respectivamente. El nodo de origen se inicializa con un valor de distancia cero ($D_{(8,8)}$ = 0). A continuación, se analizan los valores de distancia de los nodos adyacentes. El nodo de origen está conectado a través de un segmento con los nodos adyacentes. El nodo de origen 8 tiene dos nodos adyacentes (nodos 5 y 2 en la Figura 3.14). El valor de la distancia del borde entre los nodos 8 y 5 se suma al valor de la distancia del nodo fuente $D_{(8,8)}$. Esto da como resultado un valor de distancia de $D_{(8,5)}$ = 0.3 desde el nodo 8 al nodo 5. De manera similar, el valor de distancia de $D_{(8,2)}$ = 0 desde el nodo 8 a 2 se obtiene. Los pesos se almacenan como valores de distancia en una tabla (consulte la columna 3 en la Figura 3.14).

El algoritmo continúa analizando el segmento del nodo 5 al 4. El valor de la distancia del segmento de 5 a 4 es 0.32. Sumando este valor a $D_{(8,5)}$, se obtiene el valor de la distancia total del nodo 8 al 4 ($D_{(8,5,4)}$ = 0.62). La tabla con los nuevos valores de distancia se actualiza (ver Figura 3.15).

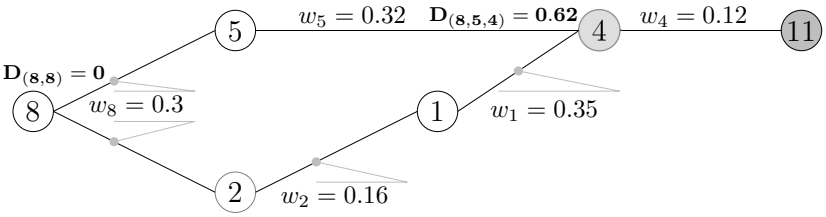
El algoritmo continúa el proceso analizando los segmentos adyacentes que conectan los nodos 8, 2 y 1 (ver Figura 3.16). El valor de la distancia acumulada desde el nodo 8 al 1 es $D_{(8,2,1)}$ = 0.46. Luego, el valor de la distancia del segmento del nodo 1 al 4 (0.35) se agrega a $D_{(8,2,1)}$. Esto da un valor de distancia total de 0.81 desde el nodo 8 al nodo 4 ($D_{(8,2,1,4)} = 0.81$). Esta información se actualiza en la columna $D_{(8,2,1,4)}$ en la tabla de la Figura 3.17.

El valor de la distancia $D_{(8,2,1,4)} = 0.81$ se compara con $D_{(8,5,4)} = 0.62$. En este caso, el valor



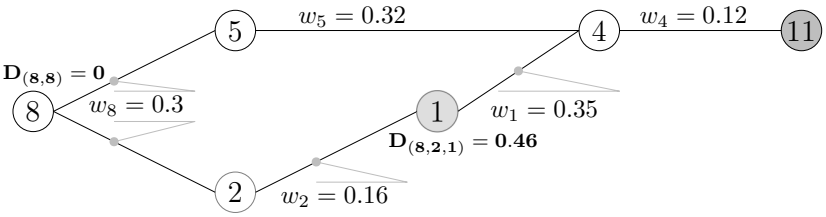
Nodo	Distancia $D_{(8,8)}$	Distancia $D_{(8,5)}$	Distancia $D_{(8,2)}$
8	0	0	0
5		**0.3**	0
2		0	**0.3**

Figura 3.14: Grafo ponderado con la ruta de nodos 8 a 5 y 8 a 2.



Nodo	Distancia $D_{(8,8)}$	Distancia $D_{(8,5)}$ $D_{(8,2)}$	Distancia $D_{(8,5,4)}$
8	0	0	0
5		0.3	0.3
2		0.3	0.3
4			**0.62**

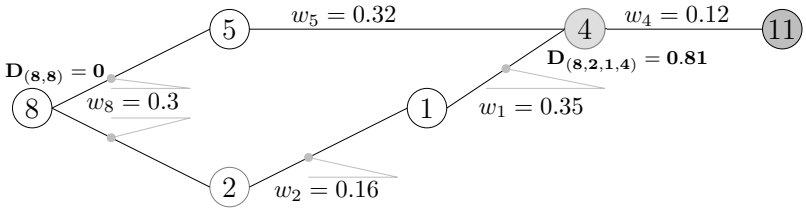
Figura 3.15: Grafo ponderado, nodos de 5 a 4.



Nodo	Distancia $D_{(8,8)}$	Distancia $D_{(8,5)}$ $D_{(8,2)}$	Distancia $D_{(8,5,4)}$	Distancia $D_{(8,2,1)}$
8	0	0	0	0
5		0.3	0.3	0.3
2		0.3	0.3	0.3
4			0.62	0.62
1				**0.46**

Figura 3.16: Grafo ponderad, nodos del 2 al 1.

de la distancia de $D_{(8,2,1,4)}$ es mayor que $D_{(8,5,4)} = 0.62$. Por lo tanto, el valor más alto se utilizará como la distancia dominante desde el origen (nodo 8) al nodo 4. Finalmente, el valor de la distancia del segmento del nodo 4 al 11 (0.12) se suma al valor de la distancia de 8 a 4. Como resultado, se obtiene la distancia $D_{(8,2,1,4,11)} = 0.93$, que es la ruta lógica con el valor capacitivo más alto. La información actualizada se muestra en la columna $D_{(8,2,1,4,11)}$ en la Tabla de la Figura 3.17.



Nodo	Distancia $D_{(8,8)}$	Distancia $D_{(8,5)}$ $D_{(8,2)}$	Distancia $D_{(8,5,4)}$	Distancia $D_{(8,2,1)}$	Distancia $D_{(8,2,1,4)}$	Distancia $D_{(8,2,1,4,11)}$
8	0	0	0	0	0	0
5		0.3	0.3	0.3	0.3	0.3
2		0.3	0.3	0.3	0.3	0.3
4			0.62	0.62	**0.81**	0.81
1				0.46	0.46	0.46
11						**0.93**

Figura 3.17: Nodos del 1 al 4, del grafo ponderado.

Capítulo 4

Resultados

4.1. Circuitos de referencia ISCAS'85

La metodología propuesta para identificar rutas críticas influenciadas por acoplamientos severos, inicia tomando como referencia un circuito electrónico en Verilog y sus acoplamientos capacitivos, para ser representado por un grafo. Al aplicar un algoritmo de recorrido de búsqueda primero en profundidad (DFS), se pueden obtener todas las rutas, para analizarlas con el fin de filtrarlas y tener un grafo optimizado, que es utilizado como entrada en el algoritmo de Dijkstra modificado. El programa se ejecutó en un sistema operativo Unix con procesador de 1.8 GHz Intel Core i7 y 12GB en RAM. La efectividad de la metodología fue evaluada en circuitos de prueba estándar ISCAS'85 utilizando tecnología de 65nm.

La Tabla 4.1 presenta las características más importantes de los circuitos analizados. El número de entradas, número de salidas y de compuertas son mostradas de la columna 2 a la 4, respectivamente.

4.2. Análisis de las líneas de segmento.

La Tabla 4.2 muestra los resultados obtenidos del análisis de las líneas de segmento realizado por la metodología propuesta. La segunda columna presenta el número total de líneas de segmento en los circuitos de referencia ISCAS'85 analizados. Las columnas 3 y 4 contienen los resultados para factores de selección (f) de 0.2 y 0.4, respectivamente. Además se observa que el número de líneas de segmento seleccionadas disminuye a medida que aumenta el factor de selección. Esto se debe a que un número menor de líneas acopladas a los segmentos cumplen la condición de la Ecuación (2.3). Por lo tanto, el diseñador o ingeniero de pruebas puede definir el grado de severidad del acoplamiento a analizar.

Tabla 4.1: Lista de circuitos ISCAS'85 analizados en este trabajo [47].

Circuito	# Compuertas	# Entradas	# Salidas
C17	6	5	2
C432	160	37	7
C499	202	41	32
C1908	880	33	25
C1355	546	41	32
C3540	1669	50	22
C5315	2406	178	123
C7552	3512	207	108

Tabla 4.2: Resultados del análisis de las líneas de segmento.

Circuitos	# Total Segmentos	Segmentos seleccionados	
		$f = 0.2$	$f = 0.4$
C17	14	14	8
C432	369	156	44
C499	392	85	4
C1355	392	84	6
C1908	506	149	39
C3540	1963	1134	417
C5315	2596	1125	397
C7552	2186	906	326

4.3. Rutas seleccionadas

Al ejecutar el programa con Dijkstra, se detectó que se podía reducir los tiempos de ejecución, principalmente en circuitos con una gran cantidad de rutas, por lo que se aprovechó la aplicación del algoritmo DFS para reducir la cantidad de rutas a analizar reduciendo el grafo, es decir generar un subgrafo que se obtiene mediante un filtrado de rutas.

4.3.1. Filtrado de rutas

El procedimiento para filtrar rutas consiste en identificar cada ruta entre cada entrada y cada salida y al mismo tiempo contabilizar los segmentos. Posteriormente se agrupan las rutas por el número de segmentos (por ejemplo: grupo de rutas con segmentos de 1 a 5, grupo de rutas con segmentos de 6 a 10, etc.). Finalmente se descartan las rutas consideradas con pocos segmentos, que sumen aproximadamente 20 % del total de las rutas de cada circuito. Se aplica el algoritmo DFS a un circuito bajo prueba, para buscar todas las posibles rutas para todas las entradas, contabilizando los segmentos contenidos en cada una y descartando aquellas rutas con pocos segmentos. Las rutas filtradas son aquellas rutas con mayor número de segmentos y mayor número de líneas acopladas por segmento. La Tabla 4.3 muestra el resultado obtenido por el algoritmo DFS, la columna 2 contiene el número de rutas totales de los circuitos bajo prueba en la columna 1, y en la columna, rutas filtradas se encuentra el número de rutas a considerar para que sean analizadas por el algoritmo de Dijkstra (tercera columna de la Tabla 4.6). Finalmente el algoritmo de Dijkstra modificado identifica el conjunto de rutas lógicas para acoplamientos capacitivos para diferentes valores del factor de selección.

Tabla 4.3: Rutas filtradas por el algoritmo DFS

Circuito	Rutas Totales	Rutas Filtradas
C17	11	6
C432	79822	65065
C499	7648	6016
C1355	7648	5632
C1908	23307	17496
C3540	436495	336413
C5315	31860	25368
C7552	26655	20745

Después de analizar el grafo original (circuito completo) y realizar un filtrado de rutas, se vuelve a generar un grafo ponderado (subgrafo), que será empleado como entrada en el algoritmo de Dijkstra modificado. En las columnas 3 y 6 de la Tabla 4.4, se observa la disminución de tiempo que tomó el algoritmo para analizar el grafo completo y el subgrafo. Es decir, aplicando un factor de 0.4, en una primera instancia sin filtrar rutas y posteriormente con rutas filtradas. Es importante aclarar que el programa que implementa el algoritmo, no toma en cuenta el factor de selección, la entrada al programa es sólo el grafo del circuito correspondiente, los pesos de éste, y el nodo origen (entrada primaria). Las siguientes columnas se dividen en 2 secciones, una primera sección

muestra en la columna 2 las rutas seleccionadas tomando como entrada el grafo completo (circuito completo) y la segunda sección con las rutas seleccionadas del subgrafo; en las columnas 3 y 6, muestra el tiempo que tardó en analizarlas. Las columnas 4 y 7 contienen el porcentaje de rutas seleccionadas (Rutas Seleccionadas / Rutas Totales) respecto al total de rutas del grafo. El total de rutas se encuentra en la columna 2 de la Tabla 4.3. Se observa en los circuitos donde la cantidad de rutas es pequeña, como el C17, la diferencia de tiempo es mínima, pero en circuitos cuya diferencia en la cantidad de rutas filtradas es grande, como C3540 o C5315, la diferencia de tiempo es notoria.

Tabla 4.4: Comparación de tiempos, con rutas totales (grafo completo) y filtradas (subgrafo) para f = 0.4 en cada circuito ISCAS'85.

Circuito	Grafo completo, f=0.4			Subgrafo, f=0.4		
	Rutas Seleccionadas	Tiempo (ms)	RS/RT (%)	Rutas Seleccionadas	Tiempo (ms)	RS/RT %
C17	7	0.05	63.64	4	0.032	36.36
C432	200	4.08	0.25	163	3.30	0.20
C499	652	3.47	8.53	652	3.41	8.53
C1355	863	4.77	11.28	863	4.54	11.28
C1908	618	4.93	2.65	295	3.14	1.27
C3540	704	20.38	0.16	516	17.20	0.12
C5315	2687	20.24	8.43	2221	17.92	6.97
C7552	2531	25.74	12.26	2158	26.50	10.45

Se puede observar en la segunda y quinta columna de la Tabla 4.4, que se refiere a Rutas Seleccionadas, en el caso de la segunda columna se obtiene ese número analizando el circuito completo, en la quinta columna se presenta un subgrafo, teniendo una cantidad más exacta de rutas críticas, cuando se hace un filtrado de rutas, y el tiempo de proceso se reduce.

La Tabla 4.5, contiene resultados del algoritmo de Dijkstra modificado ahora tomando en cuenta un factor de selección f=0.2, a diferencia del factor de selección f=0.4 (resultados de la Tabla 4.4) se observa un mayor número de rutas seleccionadas. La columna 2 contiene la cantidad de rutas seleccionadas que, en el caso del circuito C17, se observa que el resultado es 8, es decir, todas las rutas peores posibles que se pueden presentar para ese circuito. La Figura 4.1 muestra las 11 rutas posibles del circuito C17, resaltando 8 (líneas gruesas) como las peores posibles. En la quinta columna se muestra el total de rutas seleccionadas por el algoritmo de Dijkstra modificado, en el caso del circuito C17, que presenta 4, coincide cuando también es analizado con un factor de selección de f=0.4. Esta situación se presenta porque el circuito es pequeño en cuanto a compuertas y cantidad de rutas.

También se puede observar en la Tabla 4.5 una diferencia importante en las columnas 2 y 5, en

Tabla 4.5: Comparación de tiempos, con rutas totales (grafo completo) y filtradas (subgrafo) para f = 0.2 en cada circuito ISCAS'85.

Circuito	Grafo completo, f=0.2			Subgrafo, f=0.2		
	Rutas Seleccionadas	Tiempo (ms)	RS/RT %	Rutas Seleccionadas	Tiempo (ms)	RS/RT %
C17	8	0.05	72.73	4	0.038	36.36
C432	220	5.05	0.28	180	4.74	0.23
C499	1312	5.07	17.15	1312	4.99	17.15
C1355	1309	6.00	17.12	1309	5.48	17.12
C1908	801	5.44	3.44	295	3.55	1.27
C3540	720	21.41	0.16	517	17.75	0.12
C5315	2879	22.89	9.04	2221	20.23	7.09
C7552	2632	30.00	12.75	2191	27.32	10.61

referencia a las rutas seleccionadas y el tiempo que tarda en realizar el análisis en el algoritmo de Dijkstra modificado. Cuando se obtiene el grafo basado en un factor de selección f=0.2, la cantidad de segmentos mayor a 0 es alta, por lo tanto el tiempo de cómputo se eleva, esto se ve reflejado en las columnas 3 y 6 que se refieren al tiempo que toma el algoritmo en analizar cada circuito.

4.3.2. Rutas seleccionadas con acoplamiento severo

La Tabla 4.6 contiene los circuitos ISCAS'85 analizados. El total de rutas para cada circuito se muestra en la columna 2, y en la columna 3 se muestran las rutas filtradas. Las rutas seleccionadas, tiempo de cómputo y porcentaje, para factores de selección de 0.2, 0.4 y 0.8, se muestran en las columnas 4 a la 12. Las columnas 4, 7 y 8 muestran las rutas seleccionadas por el Dijkstra modificado respecto a cada factor de selección. Este valor es dividido entre el valor correspondientes a la columna de 2, cuyo resultado se encuentra en la columna RS/RT.

La metodología propuesta proporciona un subconjunto de caminos considerados susceptibles de sufrir fallas debido a un mayor acoplamiento capacitivo. Para la condición menos severa del factor de selección (f = 0.2), el número de rutas seleccionadas representa un porcentaje moderado del número total de rutas, y el tiempo de CPU está en el rango entre 0.038ms y 27.32ms para los circuitos de referencia analizados. Para factores de selección mayores a 0.2 el número seleccionado de líneas como críticas es menor.

Tabla 4.6: Rutas seleccionadas por el algoritmo de Dijkstra modificado en la metodología propuesta

Circuito	Rutas Totales (RT)	Rutas Filtradas	$f = 0.2$			$f = 0.4$			$f = 0.8$		
			Rutas Selec. (RS)	Tiempo (ms)	RS/RT %	Rutas Selec. (RS)	Tiempo (ms)	RS/RT %	Rutas Selec. (RS)	Tiempo (ms)	RS/RT %
C17	11	6	4	0.040	36.36	4	0.038	36.36	2	0.032	18.18
C432	79822	65065	180	4.74	0.23	163	3.30	0.20	0	1.74	0.00
C499	7648	6016	1312	4.99	17.15	652	3.41	8.53	0	3.15	0.00
C1355	7648	5632	1309	5.48	17.12	863	4.54	11.28	0	3.17	0.00
C1908	23307	17496	295	3.55	1.27	295	3.14	1.27	0	2.12	0.00
C3540	436495	336413	517	17.75	0.12	516	17.20	0.12	145	6.56	0.03
C5315	31860	25368	2558	20.23	7.09	2221	17.92	6.97	321	13.66	1.01
C7552	26655	20745	2191	27.32	10.61	2158	26.50	10.45	131	16.72	0.63

4.4. Validación de la propuesta

En las pruebas realizadas con el algoritmo de Dijkstra y un factor de selección hay dos aspectos de interés para ser validados: a) el algoritmo de Dijkstra debe poder seleccionar una única ruta lógica desde cada entrada primaria a cada salida primaria, b) la ruta lógica seleccionada debe ser la que tenga la mayor capacitancia de acoplamiento cuando haya más de una ruta lógica desde una entrada primaria a una salida.

La Tabla 4.7 muestra el número de rutas lógicas entre cada entrada y cada salida para el circuito C17 se aprecia que para la entrada N1 y la salida N22 existe sólo una ruta, para la salida N23 no hay ruta, caso contrario para la entrada N7 no hay ruta lógica a la salida N22, pero sí a la salida N23. En el caso de la entrada N3, presenta 2 rutas lógicas para la salida N22 y N23. El algoritmo de Dijkstra modificado analizará cada ruta lógica con sus respectivos valores capacitivos por segmento seleccionando aquellos que presenten el mayor valor capacitivo desde la entrada hasta la salida.

Tabla 4.7: Rutas lógicas del circuito ISCAS'C17

Entradas	Salidas	
	N22	N23
N1	1	0
N2	1	1
N3	2	2
N6	1	2
N7	0	1

Los resultados globales para otros circuitos ISCAS'85 se muestran en la Tabla 4.8. La segunda columna muestra el número de rutas lógicas para cada circuito. La tercera columna contiene el número de rutas seleccionadas aplicando un factor de selección $f = 0.2$. Se puede observar que el número de caminos lógicos seleccionados por el algoritmo de Dijkstra sigue de cerca al número de caminos lógicos del diseño del circuito contando un camino único desde cada entrada a cada salida. Adicionalmente, el algoritmo de Dijkstra puede seleccionar un número menor de rutas lógicas como las esperadas debido al factor de selección (ver Tabla 4.6).

Tabla 4.8: Comparación de las rutas seleccionadass por Dijkstra y las de cada circuito

Circuit	Rutas lógicas (Diseño de circuito)	Rutas seleccionadas Dijkstra (f = 0.2)
C17	8	4
C432	225	180
C499	1312	1312
C1355	1312	1309
C1908	807	295
C3540	724	517
C5315	2900	2558
C7552	2254	2191

La Figura 4.1 muestra el total de rutas lógicas contenidas del diseño del circuito C17 con sus respectivos pesos w asignados a cada segmento. La figura 4.1 muestra algunas rutas con líneas más gruesas que representan los caminos críticos seleccionados por el algoritmo de Dijkstra modificado. Existen rutas únicas para las que el algoritmo identifica como críticas por ser únicas. Para las rutas lógicas que van desde N3 hasta N23 y N22, así como de N6 hasta N23, el algoritmo identifica solo las rutas (líneas más gruesas) con mayor peso w acumulado. Se observa que desde N3 hasta N23 y N3 hasta N22 se tienen las siguientes rutas:

$N3 \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow N23$
$N3 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow N23$
$N3 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow N22$
$N3 \rightarrow 5 \rightarrow 4 \rightarrow N22$

El algoritmo da como resultado la ruta $N3 \rightarrow 2 \rightarrow 1 \rightarrow 0 \rightarrow N23$, debido a que la suma de los pesos w_8 + w_2 + w_1 + w_0 acumulan un total de 2.07. Mientras que la ruta $N3 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow N23$ presenta una suma total de 2.04 (w_8 + w_2 + w_3 + w_0). Por otro lado para las rutas conformadas entre la entrada N3 y la salida N22, el algoritmo identifica la ruta $N3 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow N22$ como

crítica, ya que presenta un acumulado de pesos de 2.23 superior a 1.73 de la ruta $N3 \rightarrow 5 \rightarrow 4 \rightarrow N22$. Otro caso es N6 que tiene 2 rutas hacia N23: $N6 \rightarrow 1 \rightarrow 3 \rightarrow 0 \rightarrow N23$ con una suma total de pesos de 2.04 y $N6 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow N23$ con suma de pesos 2.01 identificando como crítica la primera (w_9 + w_2 + w_1 + w_0 = 2.04).

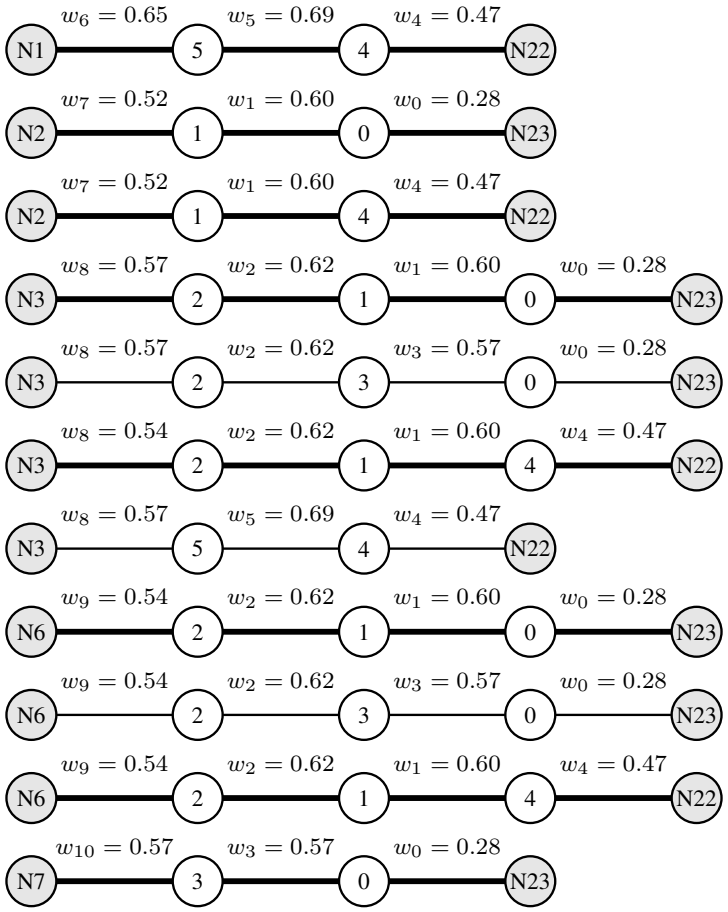


Figura 4.1: Rutas lógicas analizadas desde cada entrada hacia cada salida correspondiente.

4.5. Aplicación de la propuesta

La metodología propuesta obtiene un conjunto de caminos más largos entre las entradas primarias y las salidas primarias más influenciadas por las capacitancias de acoplamiento a un bajo costo computacional. A continuación se describen dos aplicaciones:

1) Aplicarse en la detección de defectos de interconexión (*p. e.*, *full opens* y *short defects*) [48]. Aplicando valores lógicos apropiados en las líneas acopladas a esas líneas de segmento de las rutas críticas se favorece la detección de defectos del tipo *full open*. Por ejemplo, los valores de 0 lógico

(1 lógico) se aplican a las líneas acopladas para probar una falla *stuck-0* (*stuck-1*) en una línea de segmento. Adicionalmente, la cobertura de defectos de las aperturas completas se puede cuantificar usando un simulador de fallas y, el intercambio entre el costo computacional y la cobertura de defectos [48]. Vale la pena mencionar que nuestra propuesta obtiene un conjunto de rutas más pequeño que en [48].

2) También se puede utilizar para analizar el impacto del ruido en el comportamiento del circuito en las primeras etapas del ciclo de diseño (por ejemplo diseñ inicial y ruteo) [49]. Por lo tanto, ayuda a acelerar el tiempo de entrega [49]. Se puede realizar un análisis de acoplamiento más preciso para diseñar [49]. Además, un diseñador puede identificar los segmentos de una ruta crítica más afectados por las capacidades de acoplamiento. Se pueden aplicar diferentes técnicas para mitigar el impacto de las capacitancias de acoplamiento, como el ordenamiento, optimización y separación de líneas.

4.5.1. Trabajo relacionado con la metodología propuesta

Algunos aspectos esenciales que involucran los conceptos básicos y el desempeño de esta propuesta se discuten en perspectiva con trabajos relacionados en la literatura. Específicamente, nos enfocamos en la composición del conjunto de caminos obtenidos, el método para obtener el conjunto de caminos y, brevemente, calcular el tiempo.

Nuestra propuesta se diferencia de otros trabajos relacionados en la composición del conjunto de caminos críticos obtenidos. Los primeros trabajos de investigación sobre la generación de las rutas más largas a través de cada compuerta se pueden encontrar en [50, 51]. Además, en [52], el conjunto de caminos se compone de *K* caminos más largos a través de cada compuerta. Se obtiene más de un camino por cada compuerta para favorecer la detección de pequeños defectos de retardo.

También se puede utilizar una herramienta de análisis de tiempo para encontrar las rutas comprobables más largas [53]. El análisis de tiempos se basa en los retardos de las compuertas y su propagación a través de las rutas lógicas del circuito.

En [54], se encontró un conjunto de caminos críticos para pruebas de retardo, considerando el impacto de las capacitancias de acoplamiento. Sin embargo está metodología obtiene las rutas más largas de cada entrada primaria a cada salida primaria, considerando el impacto del acoplamiento. Así, se obtiene un conjunto reducido de caminos.

Existen varios métodos para encontrar un conjunto de rutas críticas. Algunos métodos para encontrar un conjunto de rutas críticas implican la estimación de retrasos en las rutas. La selección de la ruta se puede hacer comparando las demoras de la ruta con un valor umbral [55] o clasificando las demoras de las rutas [52]. En [52], se utiliza una metodología ATPG para encontrar las rutas críticas más largas de K. En [54], se determinan tanto las limitaciones lógicas como de tiempo. Luego, el retardo de ruta máximo bajo efectos de acoplamiento se encuentra usando una formulación de optimización restringida. Nuestra propuesta se basa en un algoritmo de Dijkstra modificado. Las aristas (líneas de segmento) del grafo se ponderan por el impacto de las capacitancias de acoplamiento para seleccionar la ruta crítica más larga desde una entrada primaria a una

salida primaria.

El algoritmo de Dijkstra es un algoritmo eficiente de ruta más corta [40,56]. Por lo tanto, nuestra propuesta de modificación presenta beneficios en el tiempo de cómputo para encontrar aquellos caminos críticos más largos influenciados más fuertemente por las capacitancias de acoplamiento.

Capítulo 5

Conclusiones

El número de transistores en un solo chip se ha incrementado rápidamente con el paso de los años, con un mayor número de interconexiones y efectos capacitivos más severos. Debido a esto, el número de fallas es alto, por lo tanto numerosas técnicas y algoritmos de detección de fallas en interconexiones se han propuesto. El algoritmo de Dijkstra es conocido por su eficiencia en la detección de caminos cortos. Aprovechando sus características se modificó para encontrar el camino más largo. La detección de líneas de interconexión con acoplamientos altamente capacitivos permite al diseñador aplicar técnicas de *Design For Testability* (DFT). El factor de selección f es una característica importante de esta metodología. El factor f permite controlar la cantidad de acoplamientos capacitivos analizados por segmento. El algoritmo se aplicó a los circuitos ISCAS'85 descritos en la tabla 4.6.

En [52–54] se presentan trabajos con propuestas de análisis tomando criterios basados en estimación de retardo para encontrar caminos (más de un camino) por cada compuerta. En [54] propone una metodología para encontrar un conjunto de caminos críticos considerando líneas de acoplamiento en pruebas de retardo. Nuestra metodología genera un grafo ponderado partiendo de la Netlist del circuito y el archivo de extracción de capacitancias parásitas identificando la ruta más larga entra cada entrada y salidas primarias. La asociación de pesos a los segmentos empleado en esta metodología se basa en un modelo eléctrico de acoplamientos capacitivos entre líneas de interconexión, así como capacitancias de compuerta conectadas a la línea del segmento (*fanout*).

Se ha comprobado que una forma eficiente de encontrar las rutas más cortas, o largas, en un diagrama de conexiones de un circuito integrado es utilizando algoritmos de grafos [40]. El empleo de grafos permite generar subgrafos disminyendo el tiempo de análisis de rutas.

La metodología propuesta con el algoritmo Dijkstra modificado permite descartar las rutas con segmentos menos afectados por acoplamientos capacitivos y enfocar el análisis en aquellas rutas con mayor número de segmentos y acoplamientos. Esto puede ser usado en la validación del comportamiento de circuitos y en *testing* permitiendo mejorar la estrategia en la generación de vectores de prueba.

La tecnología utilizada en los circuitos ISCAS'85 fue de 65nm. Ya que se trata de circuitos

de prueba estándar que constituyen lógica combinacional, la metodología desarrollada es independiente de la tecnología y puede ser empleada en tecnologías actuales.

Los algoritmos empleados fueron desarrollados en C++ para procesar Netlist y extracción de capacitancias parásitas y generar un grafo ponderado del circuito. También fue necesario desarrollar una rutina para generar rutas topológicas y flltrar rutas (subgrafo). Además del desarrollo del algoritmo de Dijkstra modificado para obtener las rutas críticas.

Como se detalló en el Capítulo 4, los tiempos de cómputo de rutas analizadas se redujeron considerablemente gracias a los criterios de selección como el factor f y al análisis de rutas topológicas. Como trabajo futuro se considera el desarrollo de proceso para la generación de vectores de prueba y detectar defectos del tipo *stuck-at* en las rutas consideradas como críticas y otro tipo de fallas en interconexiones.

Índice de figuras

Índice de tablas

Bibliografía

[1] M. Abramovici, M. A. Breuer, A. D. Friedman *et al.*, *Digital systems testing and testable design*. Computer science press New York, 1990, vol. 2.

[2] K. Baker, G. Gronthoud, M. Lousberg, I. Schanstra, and C. Hawkins, "Defect-based delay testing of resistive vias-contacts a critical evaluation," in *International Test Conference 1999. Proceedings (IEEE Cat. No. 99CH37034)*. IEEE, 1999, pp. 467–476.

[3] D. B. Feltham and W. Maly, "Physically realistic fault models for analog cmos neural networks," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 9, pp. 1223–1229, 1991.

[4] W. Needham, C. Prunty, and E. H. Yeoh, "High volume microprocessor test escapes, an analysis of defects our tests are missing," in *Proceedings International Test Conference 1998 (IEEE Cat. No. 98CH36270)*. IEEE, 1998, pp. 25–34.

[5] A. Z. Ramırez, "Modeling of open defects in CMOS integrated circuits and test techniques for submicron technologies," Ph.D. dissertation, PhD thesis, INAOE, Puebla, México, 2000.

[6] A. Pancholy, J. Rajski, and L. J. McNaughton, "Empirical failure analysis and validation of fault models in cmos vlsi," in *Proceedings. International Test Conference 1990*. IEEE, 1990, pp. 938–947.

[7] W. Maly, "Realistic fault modeling for VLSI testing," in *24th ACM/IEEE Design Automation Conference*. IEEE, 1987, pp. 173–180.

[8] H. Fujiwara, *Logic testing and design for testability*. MIT press Cambridge, MA, 1985.

[9] R. Rajsuman, *Digital Hardware Testing: Transistor-Level Fault Modeling and Testing*. Artech House, Inc., 1992.

[10] J. A. Abraham and H. Shih, "Testing of mos vlsi circuits," in *IEEE Int. Symp. on Circuits and Systems*, vol. 12971300, 1985.

[11] R. R. Fritzemeier, C. F. Hawkins, and J. M. Soden, "CMOS IC fault models, physical defect coverage, and I/sub DDQ/testing," in *Proceedings of the IEEE 1991 Custom Integrated Circuits Conference*. IEEE, 1991, pp. 13–1.

[12] D. G. Edwards, "Testing for mos integrated circuit failure modes," in *IEEE, Reprint from Proceedings International Test Conference*, 1980, pp. 303–312.

[13] M. E. Turner, D. G. Leet, R. J. Prilik, and D. J. McLean, "Testing cmos vlsi: Tools, concepts, and experimental results." in *ITC*, 1985, pp. 322–328.

[14] T. W. Williams and N. Brown, "Defect level as a function of fault coverage," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 987–988, 1981.

[15] A. E. Gattiker and W. Maly, "Toward understanding Iddq-only fails," in *Proceedings International Test Conference 1998 (IEEE Cat. No. 98CH36270)*. IEEE, 1998, pp. 174–183.

[16] K.-J. Lee and M. A. Breuer, "Constraints for using iddq testing to detect cmos bridging faults," in *Digest of Papers 1991 VLSI Test Symposium'Chip-to-System Test Concerns for the 90's*. IEEE Computer Society, 1991, pp. 303–304.

[17] J. Segura and C. F. Hawkins, *CMOS electronics: how it works, how it fails*. John Wiley & Sons, 2004.

[18] F. J. Ferguson, M. Taylor, and T. Larrabee, "Testing for parametric faults in static cmos circuits," in *Proceedings. International Test Conference 1990*. IEEE, 1990, pp. 436–443.

[19] S. F. Midkiff and S. W. Bollinger, "Circuit-level classification and testability analysis for cmos faults," in *Digest of Papers 1991 VLSI Test Symposium'Chip-to-System Test Concerns for the 90's*. IEEE, 1991, pp. 193–198.

[20] W. Maly, P. K. Nag, and P. Nigh, "Testing oriented analysis of cmos ics with opens," in *[1988] IEEE International Conference on Computer-Aided Design (ICCAD-89) Digest of Technical Papers*. IEEE, 1988, pp. 344–347.

[21] C. L. Henderson, J. M. Soden, and C. F. Hawkins, "The behavior and testing implications of cmos ic logic gate open circuits," *NASA STI/Recon Technical Report N*, vol. 92, p. 12189, 1991.

[22] R. Rodríguez-Montañés, J. A. Segura, V. H. Champac, J. Figueras, and J. Rubio, "Current vs. logic testing of gate oxide short, floating gate and bridging failures in cmos," in *1991, Proceedings. International Test Conference*. IEEE, 1991, p. 510.

[23] C. F. Hawkins, H. T. Nagle, R. R. Fritzemeier, and J. R. Guth, "The vlsi circuit test problem-a tutorial," *IEEE Transactions on industrial Electronics*, vol. 36, no. 2, pp. 111–116, 1989.

[24] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM journal of Research and Development*, vol. 10, no. 4, pp. 278–291, 1966.

[25] R. R. Fritzemeier, H. T. Nagle, and C. F. Hawkins, "Fundamentals of testability-a tutorial," *IEEE Transactions on Industrial Electronics*, vol. 36, no. 2, pp. 117–128, 1989.

[26] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems,*. IEEE, 1989, pp. 1929–1934.

[27] J. P. Uyemura, "Diseño de sistemas digitales: Un enfoque integrado." International, 2000.

[28] M. Sachdev and J. P. De Gyvez, *Defect-oriented testing for nano-metric CMOS VLSI circuits*. Springer Science & Business Media, 2007, vol. 34.

[29] N. H. Weste and D. Harris, *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[30] J. P. Uyemura, *CMOS Logic Circuit Design*. Georgia Institute of Technology, 2002.

[31] M. M. Mano, C. R. Kime, T. Martin *et al.*, *Logic and computer design fundamentals*. Prentice Hall, 2008, vol. 3.

[32] P. Pfeifer, J. Raik, M. Jenihhin, R. Ubar, and Z. Pliva, "Measuring and identifying aging-critical paths in fpgas," 2015.

[33] M. Tehranipoor, K. Peng, and K. Chakrabarty, *Test and diagnosis for small-delay defects*. Springer, 2011.

[34] F. Moll and A. Rubio, "Spurious signals in digital cmos vlsi circuits: a propagation analysis," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 10, pp. 749–752, 1992.

[35] P. Heydari and M. Pedram, "Analysis and reduction of capacitive coupling noise in high-speed vlsi circuits," in *Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference on*. IEEE, 2001, pp. 104–109.

[36] W. Chen, S. K. Gupta, and M. A. Breuer, "Analytic models for crosstalk delay and pulse analysis under non-ideal inputs," in *Proceedings International Test Conference 1997*. IEEE, 1997, pp. 809–818.

[37] S. Jayanthy and M. Bhuvaneswari, *Test Generation of Crosstalk Delay Faults in VLSI Circuits*. Springer, 2018.

[38] S. Sayil, A. B. Akkur, and N. Gaspard III, "Single event crosstalk shielding for cmos logic," *Microelectronics Journal*, vol. 40, no. 6, pp. 1000–1006, 2009.

[39] P. Heydari and M. Pedram, "Capacitive coupling noise in high-speed vlsi circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 478–488, 2005.

[40] M. A. Weiss, "Data structures and algorithm analysis in c++," 2014.

[41] A. Gomez and V. Champac, "A new sizing approach for lifetime improvement of nanoscale digital circuits due to bti aging," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2015, pp. 297–302.

[42] L. Joyanes Aguilar, "Programación en c++: Algoritmos: Algoritmo, estructurade datos y objetos." 2002.

[43] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[44] L. Joyanes, L. Joyanes, and I. Zahonero, *Estructuras de datos en C++*. McGraw-Hill/Interamericana de España, 2007.

[45] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to algorithms second edition," *The Knuth-Morris-Pratt Algorithm, year*, 2001.

[46] L. Zhao and J. Zhao, "Comparison study of three shortest path algorithm," in *2017 International Conference on Computer Technology, Electronics and Communication*. IEEE, 2017, pp. 748–751.

[47] M. Bhuvaneswari, *Application of evolutionary algorithms for multi-objective optimization in VLSI and embedded systems*. Springer, 2014.

[48] R. Gómez, A. Girón, and V. H. Champac, "A test generation methodology for interconnection opens considering signals at the coupled lines," *Journal of Electronic Testing*, vol. 24, no. 6, pp. 529–538, 2008.

[49] R. Gandikota, "Crosstalk noise analysis for nano-meter vlsi circuits," Ph.D. dissertation, The University of Michigan, 2009.

[50] W. Li, S. Reddy, and S. Sahni, "On path selection in combinational logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 1, pp. 56–63, 1989.

[51] A. Murakami, S. Kajihara, T. Sasao, R. Pomeranz, and S. M. Reddy, "Selection of potentially testable path delay faults for test generation," *International Test Conference*, pp. 376–384, 2000.

[52] W. Qiu and D. Walker, "An efficient algorithm for finding the k longest testable paths through each gate in a combinational circuit," *International Test Conference*, pp. 593–601, 2003.

[53] J. Bell, "Timing analysis of logic-level digital circuits using uncertainty intervals," Ph.D. dissertation, Texas A&M University, 1996.

[54] R. Tayade and J. A. Abraham, "Critical path selection for delay testing considering coupling noise," *Journal of Electronic Testing*, pp. 213–223, 2009.

[55] S. Tani, M. Teramoto, M. Fukazawa, and K. Matsuhiro, "Efficient path selection for delay testing based on partial path evaluation," *VLSI Test Symposium*, pp. 188–193, 1998.

[56] V. Pandey, S. Yadav, and P. Arora, "Retiming technique for clock period minimization using shortest path algorithm," in *2016 International Conference on Computing, Communication and Automation (ICCCA)*. IEEE, 2016, pp. 1418–1423.