UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Impacto del ruido por acople capacitivo en las violaciones de estado en nodos con operación subumbral en sistemas de 45, 32 y 22 nanómetros.

Trabajo de graduación en la modalidad de trabajo profesional presentado por:
Roberto Andrino Robles
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala
2016

Impacto del ruido por acople capacitivo en las violaciones de estado en nodos con operación subumbral en sistemas de 45, 32 y 22 nanómetros.

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de ingeniería

Impacto del ruido por acople capacitivo en las violaciones de estado en nodos con operación subumbral en sistemas de 45, 32 y 22 nanómetros.

Trabajo de graduación en la modalidad de trabajo profesional presentado por:
Roberto Andrino Robles
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala
2016

Vo. Bo.:

(f) _____

MSc. Carlos Alberto Esquit Hernández

Tribunal examinador:

(f) _____

MSc. Carlos Alberto Esquit Hernández

(f) _____

Ing. Guilmar Zadir EscobarRoch

(f) _____

Ing. Jonathan Alberto de los Santos Chonay

Fecha de aprobación: Guatemala, 8 de diciembre de 2016

Índice

Página

Índice de cuadros

Página

Índice de figuras

Página

RESUMEN

En 1958 Jack Kilby construyó el primer circuito integrado, que consistía en dos transistores, capacitores y resistencias, con interconexiones soldadas a mano. Actualmente, un procesador de Intel tiene varios miles de millones de transistores a nivel nanoscópico. Conforme se reduce el tamaño de los transistores, también se vuelven más rápidos, eficientes y baratos (Harris & Weste, CMOS VLSI Design: A Circuits and Systems Perspective, 2011). Sin embargo, esta reducción de tamaño implica que las interconexiones se acercan más y los acoples capacitivos incrementan. Este trabajo aborda este problema pues busca encontrar cómo se ve afectado el funcionamiento de los circuitos debido al ruido por acople capacitivo con distintas tecnologías. Específicamente se hizo un análisis de los circuitos operando en condiciones subumbral, pues es una técnica comúnmente utilizada para ahorrar potencia con la restricción de que la operación es más lenta.

Este trabajo consiste en hacer un análisis temporal y de integridad de las señales en circuitos con miles de transistores en operación subumbral, y con distintas tecnologías. Para esto se requirió trabajar con varias herramientas de Synopsys en la plataforma de distribución de Linux llamada CentOS. Se instalaron las versiones más recientes de las herramientas requeridas y se modificaron los *runsets* y se crearon *scripts* conforme fue necesario. Para esto se consultaron los manuales de usuario y se trabajó acorde a las indicaciones que estos proporcionaban. Cuando se encontró el flujo de análisis apropiado se logró usar las herramientas de Synopsys exitosamente.

La segunda parte del trabajo consiste en usar el flujo de análisis para analizar un circuito Sumador/Restador *Ripple-carry* de 32 *bits*. Para esto fue necesario obtener las curvas de transferencia DC y de rechazo de ruido para una compuerta inversora con voltajes de operación tanto en subumbral como en súperumbral con HSPICE. Después se obtuvo la curva de rechazo de ruido para un D Flip-flop de flanco negativo. Luego se estudió la inyección del ruido de dos compuertas inversoras que fueran agresores a una compuerta inversora víctima con longitud variable de interconexión. Estos resultados permitieron definir lo que sería una violación de estado para el Sumador/Restador *Ripple-Carry* de 32 *bits*. Finalmente se usó la herramienta NanoTime de Synopsys para analizar el circuito y se halló una mayor cantidad de violaciones de estado en operación subumbral, pero ningún error en las salidas del circuito.

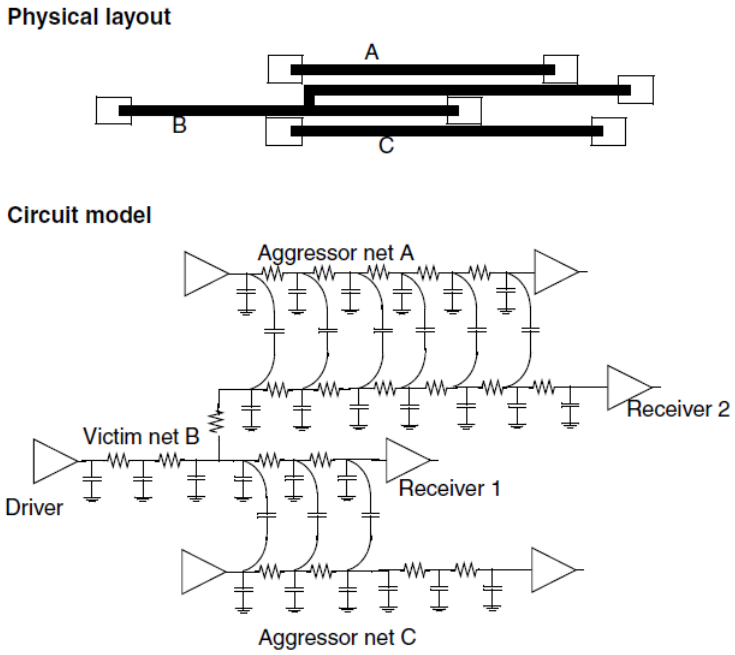
I. INTRODUCCIÓN

Las aplicaciones de bajo rendimiento que requieren un voltaje de alimentación bajo están explorando la posibilidad de usar dispositivos CMOS cuando están en operación subumbral. En operación subumbral el voltaje de alimentación es menor que el voltaje umbral del dispositivo y las capacitancias de entrada son menores en comparación con operación superumbral. La combinación de los efectos de capacitancias reducidas de entrada y el voltaje de operación reducido resulta en un menor consumo de potencia.

El ruido por acople capacitivo en un diseño se define como el pulso de voltaje inducido en una interconexión que no está conmutando, llamada la víctima, por la actividad de conmutación de las interconexiones vecinas, llamadas agresores (Nanua & Blaauw, 2007).

En la siguiente figura se puede observar la representación de las capacitancias de acople y la vista del *layout* de una porción de un circuito integrado, donde se puede observar el modelado de las capacitancias de acople y los parásitos de las interconexiones.

Figura 1. Modelo detallado de ruido por acople capacitivo en las *nets*.



(Synopsys, NanoTime, 2016)

Una herramienta de extracción de parásitos divide cada *net* en *subnets* y representa las resistencias y capacitores distribuidos como un conjunto de resistencias y capacitores discretos.

Al tener un *driver* débil, o que conduce poca corriente, en la víctima el circuito es más susceptible al ruido por capacitancias de acople. Por otra parte, un *driver* fuerte, que conduce bastante corriente, puede causar una transición rápida en la *net* del agresor e inducir un pulso de ruido por acople capacitivo en la víctima (Nanua & Blaauw, 2007).

En el artículo de Nanua y Blaauw (2007:2) y Fuketa y otros (2012:1) mencionan que el efecto del ruido por capacitancias de acople de un dispositivo en operación subumbral no es conocido y no existen trabajos previos sobre este tema. Este trabajo aborda este problema al analizar varios circuitos en operación subumbral, haciendo un análisis temporal y de integridad de las señales con la herramienta NanoTime de Synopsys.

La metodología que se usó para abordar el problema mencionado se puede resumir como: 1) Instalación de las herramientas necesarias. 2) Seguimiento del flujo de diseño para las verificaciones y extracción de parásitos del *layout* del circuito. 3) Realización de pruebas con PrimeTime. 4) Elaboración del *script* de NanoTime. 5) Pruebas de validación de resultados. 6) Análisis de los resultados.

Entre las conclusiones más importantes se puede mencionar que se realizó y validó un *script* de NanoTime al comparar los resultados que reportaba con lo esperado acorde a lo reportado por Synopsys para sus *standard-cells*, y se logró detectar errores en el tiempo y violaciones de ruido como resultado de que algunas señales tenían un *slack* negativo.

II. OBJETIVOS

A. Objetivo general

Estudiar el impacto del *crosstalk* para sistemas de 45, 32 y 22 nanómetros en términos de rechazo a ruido y violaciones de estado en nodos con operación *subthreshold*.

B. Objetivos específicos

1. Analizar la teoría de operación *subthreshold* y *crosstalk* en estas condiciones.
2. Obtener un circuito de *benchmark* en HSPICE para realización de pruebas de *crosstalk*.
3. Utilización de las herramientas necesarias para análisis de *crosstalk* y violaciones de estado en nodos.
4. Obtener la curva de rechazo de ruido para diversas tecnologías de 45, 32 y 22 nanómetros.
5. Cuantificar las violaciones debido al *crosstalk* en el circuito de *benchmark* para cada una de las tecnologías de 45, 32 y 22 nanómetros.
6. Realizar simulaciones con las herramientas de análisis.

III. JUSTIFICACIÓN

El estudio de los circuitos en operación subumbral es de interés por el ahorro de potencia. Sin embargo, es de importancia conocer el efecto del ruido por acople capacitivo, o *crosstalk*, en este tipo de operación porque puede causar que el sistema falle. Debido a que el *crosstalk* es un fenómeno inevitable, la comprensión de sus efectos en distintas tecnologías permitiría en un futuro desarrollar sistemas en operación subumbral con mayor robustez.

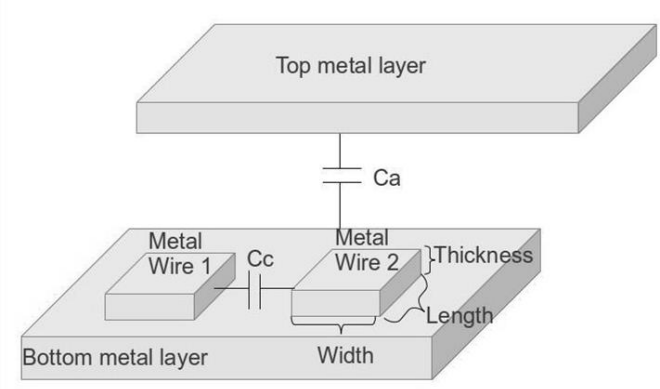
No se han realizado estudios sobre el efecto del ruido de *crosstalk* en operaciones de subumbral según las referencias consultadas. Además, los estudios disponibles públicamente son sólo para tecnologías de 65 nanómetros y 40 nanómetros.

IV. MARCO TEÓRICO

A. Ruido por acople capacitivo.

En el diseño de chips, la potencia, el rendimiento y el área son temas muy importantes que se deben considerar. La reducción de las tecnologías de transistores en los últimos años ha hecho que cada vez existan más retos por resolver, como el ruido por acople capacitivo entre interconexiones.

Figura 2. Propiedades de las interconexiones de metal con capacitancias de acople (Cc) y la zona capacitiva. (Ca).



(Gayathri, Why Coupling Capacitance Increases When CMOS Technology Shrinks, 2012)

Cuando las tecnologías son más pequeñas, el ancho decrece (Figura 1) y entonces el área de la interconexión de metal decrece. Para mantener el flujo de corriente, cuando el ancho decrece, el espesor tiene que incrementar. El espaciamiento entre las interconexiones de metal también decrece por lo que resulta en un aumento de las capacitancias de acople. Lo descrito anteriormente se puede visualizar con la ecuación de capacitancia:

$$C = \frac{\varepsilon A}{d}. \qquad (1)$$

En donde:

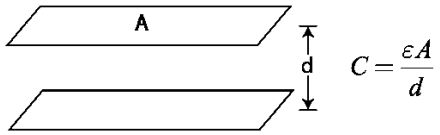
C es la capacitancia en faradios.

ε es la permitividad absoluta del dieléctrico.

A es el área de las placas en metros cuadrados.

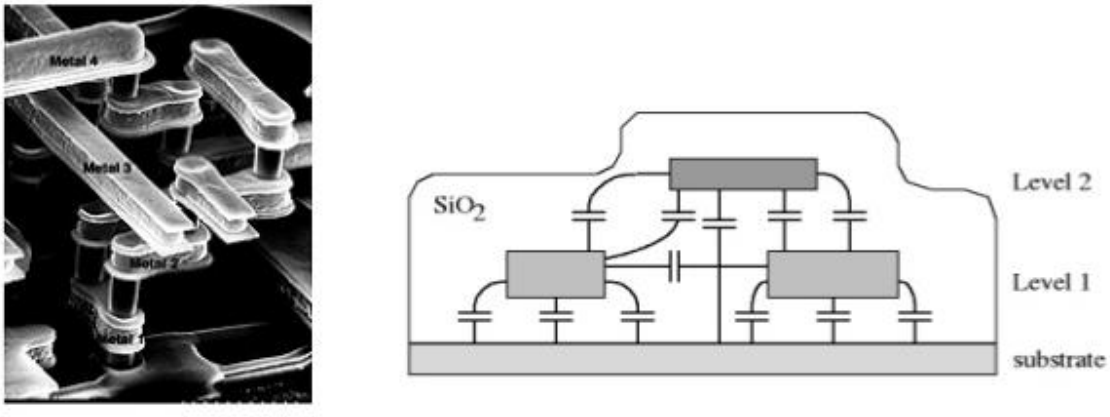
d es la distancia entre las placas en metros.

Figura 3. Ecuación para un capacitor de placas paralelas.



El ruido por acople capacitivo es una forma particular de acople electromagnético que se da al tener elementos agrupados como capacitancias intercapas o capacitancias de acople.

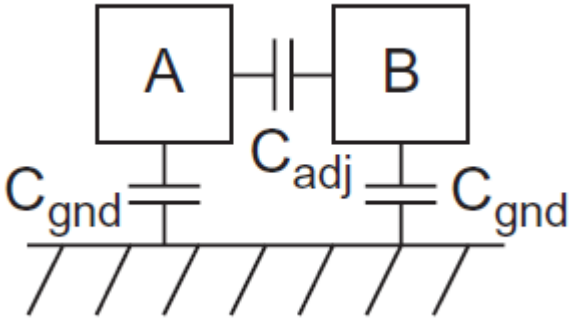
Figura 4. Capacitancias de intercapas en un diseño MOSFET.



(Kamakoti & Balachandran, 2013)

Las interconexiones tienen capacitancia hacia sus vecinos y al sustrato, que sirve de tierra (Figura 4). Este ruido por acople capacitivo causa retardos en las señales cuando los vecinos y las interconexiones están conmutando.

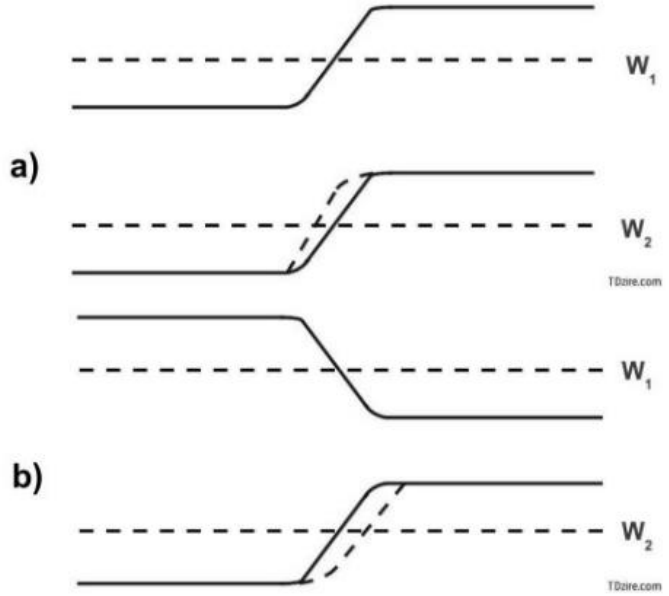
Figura 5. Capacitancias con vecinos adyacentes y hacia tierra.



(Harris & Weste, 2011:222)

Supóngase que hay dos interconexiones, W1 (agresor) y W2 (víctima). La señal que está pasando a través de W2 puede ser afectada por interferencia eléctrica de la señal que está pasando por W1. El primer efecto que ocurre es que cuando la señal de W2 está cambiando de estado, su tiempo de transición puede ser afectado por la dirección de la transición en W1. Si ambas tienen la misma dirección de cambio, W2 tendrá una transición más rápida que si no hubiera un agresor. Si la dirección de W1 es opuesta a la de W2 hará que W2 cambie más despacio.

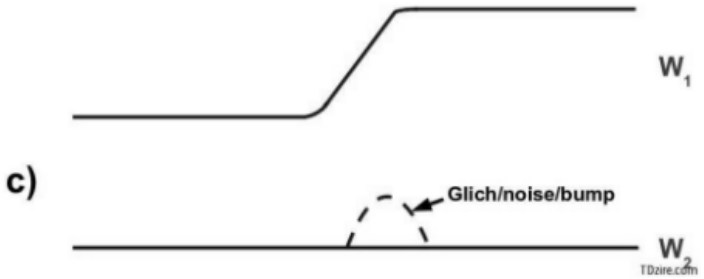
Figura 6. Efecto causado por ruido por acople capacitivo entre el agresor (W1) y la víctima (W2).



(Gayathri, What is crosstalk analysis/signal integrity analysis in static timing analysis., 2012)

El segundo efecto se da cuando W2 está en estado estable y W1 hace una transición. Esto produce un *glitch* en la señal de W2 debido al ruido que se inyecta por el acople capacitivo.

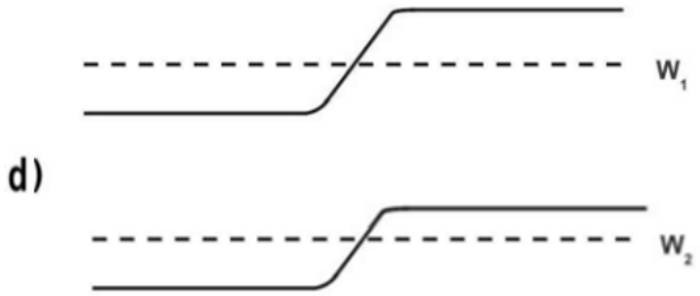
Figura 7. Efecto causado por el ruido por acople capacitivo entre el agresor (W1) y la víctima (W2).



(Gayathri, What is crosstalk analysis/signal integrity analysis in static timing analysis., 2012)

Estos efectos producidos por el ruido por acople capacitivo pueden ser resueltos aumentando el tamaño del *driver* de la víctima o disminuyendo el tamaño del *driver* del agresor. También se puede evitar si se incrementa el espaciamiento entre las interconexiones metálicas en el diseño físico.

Figura 8. Efecto causado por ruido por acople capacitivo entre el agresor (W1) y la víctima (W2) sin *driver*.



(Gayathri, What is crosstalk analysis/signal integrity analysis in static timing analysis., 2012)

En la figura anterior se muestra el comportamiento de una víctima sin *driver* cuando un agresor conmuta. El nivel de voltaje al que llega la víctima se define por la ecuación (3).

El Factor de Acoplamiento Miller, por sus siglas en inglés MCF (*Miller Coupling Factor)*, describe cómo la capacitancia de las interconexiones adyacentes se multiplica para encontrar la capacitancia efectiva que se debe cargar cuando un nodo conmuta. Algunos diseñadores utilizan un MCF = 1.5 como un compromiso estadístico cuando se estiman los retardos de propagación antes de tener la información del *layout*.

En la Figura 9 se resume el cambio en el voltaje (ΔV) entre los vecinos A y B de la Figura 4. Con la información de la capacitancia adyacente se puede conocer la carga que se entrega al capacitor de acople. La relación es la siguiente:

$$Q = C_{adj}\Delta V \qquad (2)$$

(Harris & Weste, 2011:222).

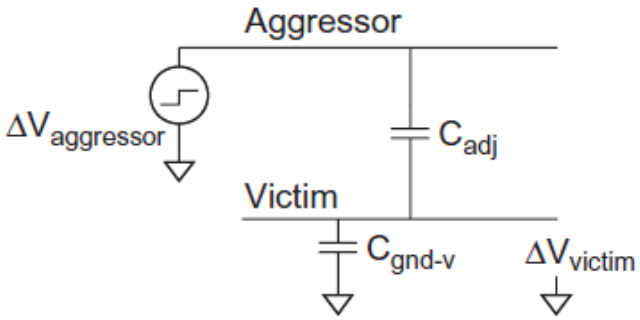
Figura 9. Dependencia de la capacitancia efectiva en la dirección de conmutación de las señales en los vecinos A (agresor) y B (víctima).

B	ΔV	$C_{eff(A)}$	MCF
Constant	V_{DD}	$C_{gnd} + C_{adj}$	1
Switching same direction as *A*	0	C_{gnd}	0
Switching opposite to *A*	$2V_{DD}$	$C_{gnd} + 2C_{adj}$	2

(Harris & Weste, 2011:223)

Si se tiene el caso en el que la interconexión A está cambiando mientras B se mantiene constante y la víctima no tiene ningún *driver*, el circuito se puede modelar como un divisor de voltaje capacitivo para aproximar el ruido en la víctima. La magnitud del cambio de voltaje en el agresor, ΔV, es VDD si se está usando lógica CMOS.

Figura 10. Acoplamiento a la víctima sin *driver*.

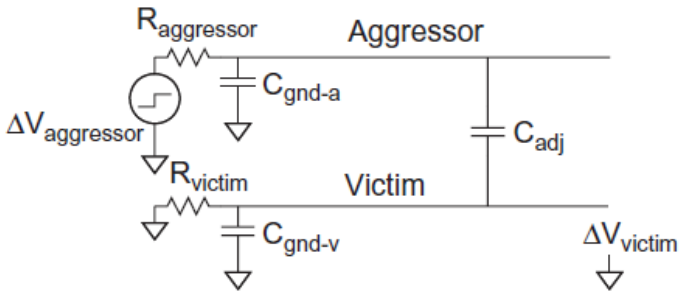


(Harris & Weste, 2011:223).

$$\Delta V_{victim} = \frac{C_{adj}}{C_{gnd-v} + C_{adj}} \Delta V_{aggressor} \qquad (3)$$

Si la víctima tiene un *driver*, este le suministrará la corriente para oponerse al cambio y reducirá el ruido en la víctima. El *driver* es modelado como una resistencia a tierra, si está imponiendo un '0' lógico, o a VDD, si está imponiendo un '1' lógico. El tamaño de esta resistencia depende del tamaño del *driver*. Un *driver* más grande se modela como una resistencia más pequeña porque conduce más corriente.

Figura 11. Acoplamiento a la víctima con *driver*.



(Harris & Weste, 2011:223).

El capacitor de acople de la víctima a tierra se descarga a través de la resistencia del *driver* de la víctima. El voltaje máximo alcanzado por el ruido se vuelve dependiente de la razón de la constante τ del agresor a la de la víctima.

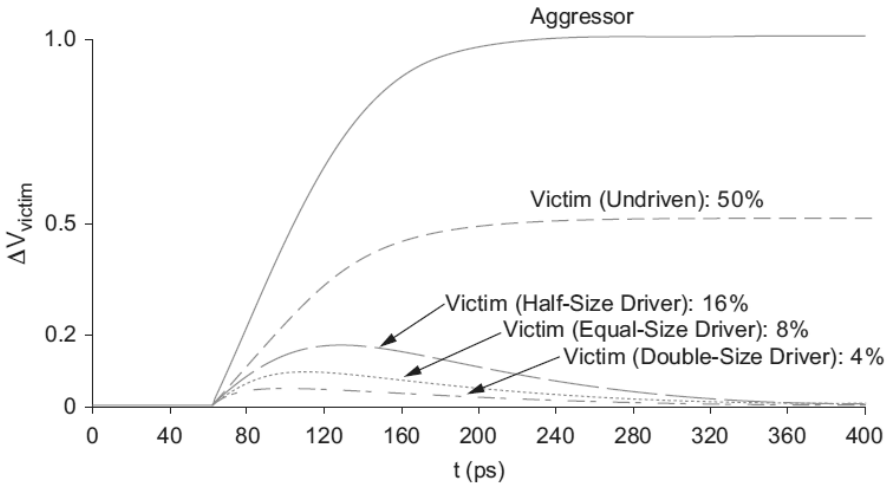
$$\Delta V_{victim} = \frac{C_{adj}}{C_{gnd-v} + C_{adj}} \frac{1}{1+k} \Delta V_{aggressor} \quad (4)$$

Donde

$$k = \frac{\tau_{aggressor}}{\tau_{victim}} = \frac{R_{aggressor}(C_{gnd-a} + C_{adj})}{(C_{gnd-v} + C_{adj})} \quad (5)$$

En la siguiente figura se muestra una simulación del efecto de un agresor sobre una víctima cuando la víctima no tiene *driver*, y cuando tiene un *driver* de la mitad del tamaño, del mismo tamaño, y del doble del tamaño del *driver* del agresor. En este ejemplo se tomó $C_{adj} = C_{gnd}$.

Figura 12. Acoplamiento a la víctima con *driver*.



(Harris & Weste, 2011:224)

Cuando la víctima no tiene un *driver* el ruido es permanente, es decir, el voltaje en la víctima nunca regresará a su nivel previo a la inyección del ruido. Esto ocurre porque la víctima no tiene una resistencia por dónde descargar su capacitor de acople a tierra. Si la víctima tiene *driver,* este restaura a la víctima a su voltaje inicial. Un *driver* que es más fuerte, o grande, se opone más al ruido inyectado por el acoplamiento y el ruido resulta tener un pico menor.

B. Operación subumbral.

La mayoría de fuentes de fuga surgen cuando hay una corriente cuando los transistores están apagados. Cuando $|V_{gs}| < |V_t|$ y el transistor entra en modo corte, la corriente decae de forma exponencial en lugar de pasar a cero de forma abrupta. A esto se le llama conducción subumbral. La operación subumbral de un dispositivo es cuando opera con un voltaje de alimentación menor a su voltaje umbral, por lo que siempre

está en modo corte. Aunque en la teoría se supone que la corriente de conducción del transistor es cero cuando está en modo corte, en la práctica resulta que tiene corrientes de fuga. La operación subumbral usa estas corrientes de fuga para descargar o cargar los nodos hasta el '0' o '1' lógico, respectivamente. La corriente de fuga de la operación subumbral incrementa exponencialmente con V_{ds} y V_{gs} como se muestra en la siguiente ecuación (Nanua & Blaauw, 2007:2).

$$I_{sub} = I_o e^{\frac{V_{gs}-V_t}{nV_{th}}}(1 - e^{\frac{V_{ds}}{V_{th}}}) \quad (6)$$

Donde

V_{th} es el voltaje termal definido como kT/q.

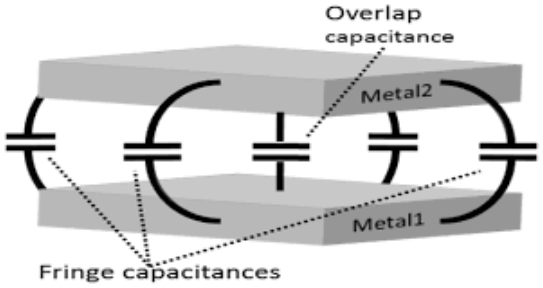
n es el factor de la pendiente de la operación subumbral.

I_o está definida como

$$I_0 = \mu_o C_{ox} \frac{W}{L}(n-1)V_{th}^2 \quad (7)$$

La capacitancia de entrada del dispositivo es la suma de las capacitancias intrínsecas (C_i) y las capacitancias parásitas (C_p) pues están conectadas en paralelo (Nanua & Blaauw, 2007:2). Los componentes de las capacitancias parásitas son las capacitancias de *overlap* y capacitancias de *fringe*, que están conectadas en paralelo cómo se muestra en la siguiente figura.

Figura 13. Capacitancias de *overlap* y *fringe*.



(Shomalnasab, Heys, & Zhang)

Las componentes de la capacitancia intrínseca son la capacitancia de óxido y la capacitancia de deplexión conectadas en serie.

$$C_i = \frac{C_{ox} * C_{dep}}{C_{ox} + C_{dep}} \quad (8)$$

Al comparar esto con los dispositivos en operación superumbral, cuya capacitancia intrínseca es sólo C_{ox}, se puede notar que la capacitancia intrínseca se reduce en operación subumbral y como consecuencia disminuye la capacitancia de entrada del dispositivo.

C. Galaxy Custom Compiler.

La herramienta de Galaxy Custom Compiler es similar a la herramienta de Galaxy Custom Designer que está disponible en las computadoras del laboratorio de Electrónica de la Universidad del Valle de Guatemala. Para instalar la herramienta de Synopsys se siguieron los pasos descritos por de los Santos (2014:36). Se decidió usar Galaxy Custom Compiler porque tiene una interfaz amigable para el usuario y permite realizar el diseño del *layout* y esquemáticos del circuito que se analizará. Además esta herramienta permite usar NanoTime con el ambiente de diseño y hacer el análisis a nivel de transistor de una forma gráfica. Al terminar el diseño del *layout* y del esquemático de los circuitos, este debe pasar por las siguientes verificaciones para asegurar que no exista ninguna violación en el diseño.

1. Análisis de DRC (Design Rule Checking). El principal objetivo del *Design Rule Checking* es tener una forma de asegurar la confiabilidad del diseño. Si existe alguna violación de las reglas de diseño, no se podrá manufacturar. El DRC analiza el diseño del *layout.* Algunas de las reglas de diseño incluyen: espaciamiento *active to active, well to well,* espacio mínimo del canal de los transistores, ancho mínimo del metal, entre otras. Se recomienda leer los manuales de usuario para revisar las reglas de diseño (Synopsys, Custom Compiler Help, 2016).

Para configurar este análisis se siguieron los pasos descritos por de los Santos (2014:52). Se procedió a realizar el siguiente análisis (LVS) luego de haber completado exitosamente el DRC.

2. Análisis de LVS (Layout Versus Schematic). El haber hecho exitosamente el análisis DRC no asegura que el diseño represente al circuito que se quiere fabricar. Para ello se necesita el análisis de *Layout Versus Schematic* (LVS), para revisar que el diseño físico se comporte igual que el circuito representado en el esquemático.

La verificación del LVS incluye tres pasos. En el primer paso la herramienta extrae de una base de datos el dibujo del *layout* para reconocer vías, metales, interconexiones, y demás detalles físicos del diseño. Luego el programa genera un *netlist* que representa al *layout* que se está analizando. Por último compara el *layout* extraído con el *netlist* del esquemático del circuito. Si ambos *netlist* son iguales el circuito pasa este análisis y se muestra en la pantalla una ventana que dice "*LVS clean*". De no pasar este análisis, se debe buscar y arreglar los errores que se despliegan en una ventana de reporte que despliega la herramienta del LVS. Comúnmente se dan errores de *component/parameter mismatch*. Consultar el manual de usuario (Synopsys,

Custom Compiler Help, 2016) da una mejor claridad sobre cómo arreglar los errores para continuar con el siguiente análisis.

Para configurar el análisis se siguieron los pasos descritos por de los Santos (2014:52). Además, se colocó un texto a cada interconexión del *layout* con el mismo nombre que tenía la interconexión en el esquemático. De esta forma se redujo la probabilidad de que fallara la verificación.

3. Análisis de LPE (Layout Parasitic Extraction). Al pasar la prueba del *Layout Parasitic Extraction* se genera un archivo que puede ser DSPF, SPEF, SBPF u OpenAccess. Este documento contiene información de todas las capacitancias, inductancias y resistencias parásitas que se encuentran a partir de un *runset report file* que se genera en el análisis de LVS. Por eso es necesario haber completado los dos análisis anteriores exitosamente antes de hacer la extracción de parásitos. El LPE es el análisis que más importancia tiene para poder realizar cualquier estudio sobre el funcionamiento del circuito. En el presente trabajo, hacer la extracción de parásitos es crucial para poder usar el archivo SPF que se genera con la herramienta de NanoTime para verificar el impacto de las capacitancias de acople cuando el circuito se encuentra en operación subumbral.

Para configurar el análisis se siguieron los pasos descritos por de los Santos (2014:56) con la única variación de que en la sección de *Runset* no se agregó el archivo por defecto que rige cómo se ejecuta la extracción de parásitos. Se creó un nuevo archivo con extensión ".cmd" con la configuración apropiada para la compatibilidad con la herramienta de NanoTime. La aplicación que se integra al entorno de Custom Compiler es Synopsys StarRC, por lo que los comandos agregados son comandos de StarRC que están disponibles en el manual de usuario de la herramienta (Synopsys, StarRC User Guide and Command Reference, 2016), donde se detalla el funcionamiento de cada uno.

D. PrimeTime Suite.

La herramienta de PrimeTime Suite realiza análisis de tiempo estático a nivel de compuerta de *chips* completos, lo que es una parte esencial en el proceso para diseñar *chips*. Esta herramienta analiza el desempeño en el tiempo de un circuito al revisar todos los caminos que una señal puede tomar en búsqueda de violaciones de tiempo, sin usar simulaciones lógicas o vectores de prueba (Synopsys, PrimeTime Online Help, 2016). Las características de PrimeTime y todos los análisis que se pueden realizar son bastantes por lo que se sugiere leer el manual de usuario (Synopsys, PrimeTime Online Help, 2016).

1. Requerimientos. Antes de realizar un análisis temporal se necesita leer el diseño y las librerías lógicas, y unirlas. PrimeTime sólo puede leer los siguientes formatos de archivos (Synopsys, PrimeTime Online Help, 2016):

Cuadro 1. Formatos soportados por PrimeTime

Datos de entrada	Formatos de archivos soportados
Datos de diseño	Base de datos binaria (.db)
	Milkyway
	Base de datos lógica de Synopsys (.ddc)
	Verilog
	VHDL
Librerías lógicas	Base de datos binaria (.db)
	Librería de Synopsys (.lib)

Además, el archivo obtenido al realizar el LPE (.SPF) debe estar a nivel de compuertas y no de transistores, por lo que se requiere contar con la librería donde se caracterice el comportamiento de las compuertas.

E. NanoTime.

Esta herramienta para analizar en tiempo estático puede hacer verificaciones de tiempo y de integridad de las señales sobre circuitos completos o sólo ciertos bloques. Permite detectar y corregir violaciones por *timing* y por ruido. NanoTime posee muchas características y para ello es mejor referirse al manual de usuario (Synopsys, NanoTime, 2016).

1. Requerimientos. Para usar NanoTime efectiva y correctamente se debe notar que tiene ciertas limitaciones en los diseños que puede analizar. Estas limitaciones incluyen:

- Circuitos analógicos, excepto algunos circuitos usados en memorias SRAM embebidas.
- Circuitos con lógica BiCMOS.
- Lógica Complementary Pass-transistor.
- Memoria Flash o content-addressable.

Adicionalmente, si se analizan algunos diseños no convencionales, se recomienda que los resultados se validen usando HSPICE porque puede ocurrir que NanoTime reporte resultados inesperados. Estos diseños incluyen:

- Topologías con características poco lineales de conmutación.
- Circuitos con transiciones muy grandes en sus salidas.
- Circuitos que usan el *source* o el *drain* de los transistores de paso como entradas, entre otros.

El análisis temporal que se realice con NanoTime requiere un archivo HSPICE o de Verilog para leer el circuito y los modelos de los transistores. Si ya se cuenta con modelos de *timing* para los subcircuitos, se

pueden leer si están en formato de Librería de Synopsys (.lib) o Base de Datos Binaria (.db). Los archivos de parásitos que se usen pueden estar en formato DSPF, SPEF o SBPF (Synopsys, NanoTime, 2016).

F. Análisis de la integridad de las señales con NanoTime.

La integridad de una señal en un circuito integrado se puede deteriorar debido al ruido por acople capacitivo. El análisis de NanoTime incluye efectos del ruido por acople capacitivo en los tiempos de transición, de llegada, tiempo de *slack,* y los niveles de la señal en estado estacionario (Synopsys, NanoTime, 2016).

Los tipos de análisis de la integridad de las señales son, acorde a (Synopsys, NanoTime, 2016):

- Análisis de retardo: cambios en los tiempos de llegada en las interconexiones de la víctima debido a la transición de la interconexión del agresor.
- Análisis de ruido: cambios del potencial en la interconexión de la víctima debido a la transición de la interconexión del agresor.
- Análisis de ruido por *fanout:* una extensión del análisis de ruido; el efecto del ruido sobre el *fanout* de una interconexión de la víctima.

En NanoTime, el análisis de la integridad de las señales está diseñado para ser pesimista. Se supone que las interconexiones del agresor y de la víctima cambian en una dirección que maximiza el pesimismo, por ejemplo (Synopsys, NanoTime, 2016):

- Para el análisis de *timing* mínimo se supone que el agresor cambia en la misma dirección que la víctima, haciendo el retardo de la víctima lo más pequeño posible.
- Para el análisis de *timing* máximo, se supone que el agresor cambia en dirección opuesta a la víctima, haciendo que el retardo de la víctima sea lo más extendido posible.

NanoTime determina el peor de los casos en los valores de retardo y utiliza esta información para calcular y reportar el *slack* de cada señal. La herramienta también reporta las fuentes de retardo y la cantidad de retardo por el acople capacitivo. Esto facilita que se modifique el diseño para corregir errores (Synopsys, NanoTime, 2016).

NanoTime calcula el efecto del ruido de las siguientes fuentes:

- Ruido por acople capacitivo: Considera la capacitancia de acople entre el agresor y la víctima, la ventana de llegada de la transición del agresor, las características del *driver* de del agresor, y las características de la resistencia en estado estacionario de la interconexión de la víctima.

- Ruido por propagación. La propagación del ruido en la víctima es causada por ruido en la entrada de la celda que está haciendo *driving* a la víctima. NanoTime calcula el ruido que se propaga a la salida de la celda, dado la cantidad de ruido en la entrada y la carga en la salida de la celda.
- Ruido inyectado por el usuario. El usuario puede inyectar ruido en cualquier *pin* o puerto del diseño. Además, puede reescribir o adicionar ruido a cualquier ruido existente que NanoTime haya calculado. El ruido inyectado puede afectar a la cantidad del ruido por *fanout*. Sin embargo, NanoTime no soporta el ruido de *fanout* definido por el usuario.

NanoTime analiza cuatro tipos de bultos de ruido típicamente causados por la transición del agresor: por encima de la línea de tierra (por encima del voltaje correspondiente a un '0' lógico, llamado AL), por debajo de la línea de tierra (por debajo del '0' lógico, llamado BL), por encima de la línea de alimentación (por encima del '1' lógico, llamado AH), y por debajo de la línea de alimentación (por debajo del '1' lógico, llamado BH). Los bultos de ruido entre dos líneas de voltaje que ocurren por encima del '0' lógico y por debajo del '1' lógico pueden causar fallos si se exceden del umbral lógico de la tecnología. Si ocurre ruido fuera de las líneas de voltaje, por debajo del '0' lógico y por encima del '1' lógico, podría polarizar los *gates* de los transistores en las entradas de Flip-flops y Latches y causar que se graben valores incorrectos.

Por defecto, NanoTime asume el peor caso de la combinación de estados del agresor y la víctima y calcula e inyecta ondas de ruido en el nodo de la víctima. El máximo nivel de ruido es comparado con los márgenes especificados por el usuario para determinar si existe una violación.

G. Circuitos elegidos.

Se usó un sumador/restador completo de 32 *bits* diseñado por de los Santos (de los Santos, 2014) debido a que es un circuito grande. El circuito tenía 3208 nets. Sin embargo, el circuito sería un módulo básico de un sistema como, por ejemplo, un procesador, por lo que es de baja complejidad. Para pruebas iniciales se usaron circuitos básicos para verificar que los resultados obtenidos por NanoTime, herramienta que se escogió usar para hacer un análisis a nivel de transistores, fueran coherentes con la teoría y cercanos a lo esperado en la realidad. Los circuitos fueron: una cadena de ocho compuertas inversoras, un comparador de identidad de dos *bits* y un D Flip-flop de flanco positivo. En la sección de Metodología se muestran los circuitos diseñados en Custom Compiler.

V. METODOLOGÍA

A. Instalación de herramientas.

En el laboratorio de Electrónica de la Universidad del Valle ya se encuentra instalado el sistema operativo CentOS para trabajar con las herramientas de Synopsys. Galaxy Custom Designer, el instalador de herramientas de Synopsys versión 3.1 y la librería de 32 nanómetros se encuentran listos para usarse por el trabajo realizado previamente por de los Santos (de los Santos, 2014). Para instalar cualquier herramienta de Synopsys es importante tener una versión actualizada del instalador. En la página de solvnet.synopsys.com están disponibles versiones actualizadas de todas las herramientas. Es importante tener el usuario y contraseña para poder ingresar a la página de Synopsys y descargar las herramientas y manuales o hacer consultas. Preguntar al departamento de Ing. Electrónica por estos datos ya que son confidenciales. En esta sección se explicarán los pasos a seguir para la instalación de las herramientas de PrimeTime, StarRC, Custom Compiler, NanoTime y de las librerías de las tecnologías de 90 nanómetros.

1. Actualización del instalador de aplicaciones de Synopsys. Para instalar versiones actuales de las herramientas, es necesario actualizar el instalador acorde a los siguientes pasos.

a. Ingresar con el usuario y contraseña al sitio de solvnet.synopsys.com
b. Descargar la actualización del instalador del sitio de Synopsys:

Figura 14. Directorio donde se encuentra el instalador de la actualización de Synposys Installer.

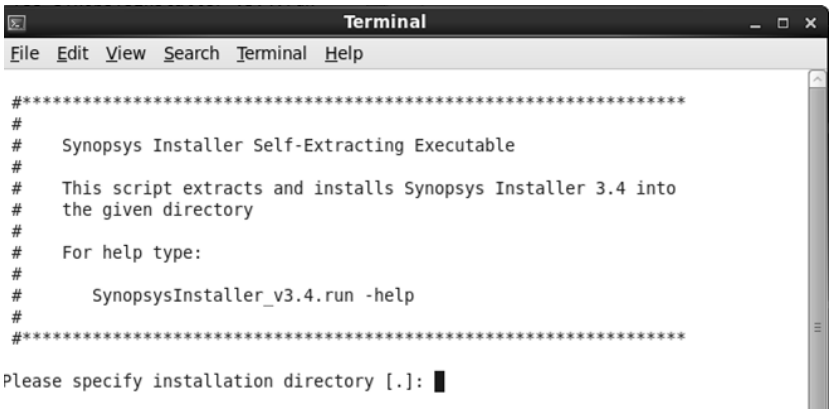
Nótese que la última versión al momento es la 3.4. En trabajos futuros es posible que existan nuevas versiones.

c. Abrir la línea de comandos en la carpeta donde se encuentre la actualización y escribir el siguiente comando para verificar y asegurar que el archivo sea ejecutable:

```
chmod 755 SynopsysInstaller_v3.4.run
```

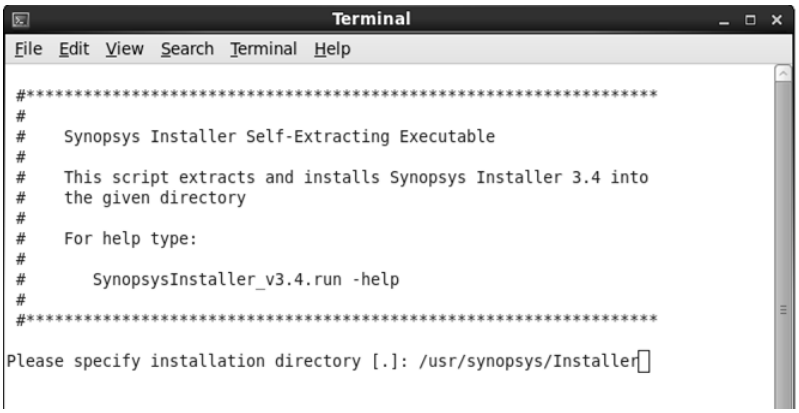
d. Comprobar el paso 3 abriendo desde la terminal el instalador de la actualización.

Figura 15. Ejecución de la actualización de Synopsys Installer.



e. Luego se debe especificar el directorio de instalación o reescribir en el directorio que contiene la versión anterior (/usr/synopsys/Installer).

Figura 16. Directorio del Instalador que se desea actualizar.



f. Comprobar que el instalador se actualizó correctamente yendo a la dirección donde se instaló y abriendo desde la terminal el archivo “setup.sh”.

Figura 17. Abrir el archivo “setup.sh” y verificar la versión del instalador.

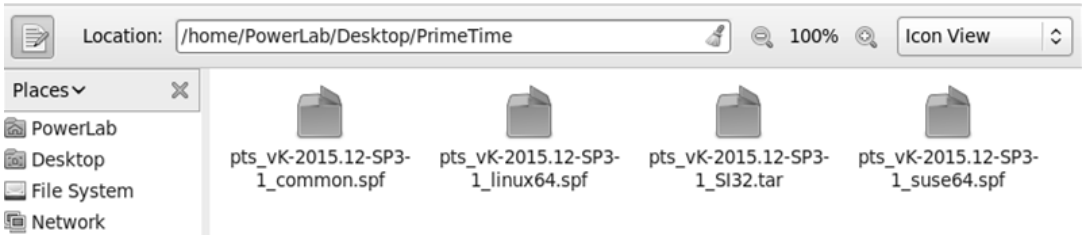
Figura 18. Verificación de la versión instalada de Synopsys Installer.

Con el instalador actualizado ya se pueden instalar las demás herramientas.

2. Instalación de PrimeTime. Inicialmente se pensó hacer el análisis de *timing* y de violaciones de estado en nodos con esta herramienta. PrimeTime permite determinar el impacto del ruido por capacitancias de acople capacitivo cuando el circuito se encuentra en operación subumbral, pero el análisis se realiza a nivel de compuerta. Los pasos para la instalación de la herramienta son los siguientes:

a. Ingresar con el usuario y contraseña al sitio de solvnet.synopsys.com
b. Descargar los instaladores de PrimeTime del sitio de Synopsys.

Figura 19. Archivos necesarios para la instalación de PrimeTime.



c. Abrir el instalador de aplicaciones Synopsys ejecutando en la terminal el archivo “setup.sh” que se encuentra en la dirección: /usr/synopsys/installer

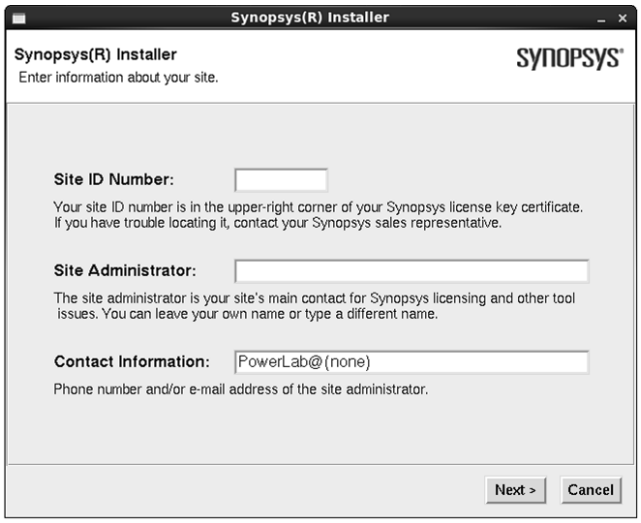
Figura 20. Ejecución del instalador de aplicaciones de Synopsys.

d. Después de correr el archivo “setup.sh” aparece la interfaz gráfica del instalador de Synopys.

Figura 21. Interfaz gráfica del instalador de aplicaciones de Synopsys

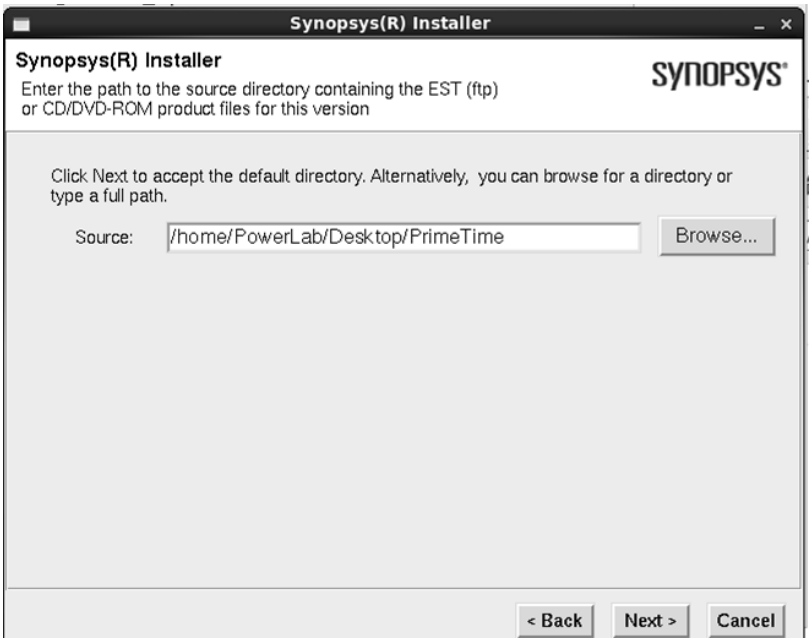
e. Al dar clic en el botón *Start* aparece otra pantalla en la que es obligatorio llenar todos los campos requeridos para iniciar el proceso de instalación.

Figura 22. Información obligatoria para empezar la instalación



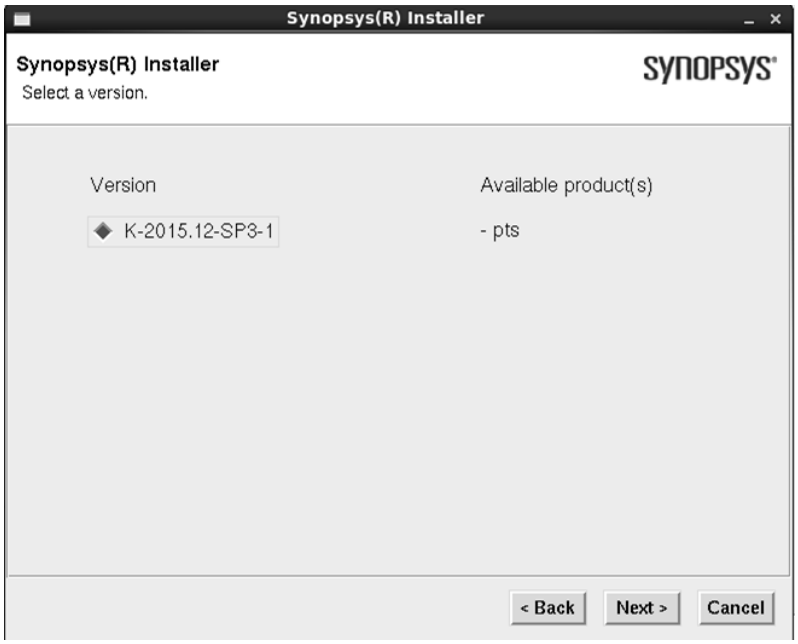
Luego en el botón de *Browse* buscar la carpeta en donde se encuentran los archivos descargados. En este caso están en una carpeta que se llama PrimeTime ubicada en el escritorio (como se indica en el paso 2).

Figura 23. Directorio en donde se encuentran los instaladores de PrimeTime.



f. Seleccionar la versión de PrimeTime a instalar.

Figura 24. Versión de PrimeTime que se instalará.

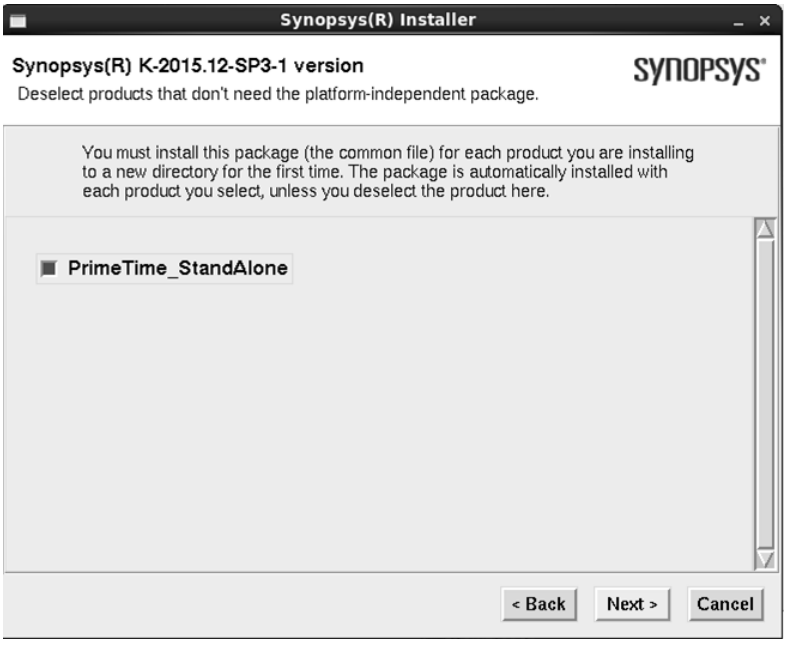


g. Seleccionar el producto que se desea instalar.

Figura 25. Selección del producto de PrimeTime a instalar.

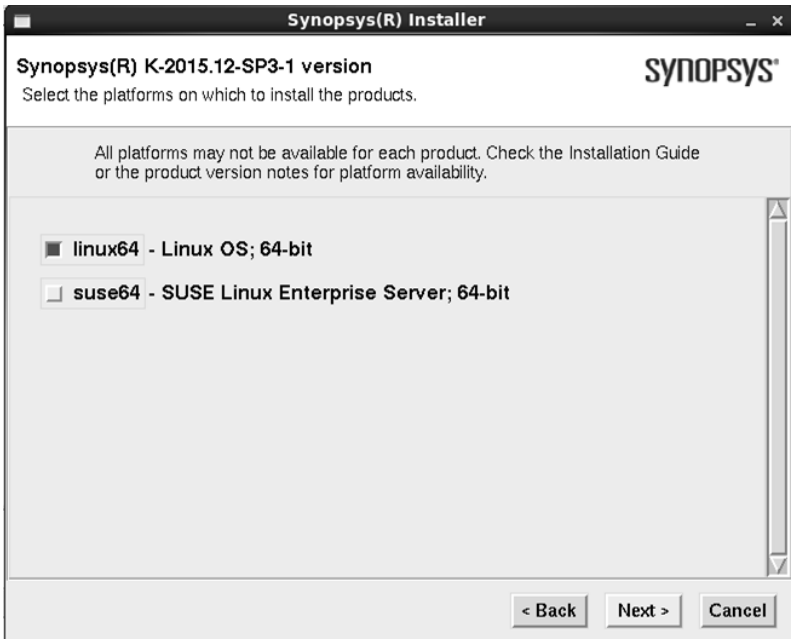
h. Deseleccionar los productos que no se desean instalar y son independientes. En este caso se deja seleccionado el producto de PrimeTime.

Figura 26. Selección de los componentes a instalar.



i. Seleccionar la plataforma de Linux64 para que sea compatible con el sistema operativo de CentOS.

Figura 27. Plataforma en la que se instalará PrimeTime (Linux64).

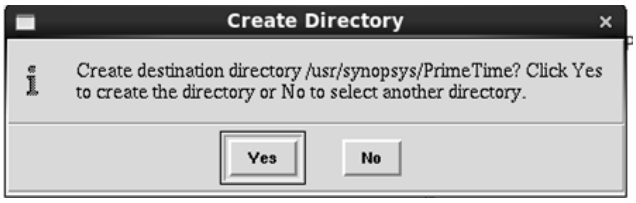


j. Seleccionar la dirección en la que se desea instalar PrimeTime.

Figura 28. Dirección destino para instalar PrimeTime.

Si la carpeta en la que se quiere instalar PrimeTime aún no existe, el instalador la crea automáticamente y muestra el siguiente mensaje.

Figura 29. Mensaje para confirmar la creación de la carpeta destino de instalación.



Al dar clic en "*Yes*" la carpeta es creada y al darle clic en "*No*" se debe elegir un directorio existente.

k. Verificar el producto elegido para instalar.

Figura 30. Ventana de verificación del producto que se va a instalar.

l. Seleccionar el botón *Accept, Install* para empezar la instalación.

Figura 31. Inicio de la instalación de PrimeTime.

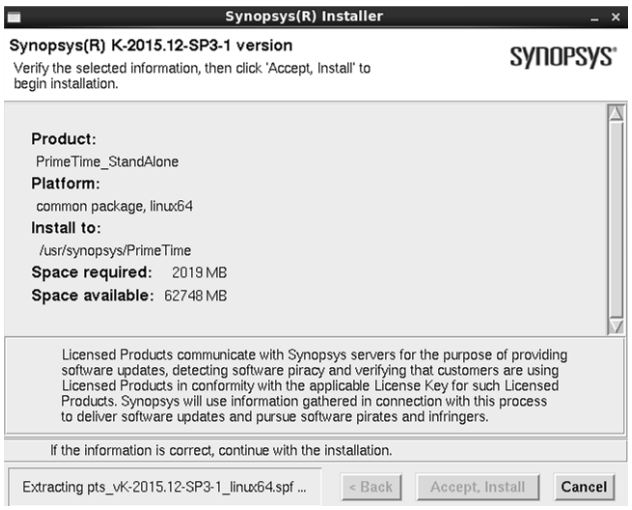
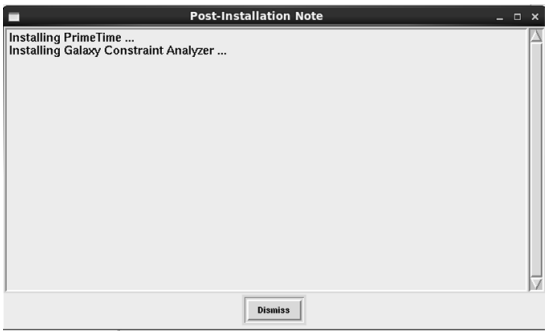


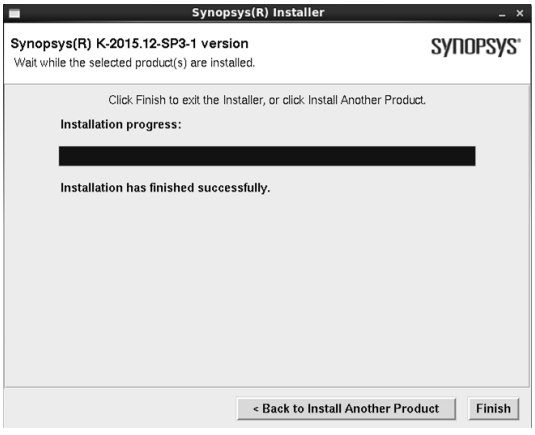
Figura 32. Ventana que muestra el avance de la instalación.

Figura 33. Mensajes de post instalación.



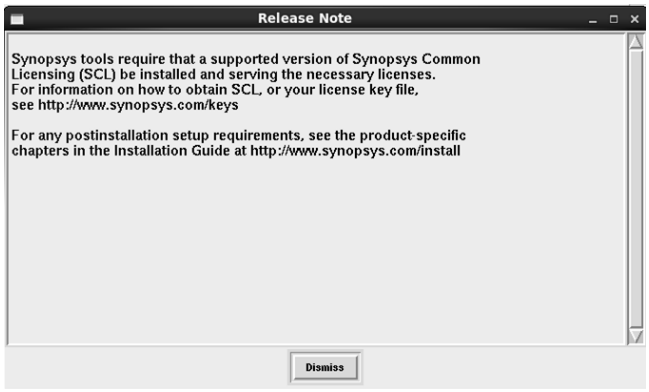
Se debe dar clic al botón *Dismiss* cuando aparecen las notas de post-instalación. Cuando la instalación termina, se muestra la siguiente imagen y hay que dar clic en el botón *Finish.*

Figura 34. Ventana que muestra la finalización de la instalación.



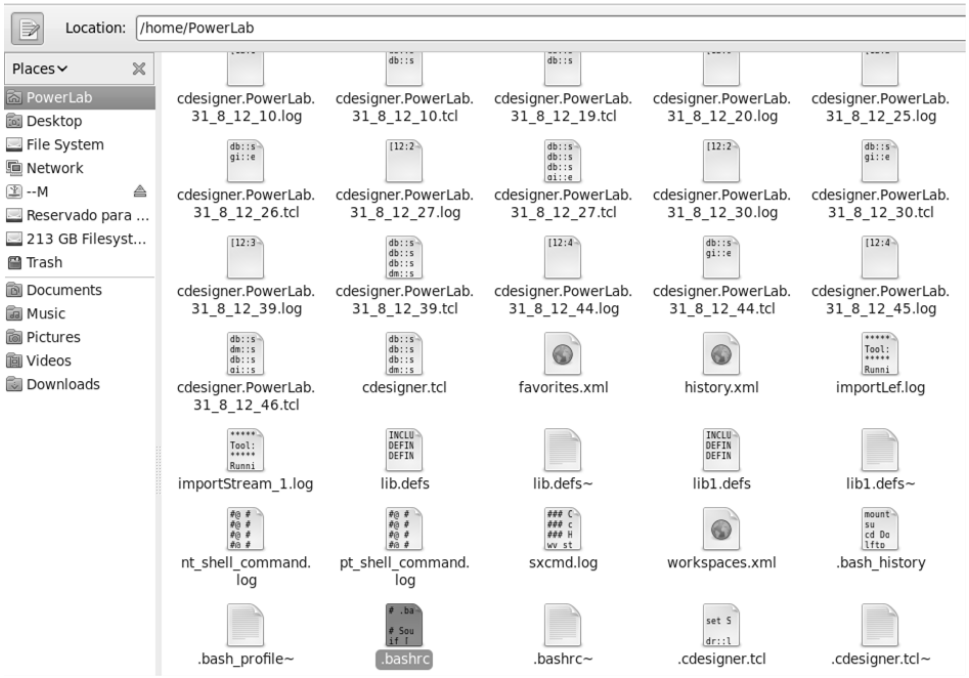
Se muestran las notas de publicación después de la instalación. Dar clic en el botón *Dismiss* para terminar con la instalación.

Figura 35. Notas de publicación.



m. Es necesario agregar las variables de entorno como explica el manual de instalación y también de los Santos (2014:35), en el archivo ".bashrc" que se encuentra oculto en la carpeta de PowerLab.

Figura 36. Directorio donde se encuentra el archivo ".bashrc".



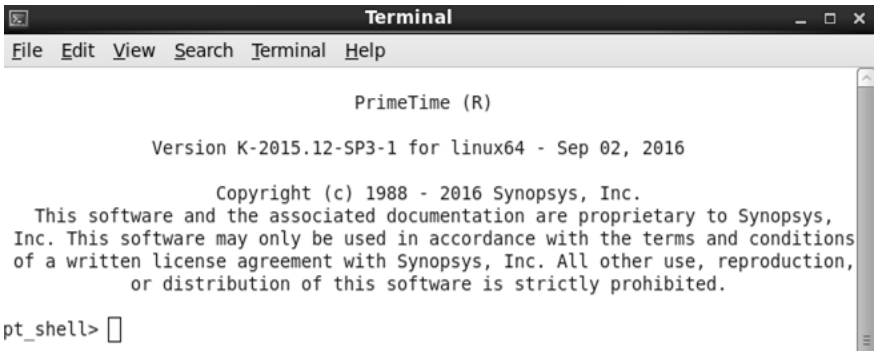
También se debe agregar las siguientes líneas de variables de entorno en el archivo ".bashrc":

```
PATH=/usr/synopsys/PrimeTime/bin/:$PATH
export PATH
```

Esto permite ejecutar PrimeTime desde cualquier carpeta de trabajo con la línea de comandos. Para verificar si PrimeTime está correctamente instalado, se debe escribir en la línea de comandos:

```
pt_shell
```

Figura 37. Comprobación de la versión y correcta instalación de PrimeTime.



Si al ejecutar el comando aparece la versión instalada de PrimeTime y el texto de la última figura, la instalación fue exitosa y PrimeTime está listo para usarse.

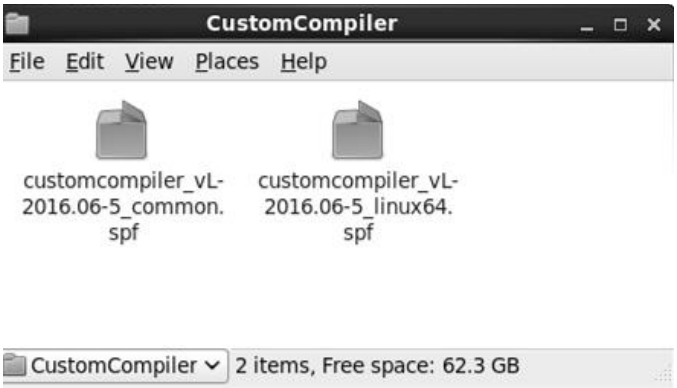
3. Instalación de Galaxy Custom Compiler. Está herramienta se instaló porque al leer los manuales de NanoTime se mencionaba que se puede instalar una interfaz para usar NanoTime con Custom Compiler. Custom Compiler es similar a Custom Designer (herramienta previamente instalada por en las computadoras del laboratorio de Ingeniería Electrónica de la Universidad del Valle de Guatemala).

Custom Compiler también posee una interfaz más amigable al usuario que Custom Designer. Además, resuelve algunos *bugs* de Custom Designer como lo es la extracción de parásitos en formato Open Access (extracción física). Para la instalación se puede seguir el manual de instalación de Custom Compiler o seguir los pasos descritos a continuación.

a. Ingresar con el usuario y contraseña al sitio de solvnet.synopsys.com

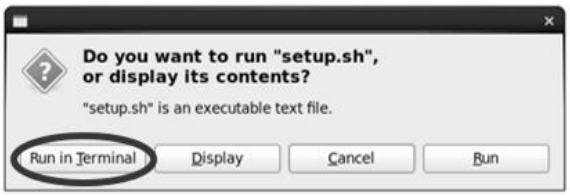
b. Descargar los instaladores de Galaxy Custom Compiler del sitio de Synopsys: agrupar todos los archivos en una sola carpeta y recordar el directorio.

Figura 38. Archivos necesarios para la instalación de Custom Compiler.



c. Abrir el instalador de aplicaciones Synopsys corriendo en la terminal el archivo “setup.sh” que se encuentra en la dirección: /usr/synopsys/installer

Figura 39. Ejecución del instalador de aplicaciones de Synopsys.

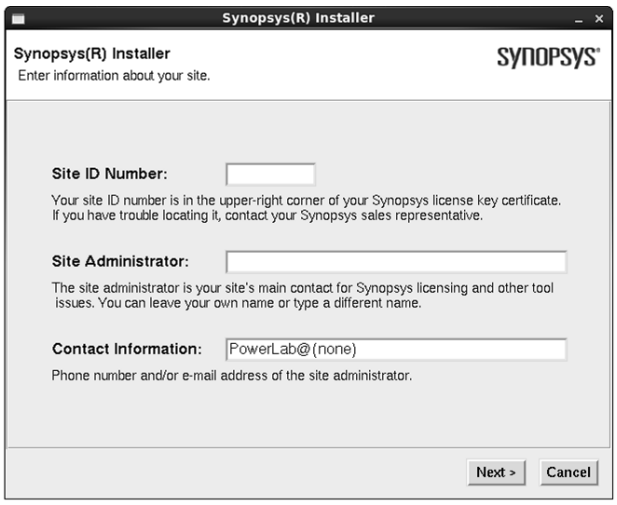


Después de ejecutar el archivo “setup.sh” aparece la interfaz gráfica del instalador de Synopys.

Figura 40. Interfaz gráfica del instalador de aplicaciones de Synopsys.

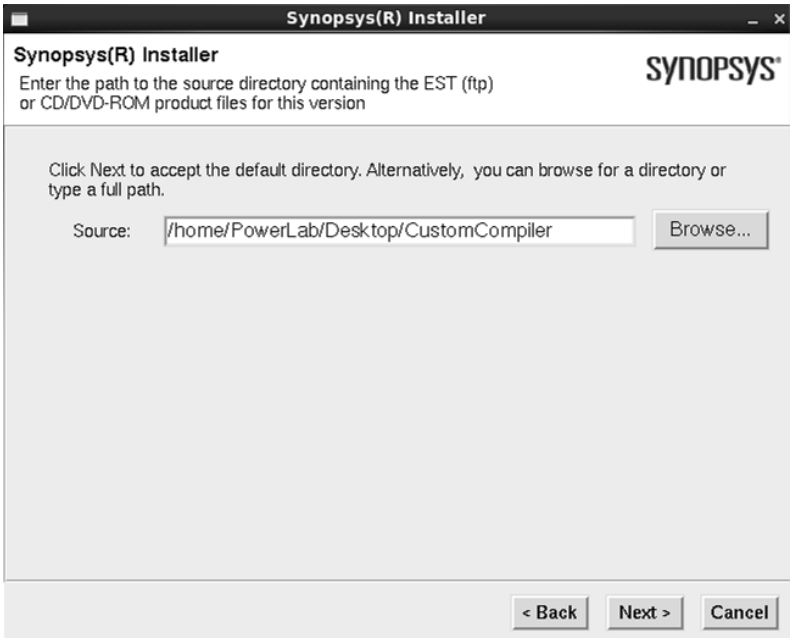
d. Al dar clic en el botón *Start* aparece otra pantalla en la que es obligatorio llenar todos los campos requeridos para iniciar el proceso de instalación.

Figura 41. Información obligatoria para empezar la instalación



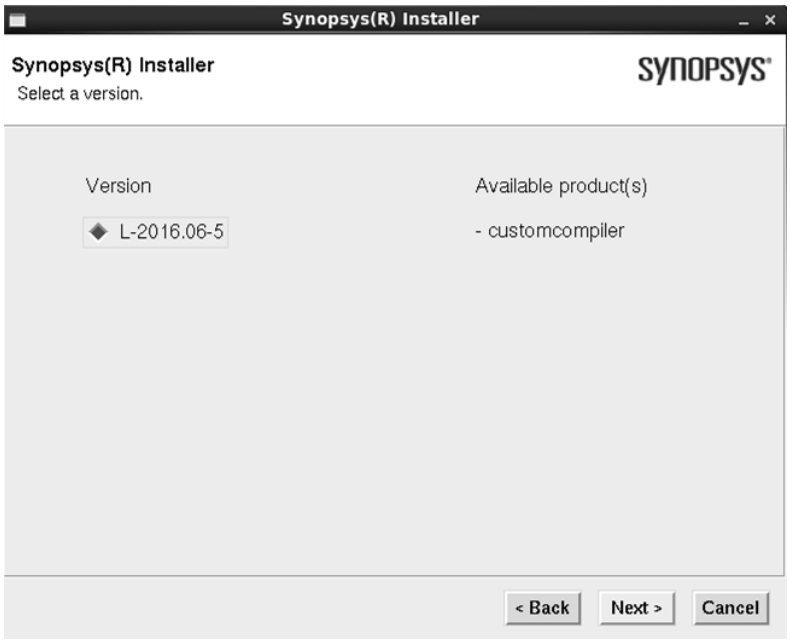
Luego en el botón de *Browse* buscar la dirección en donde se encuentran los archivos descargados, en este caso están en la carpeta indicada en el paso 2.

Figura 42. Directorio en donde se encuentran los instaladores de Custom Compiler



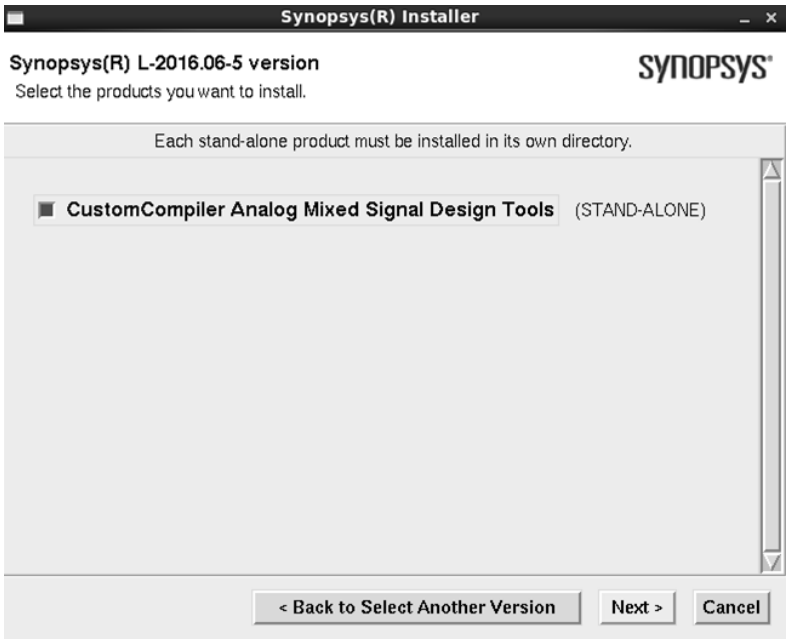
e. Seleccionar la versión de Custom Compiler a instalar.

Figura 43. Versión que se instalará de Custom Compiler.



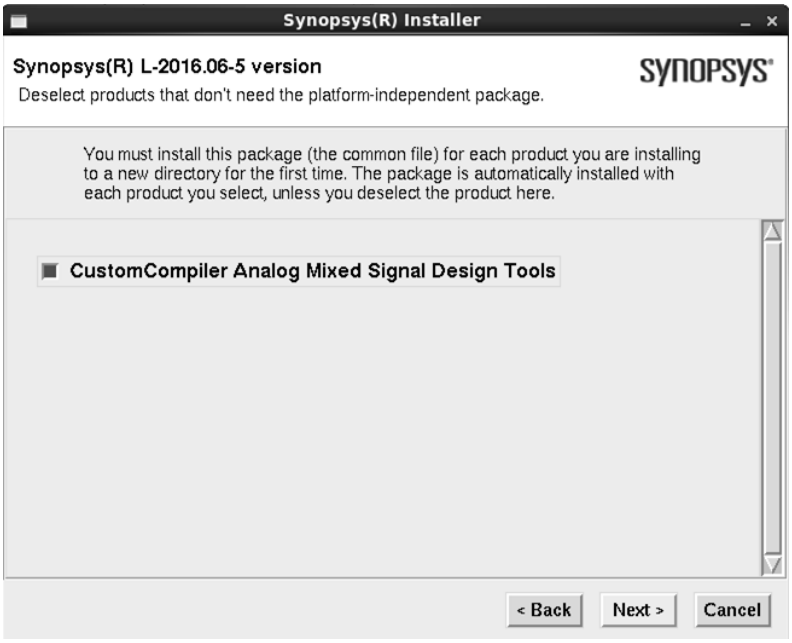
f. Seleccionar el producto que se desea instalar.

Figura 44. Selección del producto de Custom Compiler a instalar.



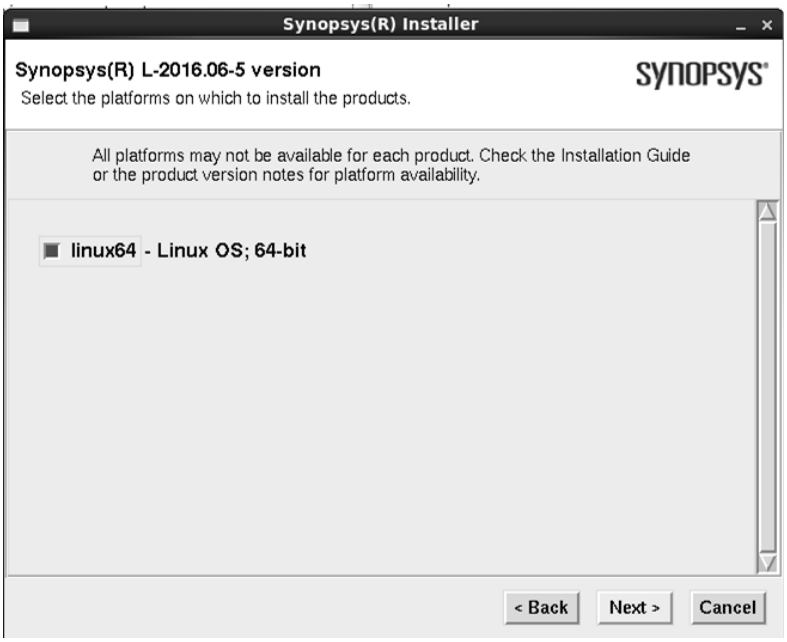
g. Seleccionar sólo los componentes que se desea instalar. En este caso se deja seleccionado el producto de Custom Compiler.

Figura 45. Seleccionar el producto para la instalación (es obligatorio para este caso).



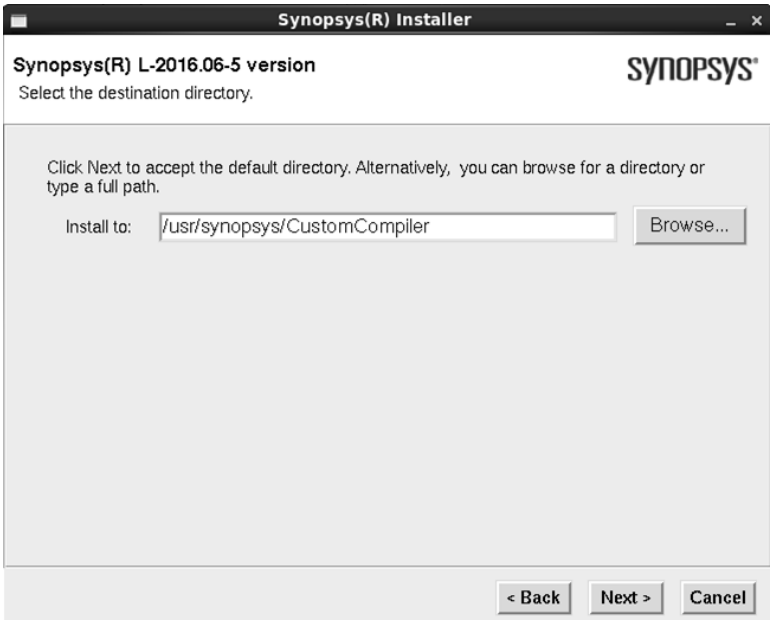
h. Seleccionar la plataforma de linux64 para que sea compatible con el sistema operativo de CentOS.

Figura 46. Plataforma en la que se instalará Custom Compiler (Linux64).



i. Seleccionar la dirección en la que se desea instalar Custom Compiler. En este caso se creó una carpeta llamada “CustomCompiler” en el directorio: /usr/synopsys/.

Figura 47. Dirección destino para instalar Custom Compiler.



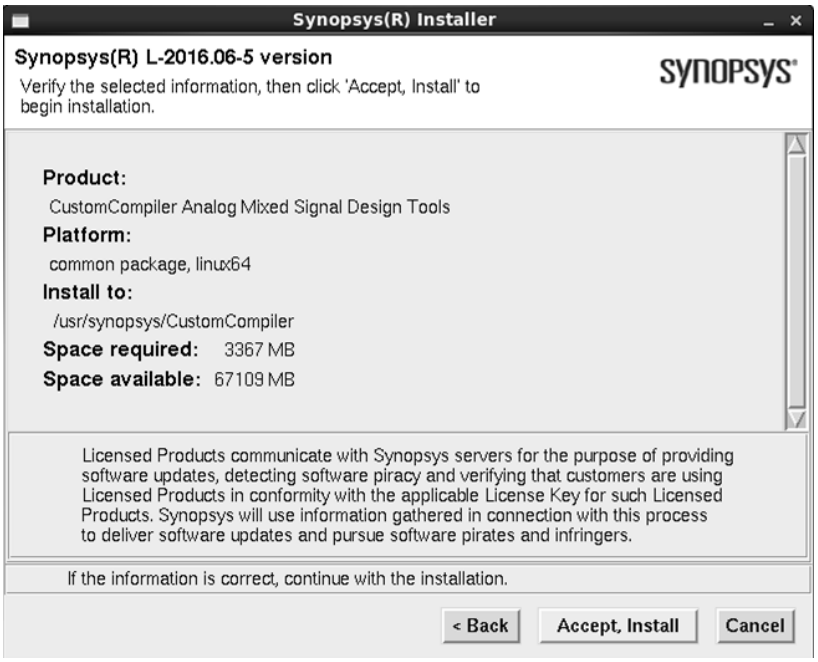
Si la carpeta en la que se quiere instalar Custom Compiler aún no existe, el instalador la crea automáticamente y muestra el siguiente mensaje.

Figura 48. Mensaje para confirmar la creación de la carpeta destino de instalación.

Al dar clic en "*Yes*" la carpeta es creada, y al hacer clic en "*No*" se debe elegir un directorio existente.

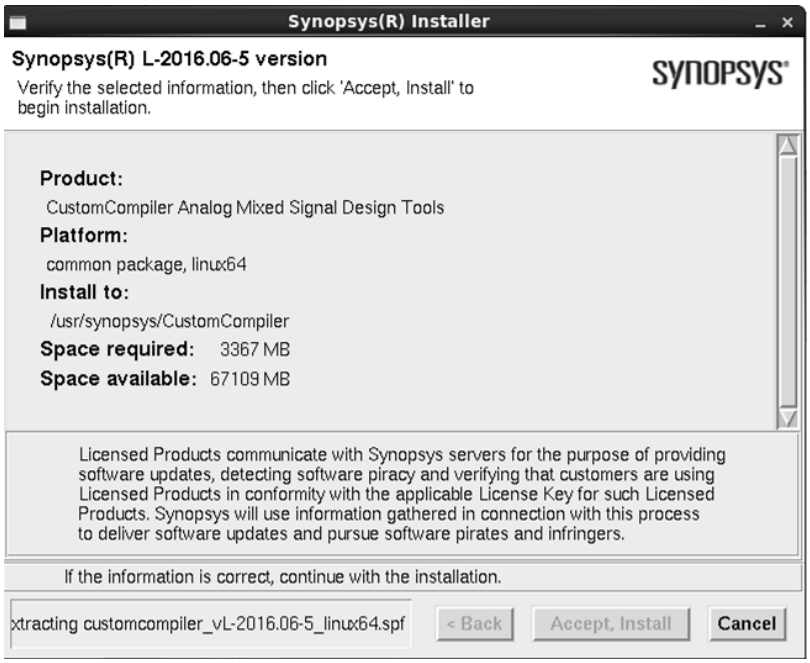
j. Verificar el producto elegido a instalar.

Figura 49. Ventana de verificación del producto que se va a instalar.



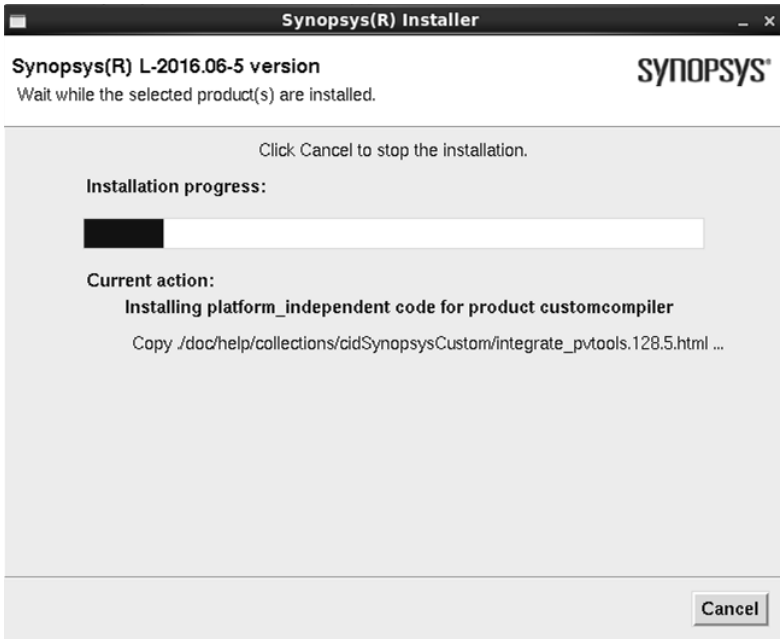
k. Seleccionar el botón *Accept, Install* y se empieza la extracción de los archivos para la instalación.

Figura 50. Inicio de la instalación de Custom Compiler.



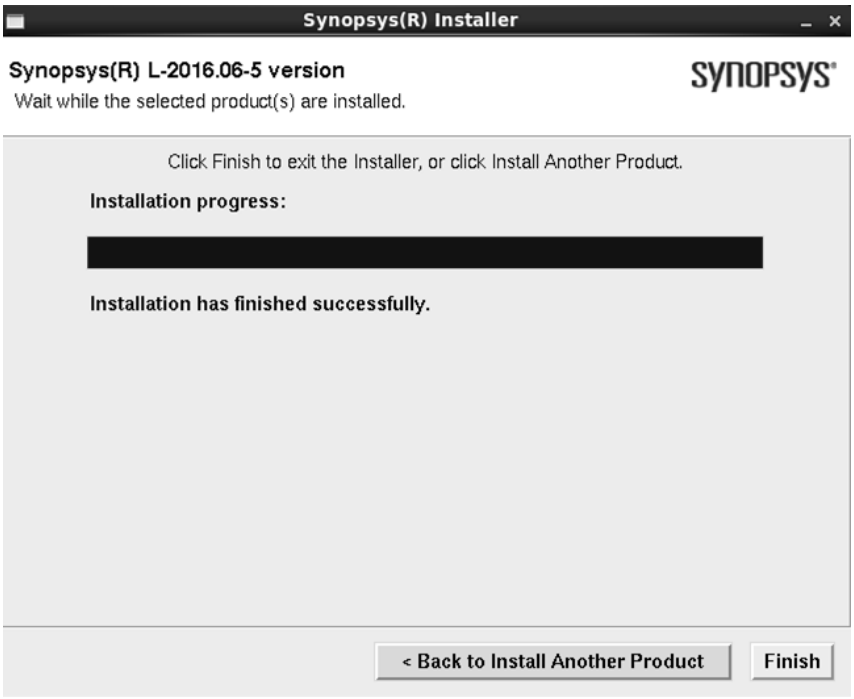
l. Esperar mientras la herramienta es instalada.

Figura 51. Ventana que muestra el avance de la instalación.



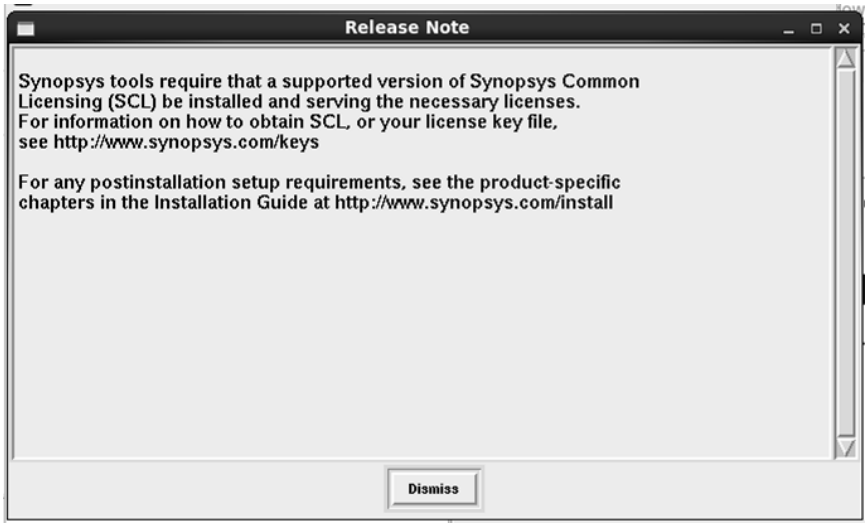
Cuando la instalación ha terminado, se debe dar clic en el botón *Finish*.

Figura 52. Ventana que muestra la finalización de la instalación.



Las notas de publicación se muestran después de la instalación. Dar clic en el botón *Dismiss* para terminar con la instalación.

Figura 53. Notas de publicación.



m. Es necesario agregar las variables de entorno de la misma forma en que se agregaron para la instalación de PrimeTime explicada anteriormente. En este caso se deben agregar las siguientes líneas de variables de entorno en el archivo ".bashrc":

```
PATH=/usr/synopsys/CustomCompiler/bin/:$PATH
export PATH
```

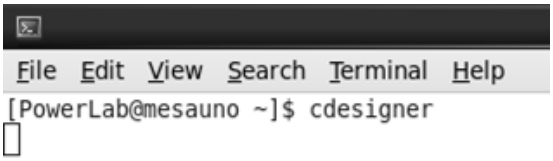
Esto permitirá ejecutar Custom Compiler con la línea de comandos desde carpeta.

n. Para verificar que Custom Compiler este correctamente instalado, escribir en la línea de comandos:

```
cdesigner o ccompiler
```

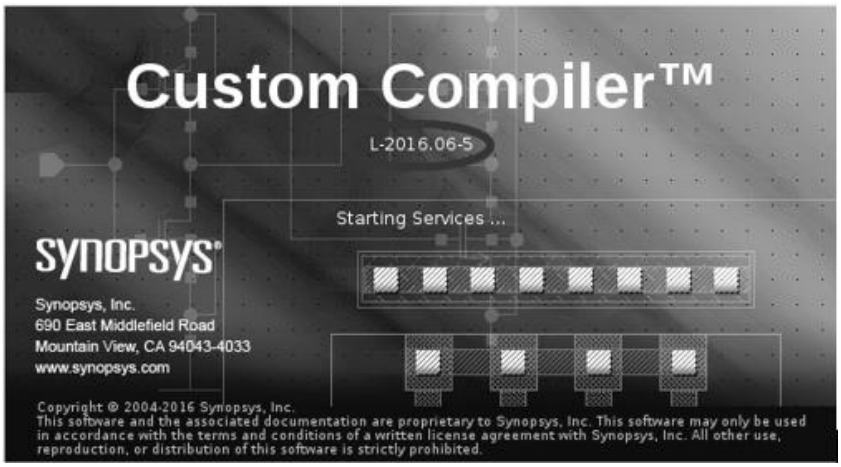
Custom Compiler se puede ejecutar con cualquiera de los dos comandos anteriores.

Figura 54. Comando para abrir Custom Compiler desde la línea de comandos.



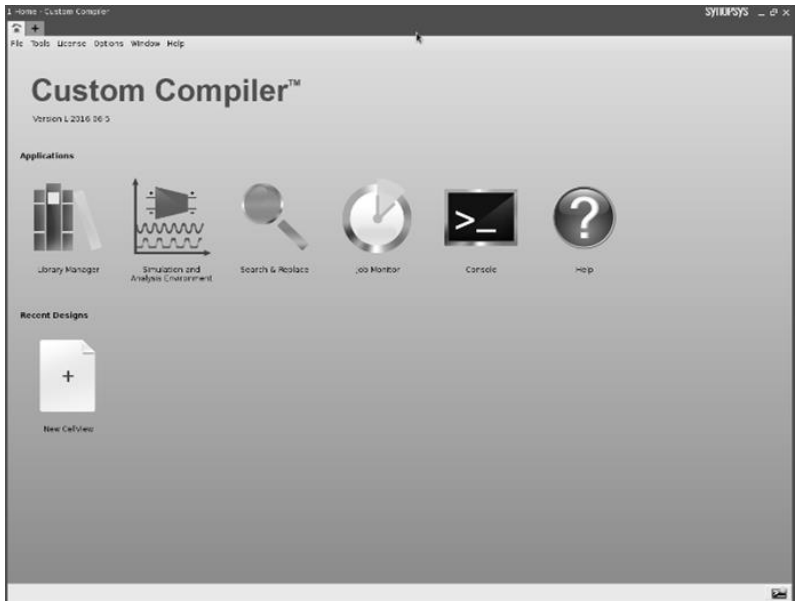
Si la instalación fue exitosa y se agregaron correctamente las variables de entorno, Custom Compiler se ejecuta y en pantalla se mostrará la ventana de la Figura 55.

Figura 55. Ventana de bienvenida de Custom Compiler.



A diferencia de Custom Designer (de los Santos, 2014), Custom Compiler muestra una interfaz gráfica amigable con el usuario.

Figura 56. Interfaz gráfica para empezar a usar Custom Compiler.

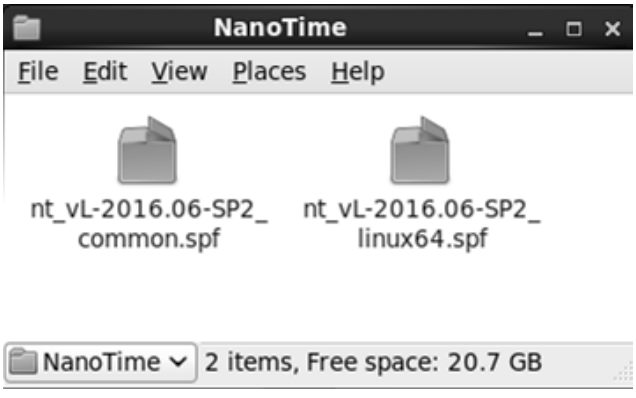


4. Instalación de NanoTime. La herramienta de NanoTime se instaló en las computadoras de la Universidad del Valle de Guatemala para poder hacer análisis a nivel transistor. Se puede seguir el manual de instalación de NanoTime que se genera en el sitio de Synopsys (Synopsys, 2016) o los pasos de instalación de alguna herramienta descritos en la tesis de Jonathan de los Santos (2014:34). A continuación se listaron los pasos que se siguieron para instalar NanoTime.

a. Ingresar con el usuario y contraseña al sitio de solvnet.synopsys.com

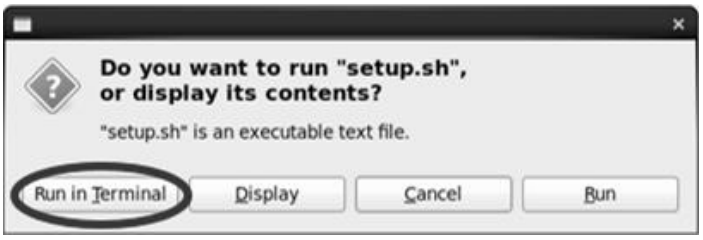
b. Descargar los instaladores de NanoTime del sitio de Synopsys. Agrupar todos los archivos en una carpeta.

Figura 57. Archivos necesarios para la instalación de NanoTime.



c. Abrir el instalador de aplicaciones Synopsys corriendo en la terminal el archivo “setup.sh” que se encuentra en la dirección: /usr/synopsys/installer.

Figura 58. Ejecución del instalador de aplicaciones de Synopsys.

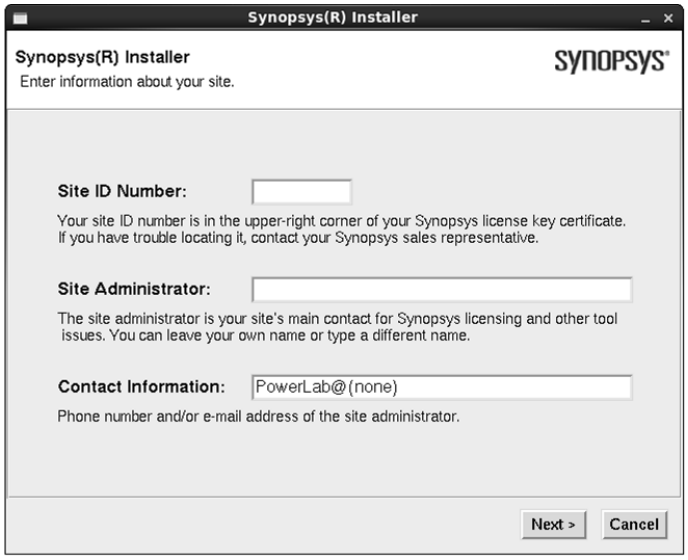


Después de correr el archivo “setup.sh” aparece la interfaz gráfica del instalador de Synopys.

Figura 59. Interfaz gráfica del instalador de aplicaciones de Synopsys.

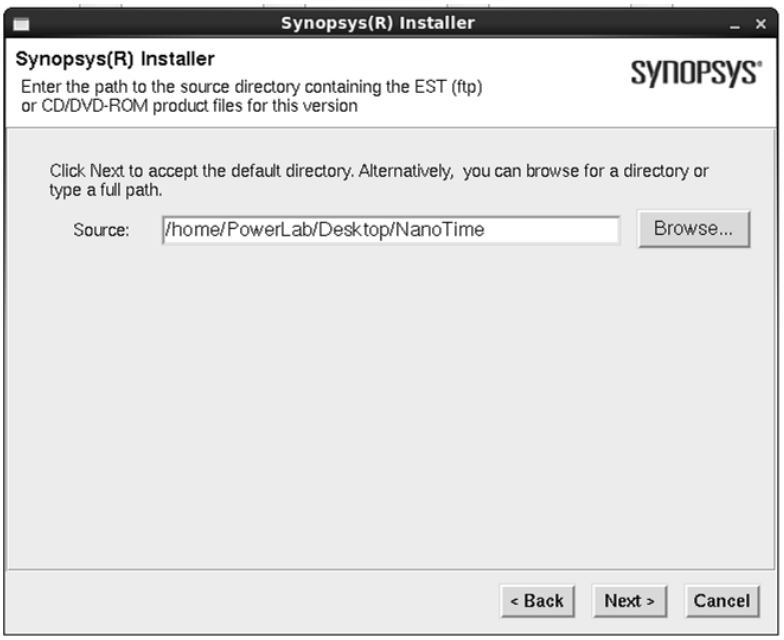
d. Al dar clic en el botón *Start* aparece otra pantalla en la que es obligatorio llenar todos los campos requeridos para iniciar el proceso de instalación.

Figura 60. Información obligatoria para empezar la instalación.



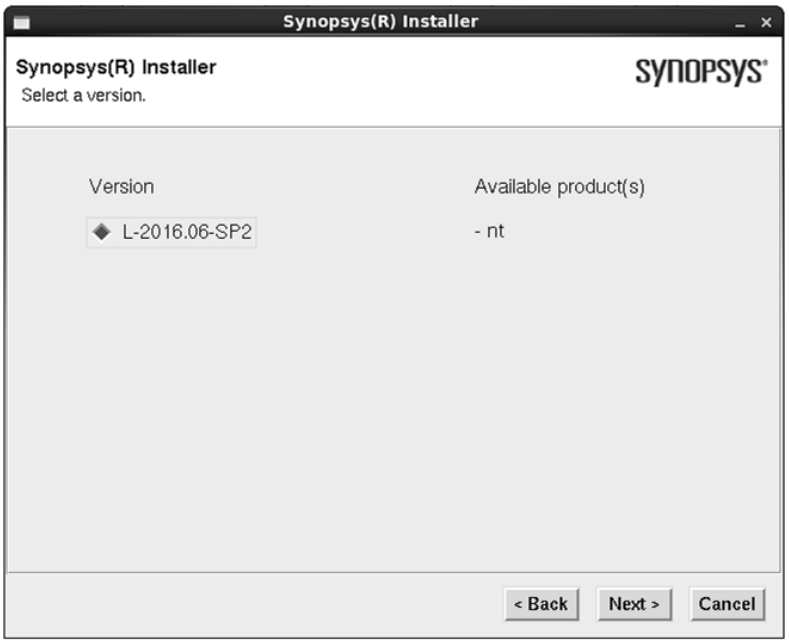
Luego en el botón de *Browse* buscar la dirección en donde se encuentran los archivos descargados, en este caso están en una carpeta que se llama NanoTime ubicada en el escritorio (como se indica en el paso 2).

Figura 61. Directorio donde se encuentran los instaladores de NanoTime.



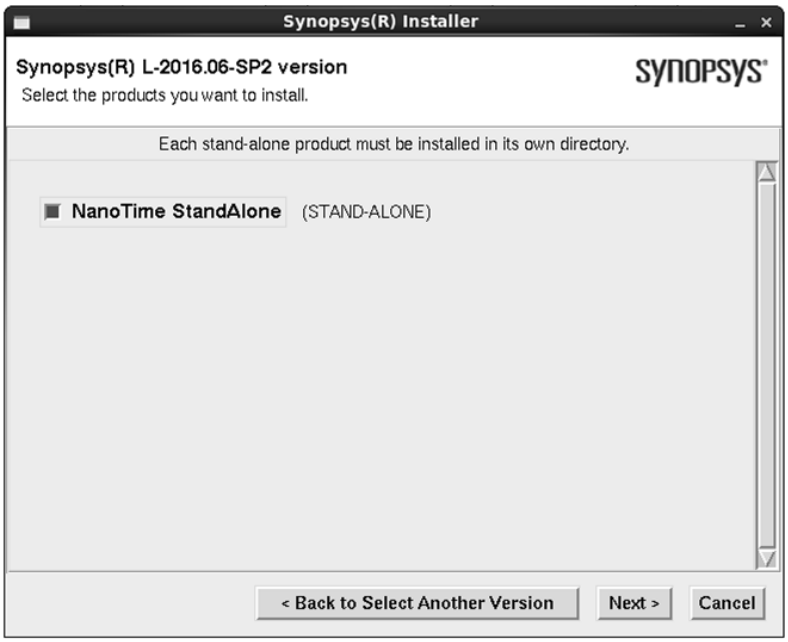
e. Seleccionar la versión de NanoTime que se está instalando.

Figura 62. Versión de NanoTime que se instalará.



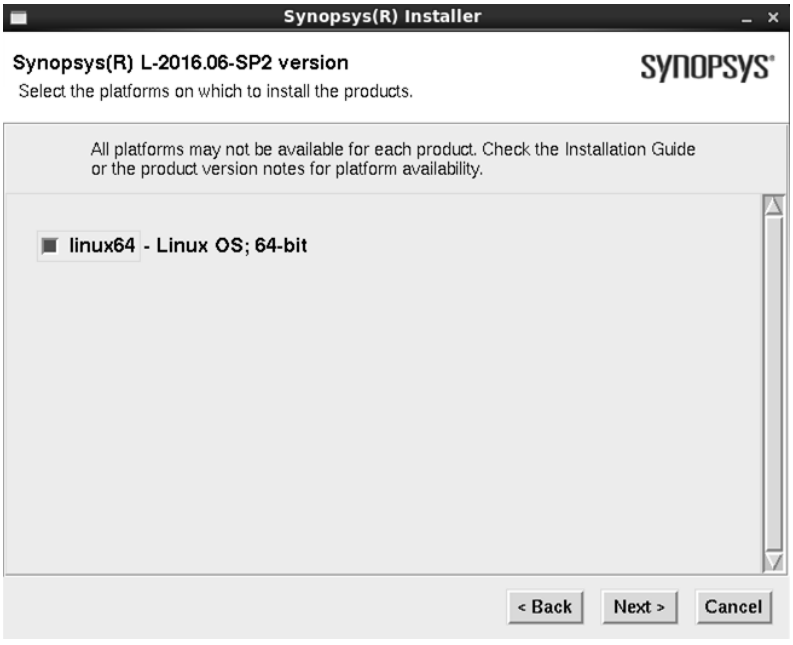
f. Seleccionar la herramienta que se quiere instalar.

Figura 63. Herramienta a instalar (NanoTime StandAlone).



g. Seleccionar la plataforma de linux64 para que sea compatible con el sistema operativo de CentOS.

Figura 64. Plataforma en la que se instalará NanoTime (Linux64)



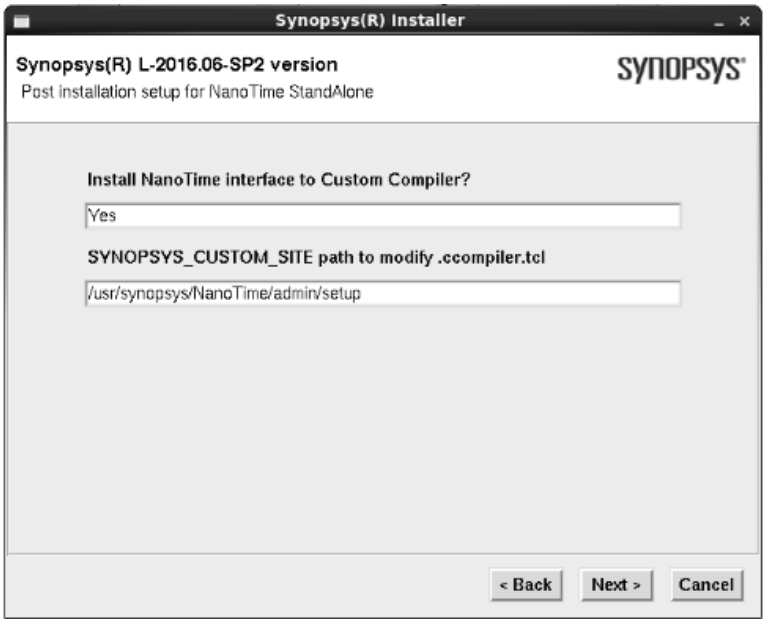
h. Elegir la dirección: /usr/synopsys/ y agregar el nombre de la carpeta en la que se desea instalar la herramienta. En este caso se nombró “NanoTime”. No hace falta crear la carpeta antes de

llegar a este paso. Si no existe la carpeta, el instalador la crea y si ya existe, el instalador sobrescribe todo lo que esté actualmente en esa carpeta.

Figura 65. Directorio en el que se desea instalar NanoTime

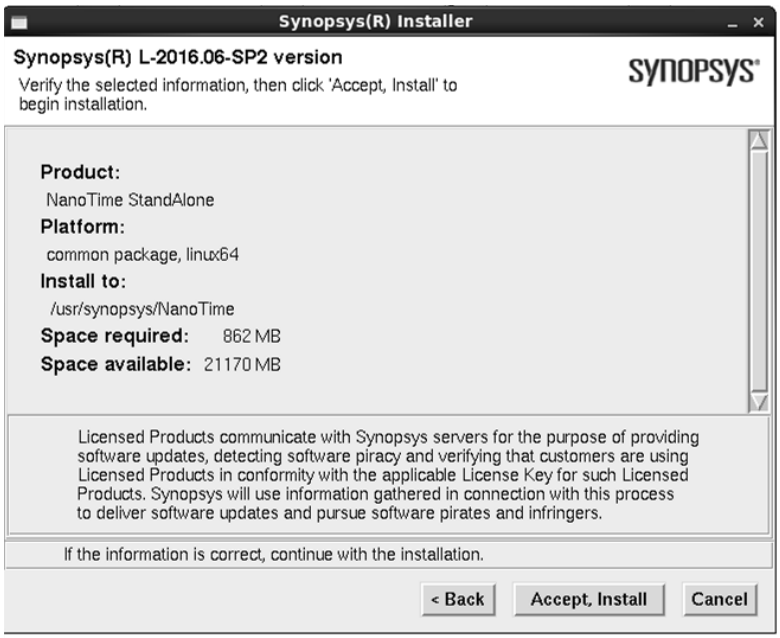
i. NanoTime se puede usar con la interfaz de Custom Compiler, que se instaló previo a instalar NanoTime. En la siguiente opción si se desea usar NanoTime junto con Custom Compiler escribir "*Yes*" en el primer cuadro de texto y en el segundo escribir la siguiente dirección: /usr/synopsys/NanoTime/admin/setup.

Figura 66. Instalar la interfaz de NanoTime para usarla con Galaxy Custom Compiler



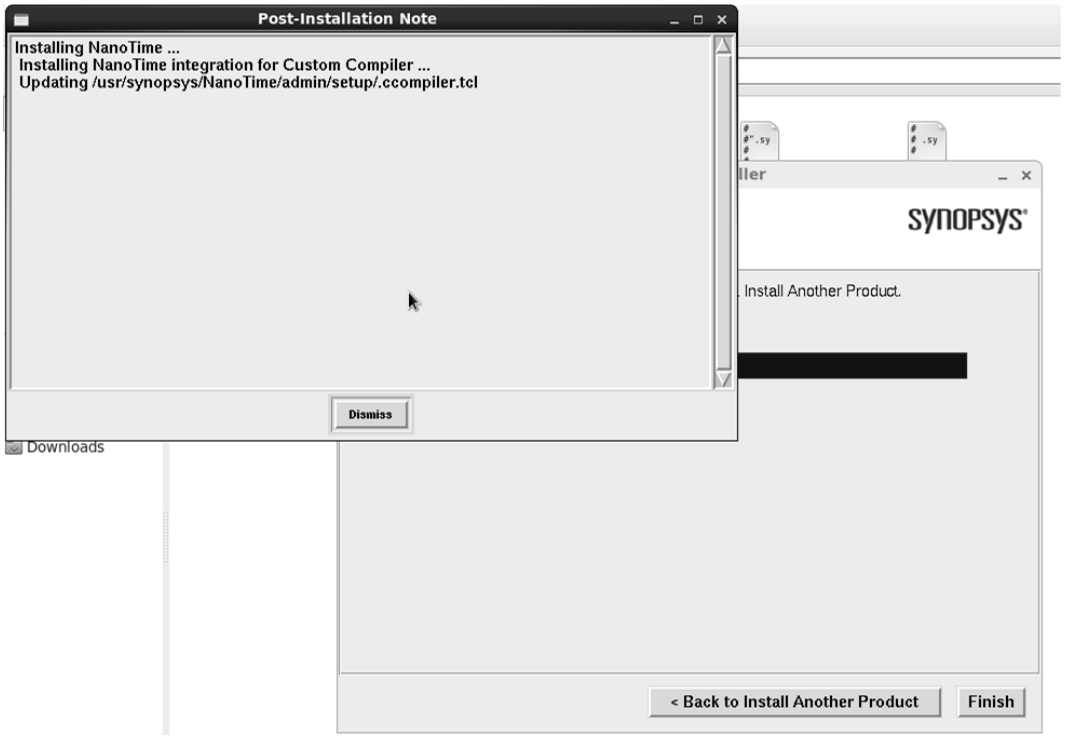
j. Hacer clic en Accept, Install.

Figura 67. Ventana de verificación de la configuración para empezar la instalación.



k. Por último, dar clic en *Dismiss* y luego en *Finish* y si no se muestra ningún error, la instalación fue exitosa.

Figura 68. Ventana que indica que la instalación ha finalizado.



l. Es necesario agregar las variables de entorno como se explica en el manual de instalación y en la tesis de Jonathan de los Santos (2014:35) en el archivo ".bashrc" que se encuentra oculto en la carpeta de PowerLab. Esto se hace de la misma manera que se explicó para las instalaciones de PrimeTime y Custom Compiler. Se deben agregar las siguientes líneas de variables de entorno en el archivo ".bashrc":

```
PATH = /usr/synopsys/NanoTime/bin/:$PATH
export PATH
export SYNOPSYS_CUSTOM_SITE = /usr/synopsys/NanoTime/admin/setup
```

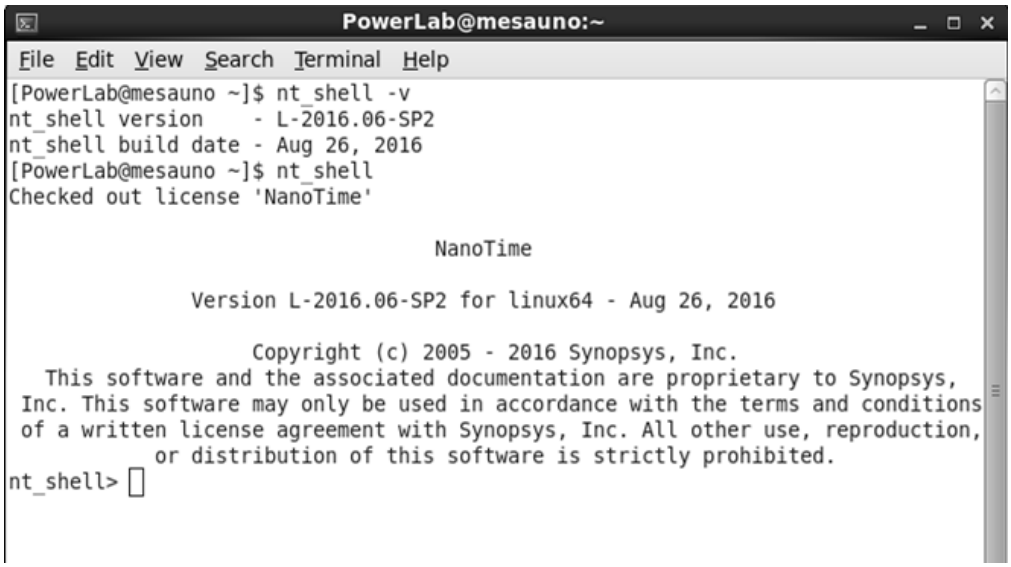
Esto permite ejecutar NanoTime desde la línea de comandos en cualquier carpeta de trabajo, y también desde Custom Compiler. Para verificar que NanoTime este correctamente instalado, escribir en la línea de comandos:

```
nt_shell –v
```

Figura 69. Versión que se instaló de NanoTime.


```
nt_shell
```

Figura 70. Comprobación de la correcta instalación de NanoTime.



Si al ejecutar los dos comandos aparece la versión instalada de NanoTime y el texto de la figura anterior, la instalación fue exitosa y NanoTime está listo para usarse.

5. Instalación de la librería de 90nm. Una de las finalidades de este trabajo es analizar el impacto del ruido por acople capacitivo cuando se encuentra en operación subumbral y comparar los resultados entre diferentes tecnologías de transistores como 32nm, 45nm y 90nm. Se necesita tener instaladas las tecnologías en la computadora y aquí se muestran los pasos para la librería de 90nm.

a. Ingresar con el usuario y contraseña al sitio de solvnet.synopsys.com

b. Entrar al siguiente enlace del sitio de Synopsys: https://www.synopsys.com/cgi-bin/protected/university/gl90nm/reg1.cgi.

c. Contestar una breve encuesta para empezar la descarga de la librería de 90nm.

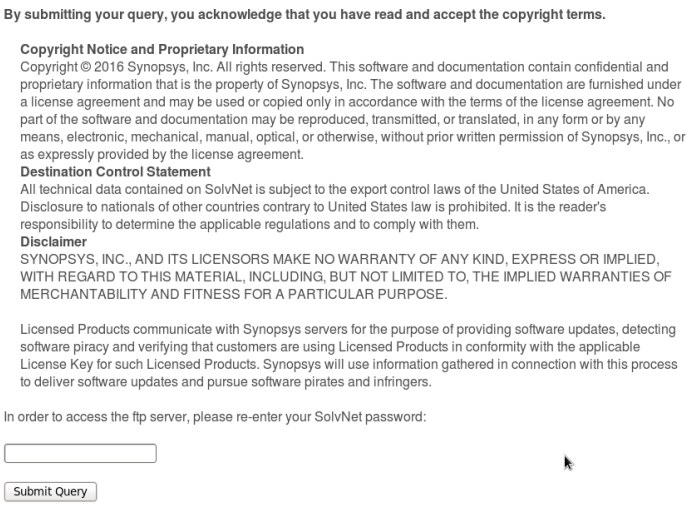
Figura 71.Encuesta para descargar la librería de 90nm.

La encuesta se contestó como se muestra en la figura anterior por el tipo de proyecto realizado. Las respuestas pueden variar si la librería se utilizará para otros fines o con otros circuitos.

Figura 72. Términos y condiciones que se deben leer y aceptar para poder usar la librería de 90nm.

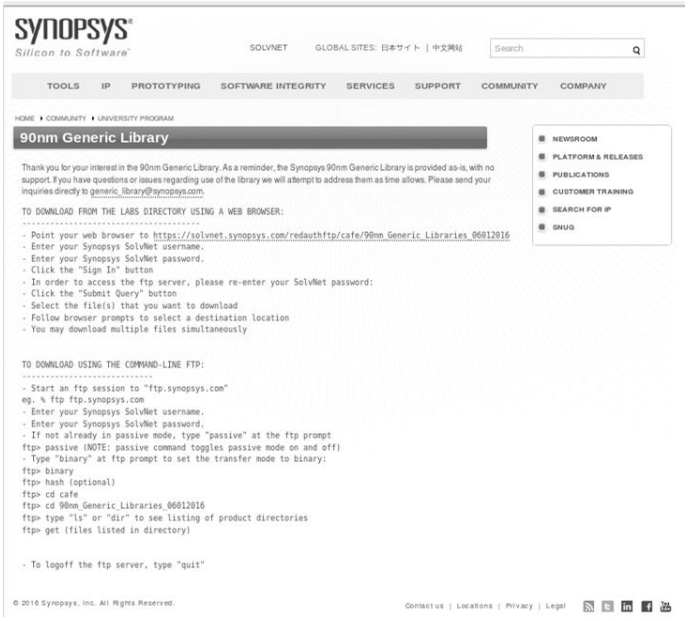
d. Por seguridad, el sitio de Solvnet pide volver a ingresar la contraseña.

Figura 73. Ventana de seguridad de Solvnet para re ingresar la contraseña.



e. Elegir la forma en que se desea descargar la librería de 90nm.

Figura 74. Instrucciones para la descarga de la librería de 90nm.



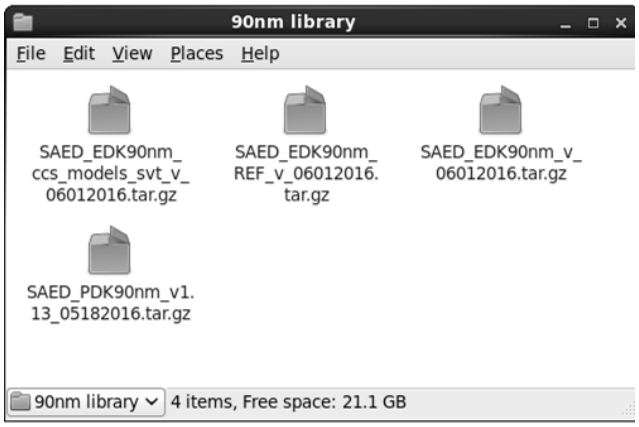
f. Agrupar los archivos descargados en una carpeta en el escritorio llamada “90nm_library”.

g. Entrar al siguiente enlace del sitio de Synopsys para descargar el iPDK de la librería de 90nm: https://www.synopsys.com/cgi-bin/protected/university/ipdk90nm/reg1.cgi.

h. Seguir los pasos del 3 al 5 nuevamente.

i. Guardar el archivo PDK en la carpeta creada en el paso 6.

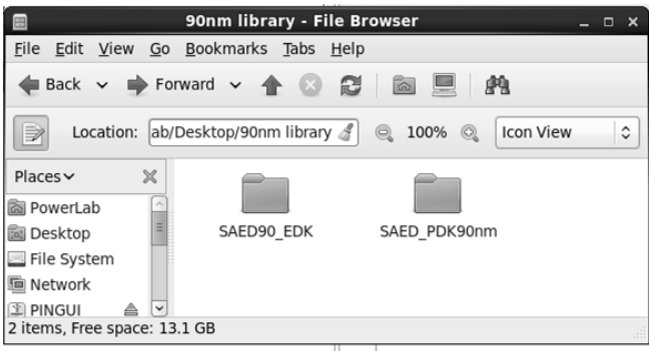
Figura 75. Archivos descargados para la instalación de la librería de 90nm.



j. Descomprimir todos los archivos y agruparlos en una sola carpeta a excepción del PDK. Este se debe descomprimir en una carpeta separada.

Figura 76. Extracción de archivos descargados para la instalación de la librería de 90nm.

Figura 77. Archivos necesarios extraídos para la instalación de la librería de 90nm.



k. Mover la carpeta que contiene el iPDK "SAD_PDK90nm" al directorio /usr/synopsys/iPDK.

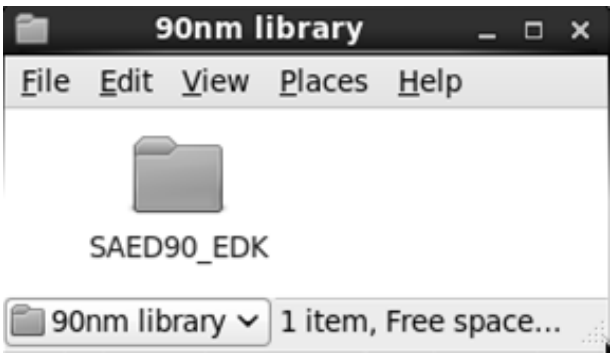
Figura 78. Directorio para mover el PDK extraído de la librería de 90nm.

l. Crear las variables de entorno. Para esto se debe abrir el archivo ".baschrc" que se encuentra oculto en la carpeta de PowerLab y agregar el siguiente comando:

```
export SAED90_PDK=/usr/synopsys/iPDK/SAED_PDK90nm/
```

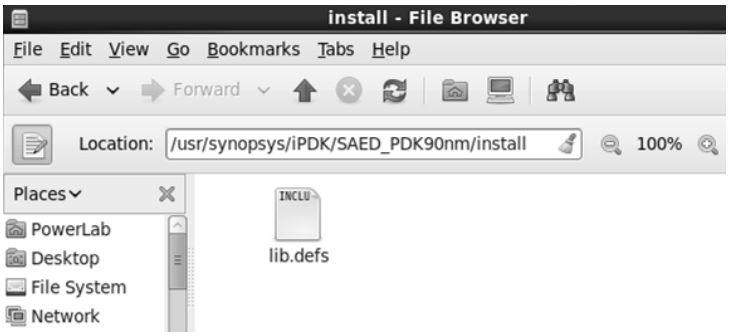
La carpeta "SAED90_PDK" debe existir. Esta carpeta es en donde se encuentran extraídos los archivos que se descargaron para la instalación de la librería de 90nm (no es el archivo PDK). Luego de mover la carpeta del PDK al directorio correcto, la carpeta de "90nm_library" se ve como la figura siguiente. Notar que la carpeta descomprimida se llama "SAED90_PDK".

Figura 79. Carpeta en donde se encuentran los archivos finales de la librería de 90nm.



m. Mover el archivo "lib.defs" del directorio: /usr/synopsys/iPDK/SAED_PDK90nm/install a la carpeta de PowerLab.

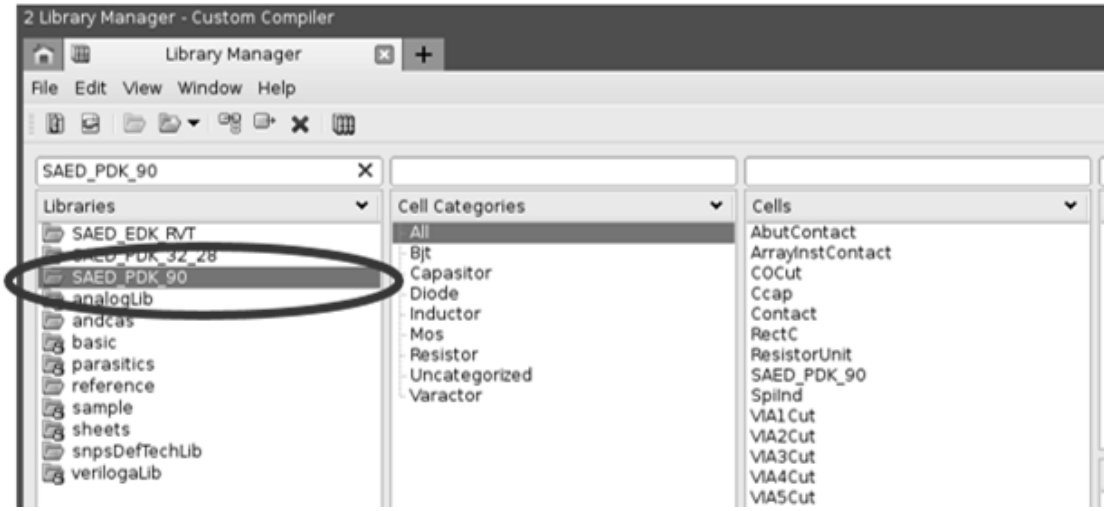
Figura 80. Directorio del archivo “lib.defs”.



Si ya existe una librería instalada, ya debe existir el archivo “lib.defs” en la carpeta PowerLab, entonces sólo se debe copiar el contenido del archivo y pegarlo al archivo “lib.defs” existente en PowerLab.

n. La librería ya se encuentra instalada y al abrir Custom Compiler se puede verificar al encontrar la carpeta de los archivos de ejemplo y celdas estándar de la librería de 90nm.

Figura 81. Instalación de la librería de 90nm finalizada.



B. Fases de diseño

1. Realización del *layout* del circuito. Para diseñar un *layout* se deben seguir las reglas de diseño (Harris & Weste, 2011:113) desde el inicio para asegurar que el diseño se pueda fabricar. El circuito podría funcionar en simulaciones si no se cumplen las reglas, pero no se podría garantizar que se pueda fabricar. Para fines de este estudio es importante cumplir con las reglas de diseño para que los resultados de las simulaciones sean lo más cercano a la realidad.

Se hicieron varios *layout* siguiendo los pasos descritos en el manual de usuario de Custom Compiler (Synopsys, Custom Compiler Help, 2016). Los *layout* fueron realizados a partir de *standard cells* que la librería de la tecnología incluye cuando esta es instalada. Se usaron *standard cells* para no perder mucho tiempo en hacer una compuerta básica desde cero que cumpliera con todas las especificaciones de diseño. Así se aseguró que las compuertas cumplieran con las reglas de diseño. De esta manera se ahorró tiempo y se invirtió en realizar interconexiones bastantes largas para hacer las pruebas y notar el efecto de las capacitancias de acople.

Los *layout* diseñados para pruebas fueron: una cadena de ocho compuertas inversoras, un comparador de identidad de dos bits, y un D Flip-Flop de flanco positivo. Se recomienda agregar un *label* a cada interconexión con el nombre de su nodo respectivo. Las *nets* deben tener el mismo nombre que en el circuito esquemático (ver: Realización del circuito esquemático) para hacer las verificaciones de DRC y LVS de una manera más efectiva y generar los *netlist* con el mismo nombre de las *nets* del circuito esquemático.

2. Realización del circuito esquemático. Se debe tener un circuito esquemático por cada *layout* para hacer la verificación del LVS. Para hacer los circuitos esquemáticos se siguieron los pasos y recomendaciones que dio de los Santos (de los Santos, 2014). Por cada circuito se debe revisar el nombre de cada *net* del esquemático para regresar al *layout* del circuito y agregarles el mismo nombre a las interconexiones correspondientes. Se recomienda revisar cada interconexión porque aunque parezca que está conectada puede no estar haciendo contacto con lo que se desea conectar y al realizar las verificaciones del LVS, esto puede causar errores, tanto en el *layout* como en el esquemático. Se realizó un circuito esquemático por cada *layout* mencionado en la sección anterior (Realización del *layout*).

3. DRC y LVS. Una vez se tiene listo el *layout* y el circuito esquemático, se procede a realizar el análisis de DRC (*Design Rule Checking*) sobre el *layout*. El DRC se usa para revisar que el *layout* cumpla con todas las reglas de diseño. Esta verificación tiene que completarse exitosamente antes de proceder al análisis del LVS *(Layout Versus Schematic*). El LVS revisa que el *layout* coincida con el circuito esquemático y al terminar sin errores este análisis ya se puede proceder a la extracción de parásitos.

Estos análisis se hicieron para todos los circuitos prueba mencionados anteriormente y además con los circuitos sumador/restador de 32 *bits* y módulo generador de PWM para hacer las simulaciones respectivas con NanoTime.

4. LPE. El *Layout Parasitic Extraction* es la última verificación que se hace. Al hacer el LPE para este estudio se debe usar una configuración que no es igual a la descrita por de los Santos (de los Santos, 2014). El *runset* de StarRC que estaba disponible para usarse con la herramienta Custom Compiler se debió modificar debido a que las opciones predeterminadas de StarRC no son compatibles con NanoTime. Si se abre con NanoTime un archivo de parásitos generado con ese *runset*, NanoTime lanzará un error "PARA-040". Es

importante notar que el usuario puede escribir el comando "man" seguido de un comando de NanoTime o el código de un error, y en la consola aparecerá información útil. Es crucial asegurarse de que ciertos comandos estén incluidos en el *runset* de StarRC. Por defecto, StarRC nombra los puertos de los transistores MOSFET como DRN, GATE, SRC y BULK, que corresponden al *drain*, *gate*, *source* y *bulk*, por lo que NanoTime se debe configurar de esta misma forma. También se usaron las opciones "HIERARCHICAL_SEPARATOR: ." y "CASE_SENSITIVE: YES" para que NanoTime detectara la jerarquía del circuito correctamente. Además, NanoTime es sensible a mayúsculas, por lo que la extracción de parásitos debe respetar los nombres de todas las instancias (Synopsys, Setting Up the Back-Annotation Flow in NanoTime Using StarRC Parasitics Netlist, 2010). También se aseguró que las siguientes opciones estuvieran habilitadas:

- XREF: YES - esto asegura que los nombres de las instancias en el archivo generado por StarRC sean los nombres especificados en el esquemático.
- NETLIST_IDEAL_SPICE_FILE: ideal_spice.dpf - esto genera un archivo que incluye todos los dispositivos del circuito con sus parámetros físicos del *layout*, incluyendo ancho, largo y demás características.
- NETLIST_FORMAT: SPF - esto hace que StarRC escriba los parásitos en formato DSPF, a pesar de que el comando haga pensar que es un archivo SPF. Se podría generar un archivo SPEF pero al abrirlo con NanoTime da errores, entonces se recomienda usar los archivos DSPF.

Siempre que se modifiquen estas opciones también se recomienda volver a ejecutar el LVS del diseño. Adicionalmente, para hacer pruebas de integridad de las señales se debe asegurar que las capacitancias no se acoplen a tierra con la opción "COUPLE_TO_GROUND: NO". Debido a que todas las tecnologías debajo de 65nm están implementadas como subcircuitos en sus librerías, se debe agregar la opción "HN_NETLIST_SPICE_TYPE: ..." en el *runset* (Synopsys, Setting Up the Back-Annotation Flow in NanoTime Using StarRC Parasitics Netlist, 2010). En este caso los transistores que se usaron fueron los modelos n105 y p105 para el NMOS y el PMOS, respectivamente. Por lo tanto, se incluyó en el *runset*:

```
HN_NETLIST_SPICE_TYPE: n105 X
HN_NETLIST_SPICE_TYPE: p105 X
```

De esta forma se logró que StarRC automáticamente pusiera una "X" antes de las instancias de los transistores en el archivo generado. Debe notarse que si el usuario desea extraer los parásitos sólo de ciertos nodos, se puede usar la opción "NETS: ..." y listar los nombres de todos los nodos de interés. El usuario se puede referir a los nodos por su nombre del esquemático con el *runset* usado en esta investigación. Esto es complementado por la opción "NET_TYPE: SCHEMATIC/LAYOUT" para que la herramienta detecte los nombres de los nodos acorde a cómo se llaman en el *layout* o en el esquemático.

C. Pruebas con PrimeTime.

Se realizó el flujo de las fases de diseño con un circuito de prueba que consistía en un oscilador en anillo con tres compuertas inversoras. Se comenzaron las pruebas después de obtener el archivo de parásitos con la herramienta StarRC en formato SPEF. Debido a que la extracción se realizó a nivel de transistores, y PrimeTime requiere que los circuitos estén construidos a nivel de compuerta, se debió hacer un paso intermedio entre la generación de los parásitos y el análisis en PrimeTime. Como el circuito de prueba consistía en tres compuertas inversoras, se hizo un *script* de Python que recorriera el archivo de parásitos y lo modificara. Se supuso que si se hacía referencia a un *gate* de un transistor, se estaba haciendo referencia a la entrada de una compuerta inversora. Además, todos los *drains* o *sources* de los transistores que no estuvieran conectados a la fuente de alimentación se supusieron como la salida de una compuerta inversora. Debe recordarse que los transistores MOSFET son simétricos, por lo que el *drain* y el *source* son intercambiables. Los nombres se modificaron a "u#", donde "#" era el número de la compuerta. Este número se detectó usando la sección del archivo SPEF llamada "*NAME_MAP". El número de la compuerta correspondía al número de la instancia a la cual pertenecía. Con el circuito que se usó, la sección "*NAME_MAP" estaba compuesta por las siguientes líneas:

```
*NAME_MAP
*42 net5
*43 net9
*44 net17
*53 XI3|MM31
*54 XI3|MM32
*76 XI14|MM0
*77 XI14|MM1
*78 XI15|MM0
*79 XI15|MM1
*80 XI16|MM0
*81 XI16|MM1
```

por lo que las tres compuertas creadas eran "u14", "u15" y "u16". Adicionalmente, los transistores de la instancia XI3 correspondían al puerto D de un D Flip-flop. Debe notarse que la extracción de parásitos se realizó sólo para los nodos de interés, "net5", "net9" y "net17", pues las compuertas inversoras y el D Flip-flop ya estaban caracterizados en una librería. Después de procesarse con el *script* de Python, el archivo de parásitos no hacía referencia a los nombres en el "*NAME_MAP" de cada transistor y sus terminales, sino a los nombres de las compuertas y sus puertos, e.g. las líneas

```
*I *78:GATE I *C 1.98400 0.486000 *L 0.00810000 // $llx=1.98400 $lly=0.486000 $urx=1.98400 $ury=0.486000 $lvl=23
```

```
  *I *79:GATE I *C 1.98400 1.07600 *L 0.0156000 // $llx=1.98400 $lly=1.07600 $urx=1.98400
$ury=1.07600 $lvl=24
  *I *79:DRN B *C 1.99900 1.07600 *D p105 // $llx=1.99900 $lly=1.07600 $urx=1.99900
$ury=1.07600 $lvl=5
  *I *78:DRN B *C 1.99900 0.486000 *D n105 // $llx=1.99900 $lly=0.486000 $urx=1.99900
$ury=0.486000 $lvl=4
```

cambian a

```
  *I u15:A I *C 1.98400 0.486000 *L 0.00810000 // $llx=1.98400 $lly=0.486000 $urx=1.98400
$ury=0.486000 $lvl=23
  *I u15:A I *C 1.98400 1.07600 *L 0.0156000 // $llx=1.98400 $lly=1.07600 $urx=1.98400
$ury=1.07600 $lvl=24
  *I u15:Y B *C 1.99900 1.07600 *D p105 // $llx=1.99900 $lly=1.07600 $urx=1.99900
$ury=1.07600 $lvl=5
  *I u15:Y B *C 1.99900 0.486000 *D n105 // $llx=1.99900 $lly=0.486000 $urx=1.99900
$ury=0.486000 $lvl=4
```

porque los *gates* de los transistores MM0 y MM1 de la instancia XI15 eran el puerto de entrada de la compuerta, llamado "A", y sus *drains* eran el puerto de salida de la compuerta, llamado "Y". Además de esta modificación al archivo de parásitos, se creó un archivo Verilog que tuviera los mismos puertos de entrada y salida que el circuito, las tres instancias de las compuertas inversoras y la instancia del D Flip-flop. Se usó la librería "saed32rvt_tt0p78v25c.lib" incluida en la librería genérica de 32nm proporcionada por Synopsys (Synopsys, Worldwide University Program, 2016). El *script* de Python se muestra en el Apéndice C. Para usar estos archivos en PrimeTime se usaron los siguientes comandos:

```
  read_lib saed32rvt_tt0p78v25c.lib
  set link_path "* saed32rvt_tt0p78v25c"
  read_verilog CadenaNots.v
  link_design CadenaNots
  set si_enable_analysis true
  read_parasitics -keep_capacitive_coupling starrc_results_modified.spef
```

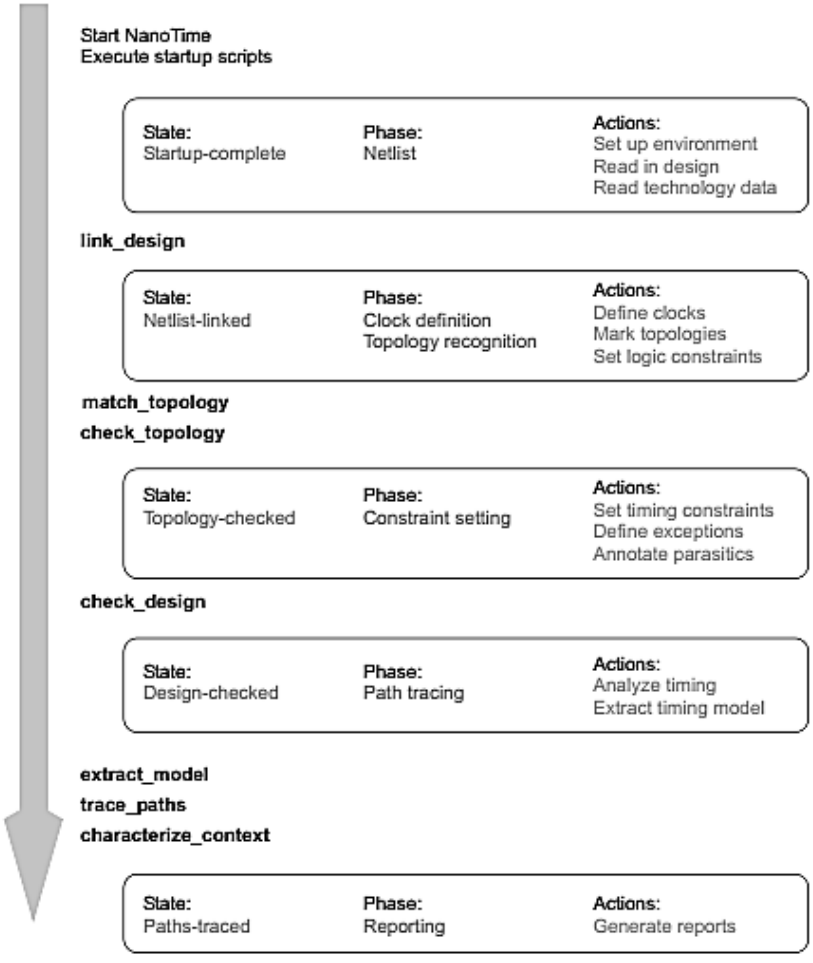
De esta forma se abrían ambos archivos y se vinculaban, para que las compuertas referenciadas en el archivo Verilog tuvieran sus características de comportamiento en el tiempo definidas por la librería, y las interconexiones entre ellas tuvieran sus parásitos definidos por el archivo SPEF. A pesar de que esto fue posible, se encontró un problema al momento de analizar circuitos con compuertas con varios puertos de entrada: no se podía diferenciar fácilmente qué *gate* correspondía a qué entrada. Se buscaron formas de extraer a nivel de compuerta los parásitos en lugar de hacerlo a nivel de transistor, pero resultó imposible. La

referencia (Synopsys, StarRC User Guide and Command Reference, 2016) indica que sí se puede extraer a nivel de compuerta, pero no se encontró una forma de hacerlo. Por simplicidad y para aprovechar las herramientas de Synopsys en lugar de crear herramientas propias, se decidió dejar de usar PrimeTime como herramienta de análisis y usar NanoTime en su lugar.

D. Elaboración del *script* de NanoTime.

La realización del análisis con la herramienta NanoTime consiste en cinco fases: fase de *netlist*, fase de reconocimiento de la topología, fase de restricciones, fase de análisis, y fase de reporte de resultados. Hay varios comandos que sólo deben de ejecutarse antes de o durante ciertas fases, pues de lo contrario la herramienta lanza errores o advertencias. Por lo tanto, es crucial respetar el flujo de análisis que se indica en el manual de usuario de NanoTime (Synopsys, NanoTime, 2016). Las cinco fases de ejecución se pueden ver a grandes rasgos en la Figura 82.

Figura 82. Flujo de análisis con la herramienta NanoTime.



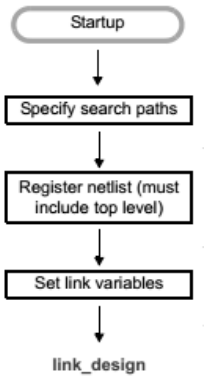
(Synopsys, NanoTime, 2016)

1. Fase de *netlist.* Todo análisis con NanoTime debe comenzar con la fase de *netlist.* La herramienta lee el diseño del circuito y lo enlaza a las librerías de tecnología. Además, todas las variables que afecten el reconocimiento de la topología deben definirse en esta fase. En el archivo de comandos que se creó para el análisis se comenzó definiendo los directorios donde la herramienta debía buscar los archivos que se referenciaran. Esto se hizo con los comandos "set search_path {.}", que busca en el directorio actual, y "set link_path {*}". Debe notarse que NanoTime reconoce los caracteres comodín "*", que representa cualquier cantidad de caracteres, y "?", que representa un caracter. Luego se definieron algunas variables para sincronizar NanoTime con StarRC, con el propósito de que NanoTime entendiera el archivo de parásitos que se generó con StarRC en la extracción de parásitos. Esto incluye definir los nombres de las terminales de los transistores con el comando "set link_transistor_*_pin_name", ayudar a NanoTime a reconocer transistores con múltiples terminales *gate* con el comando "set parasitics_fingered_device_chars \@", e identificar nodos internos del diseño físico que no existen en el esquemático con "set parasitics_xref_layout_instance_prefix ld_". Esta primera fase termina cuando se ejecuta el comando "link_design" con el nombre del subcircuito de interés (Synopsys, NanoTime, 2016). Cabe mencionar que el archivo HSPICE que se lea con la herramienta debe contener el diseño encapsulado en un subcircuito. NanoTime trabaja con un subcircuito que puede descender jerárquicamente para identificar instancias de otros subcircuitos (Roy, 2009). Es necesario que se haga referencia a las librerías con el modelo de los transistores MOSFET que se hayan usado. En este caso, esto se hizo en el archivo HSPICE del diseño analizado con las líneas

```
.option search='/usr/synopsys/iPDK/SAED32_28_iPDK/hspice'
.lib 'saed32nm.lib' TT
```

Si la librería se encuentra en otro directorio se debe modificar la dirección del primer comando. Además, se puede utilizar distintas esquinas de diseño si se modifica el segundo comando. En este caso, "TT" significa que los transistores NMOS y PMOS tienen características típicas. El flujo de la fase de *netlist* se muestra en la siguiente figura.

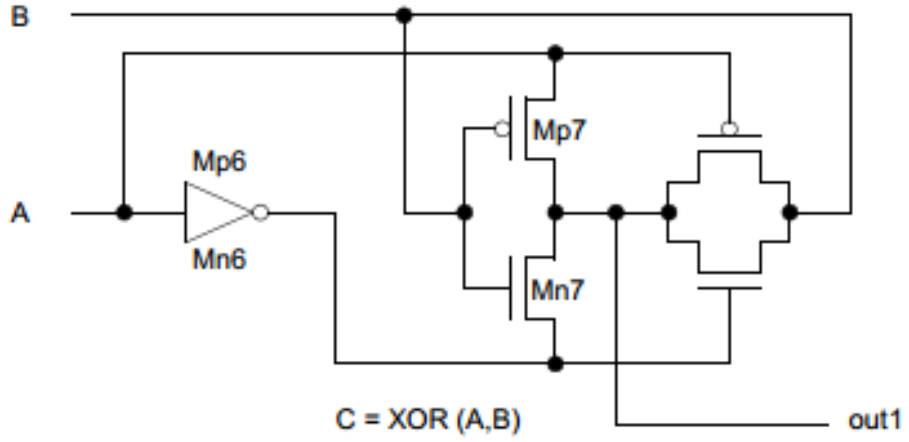
Figura 83. Flujo de la fase de *netlist.*



(Synopsys, NanoTime, 2016)

2. Fase de reconocimiento de la topología y definición del reloj. La fase de reconocimiento de la topología comienza inmediatamente después de ejecutar el comando "link_design". En esta fase se deben especificar las características del diseño. Esto incluye las señales de reloj del circuito y los puertos a donde están conectadas, la dirección de todos los puertos del diseño y los voltajes de alimentación. Se identificó el nombre del nodo de alimentación y de referencia con los comandos "set_supply_net VDD", "set_voltage 1.05 VDD" y "set_supply_net -gnd VSS" para asegurarse de que NanoTime identificara esos puertos correctamente. Con los comandos "set_port_direction -input [get_ports A*]" y "set_port_direction -output [get_ports {OUT* QN*}]" se señalaron los puertos de entrada y de salida del diseño. Debe notarse que "[get_ports A*]" hace referencia a todos los puertos cuyo nombre comience con "A" y que tenga cualquier cantidad de caracteres después. "[get_ports {OUT* QN*}]" se refiere a todos los puertos cuyo nombre empiece con "OUT" o con "QN" y tenga cualquier cantidad de caracteres después. Se recomienda no nombrar los puertos del diseño con caracteres comodín para evitar confusiones. Con el comando "create_clock -period 20 -name MCLK [get_ports CLK*]" se propaga una señal de reloj llamada "MCLK" que se conecta a todos los puertos cuyo nombre comienza con "CLK". Los nombres que se usaron en este caso pueden variar según el diseño que se pretenda analizar. Se deben especificar análisis de caso y restricciones lógicas sólo si afectan el reconocimiento de la topología. De lo contrario se puede hacer en la fase de restricciones. El comando "match_topology" hace que NanoTime reconozca la topología del diseño automáticamente, y se pueden nombrar estructuras específicas que la herramienta debería buscar al agregar el parámetro "-structure_types". El reconocimiento automático está disponible para muchas estructuras específicas. A pesar de esto, no todas las implementaciones de una estructura con la misma función se reconocen. Un ejemplo claro de esto se da con la compuerta XOR, pues NanoTime sólo reconoce automáticamente la implementación con seis transistores que se muestra en la Figura 84 (Synopsys, NanoTime, 2016).

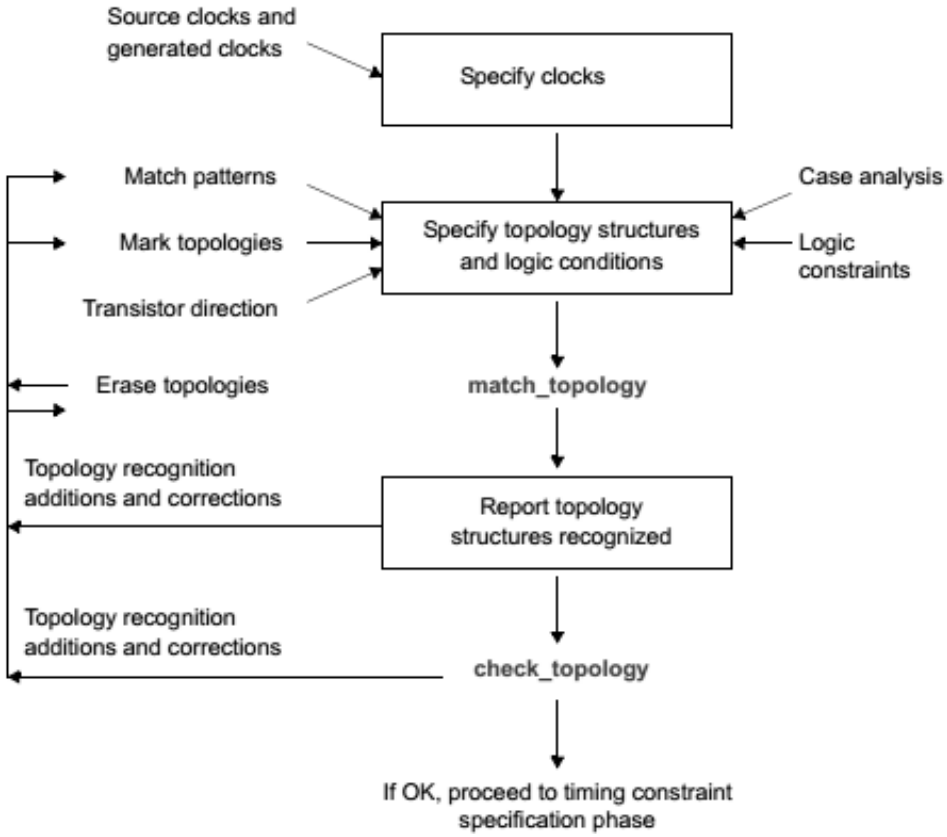
Figura 84. Implementación de compuerta XOR reconocida automáticamente por NanoTime.



(Synopsys, NanoTime, 2016)

Que no reconozca la estructura no implica que haya un error, pues al propagar las señales de prueba en la fase de análisis el diseño funcionará como debería. Además, al identificar una estructura como un D Flip-flop, que se implementa usando compuertas inversoras y *latches*, la identificación automática reconoce tanto las compuertas como los *latches* y el D Flip-flop. La fase de reconocimiento de la topología finaliza cuando se ejecuta el comando "check_topology" (Synopsys, NanoTime, 2016).

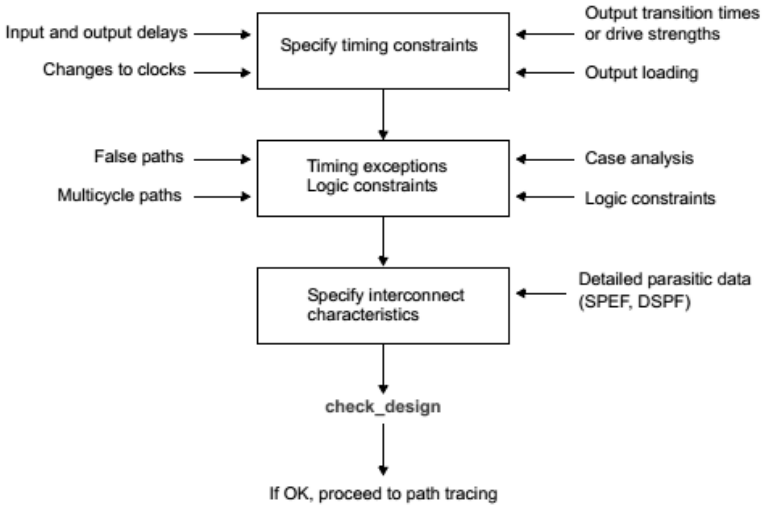
Figura 85. Flujo de la fase de reconocimiento de la topología.



(Synopsys, NanoTime, 2016)

3. Fase de restricciones. La fase de restricciones comienza después de ejecutar el comando "check_topology". En esta fase se especifican todas las restricciones sobre el funcionamiento del circuito. Los tipos de restricciones más comunes son: características del reloj, retardos de las señales en las entradas y salidas, características de los *drivers* en las entradas y las cargas capacitivas en las salidas, y datos sobre los parásitos del circuito (Synopsys, NanoTime, 2016). El flujo de la fase de restricciones se muestra en la Figura 86.

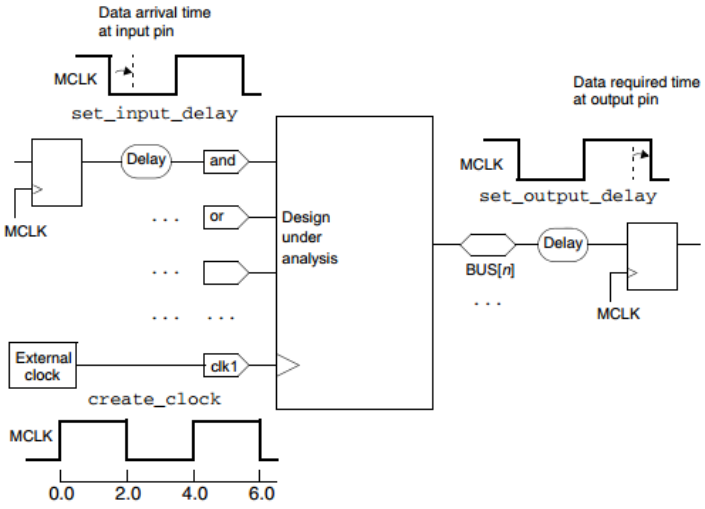
Figura 86. Flujo de la fase de restricciones.



(Synopsys, NanoTime, 2016)

Los retardos en los puertos del diseño se especificaron con los comandos "set_input_delay -clock MCLK ..." y "set_output_delay -clock MCLK ...". El número indicado después de "set_input_delay -clock MCLK" indica el retraso mínimo y máximo permitido desde un flanco de la señal de reloj hasta que una señal llega al puerto especificado. NanoTime usa esta información para buscar violaciones de tiempo en ese puerto de entrada. La transición en los puertos de entrada depende de los *drivers* externos. Sus características se pueden definir con el comando "set_drive" o "set_input_transition". El comando "set_output_delay" especifica el retraso mínimo y máximo permitido desde el puerto de salida del diseño hasta un registro externo que capture el dato de ese puerto. Esto establece los tiempos en que la señal de salida debe estar disponible en el puerto para cumplir con requisitos de *setup time* y *hold time*.

Figura 87. Restricciones de retrasos en las entradas y las salidas.



(Synopsys, NanoTime, 2016)

En el archivo de comandos que se creó también se enlazó el archivo de parásitos con el resto del diseño. NanoTime sólo interpreta archivos en formato DSPF, SPEF y SBPF. Como se mencionó previamente, se generó un archivo DSPF con la herramienta StarRC. Para realizar el análisis de integridad de señales se ejecutaron los siguientes comandos:

```
set si_enable_noise_analysis true
set si_enable_analysis true
read_parasitics -keep_capacitive_coupling starrc_results.spf
```

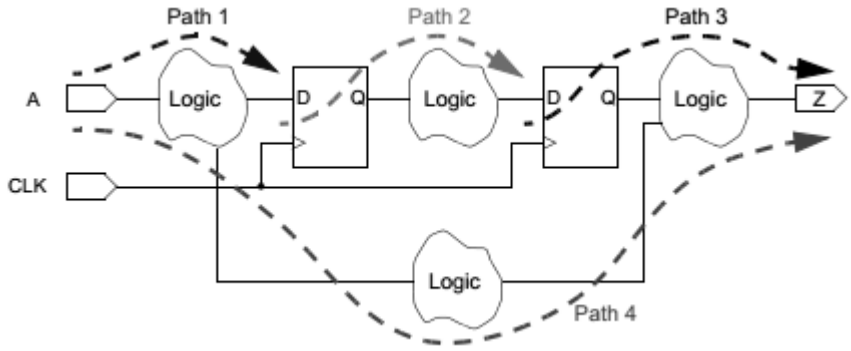
Luego de hacer esto, se finalizó la fase de restricciones con el comando "check_design -complete_with zero", acorde a una recomendación del artículo de Solvnet (Synopsys, Handling of Macro-Model Parasitics in NanoTime, 2009).

4. Fase de análisis y de reporte. Las últimas dos fases, la de análisis y de reporte se realizaron con los siguientes tres comandos:

```
trace_paths
report_paths -max
report_paths –min
```

Con el primero, NanoTime valida la funcionalidad del diseño pues revisa todas las rutas en busca de violaciones de tiempo. Para cada ruta, NanoTime calcula el retardo en la propagación de la señal. El comienzo de una ruta puede ser un puerto de entrada del diseño, un puerto de reloj, una celda secuencial, un *latch* o un puerto con un retraso de entrada especificado. El fin de una ruta puede ser un puerto de salida del diseño, un puerto de reloj, una celda secuencial, o un puerto con un retraso de salida especificado. Esto se muestra claramente en la Figura 88.

Figura 88. Posibles rutas analizadas por NanoTime en la fase de análisis.



(Synopsys, NanoTime, 2016)

Una estructura de lógica combinacional puede tener varias rutas, y NanoTime se encarga de analizarlas todas. La herramienta usa la ruta más larga para calcular el retraso máximo y la ruta más corta para calcular el retraso mínimo. El comando "report_paths" reporta los resultados del peor caso del análisis.

5. Corrección de algunos errores comunes. Se encontraron algunos errores cuando se intentó abrir el archivo de parásitos. El error más común fue el de PARA-040, pues puede significar varios problemas. Este error indica que NanoTime descartó algunos capacitores parásitos porque tenían errores. La solución a esto fue forzar a StarRC a generar las instancias de transistores como subcircuitos con los comandos "HN_NETLIST_SPICE_TYPE: n105 X" y "HN_NETLIST_SPICE_TYPE: p105 X". Además, se leyó el archivo DPF que se generó en la extracción de parásitos con el comando "read_device_parameters ideal_spice.dpf". Otra posibilidad para arreglar este error es que la forma de reconocer la jerarquía de NanoTime no esté sincronizada con la forma en que StarRC la identificó. Para esto se usa la opción "HIERARCHICAL_SEPARATOR: ." en StarRC y "set hierarchy_separator .", donde el "." es el separador de la jerarquía del diseño.

E. Pruebas de validación.

Para pruebas iniciales se usaron circuitos sin mayor complejidad para verificar que los resultados obtenidos por NanoTime fueran resultados coherentes con lo esperado. Las simulaciones se hicieron con un *script* de NanoTime que tiene las fases explicadas en la sección anterior. El *script* elaborado completo se encuentra en el anexo E. Los circuitos para validar los resultados fueron:

1. Cadena de ocho compuertas inversoras. Se diseñó una cadena con ocho compuertas inversoras y dos D Flip-flops, como se puede notar en la Figura 89. El circuito no realiza ninguna función en específico. La finalidad del circuito es poder notar el efecto de las capacitancias de acople al tener varias interconexiones.

Figura 89. *Layout* del circuito de ocho compuertas inversoras y dos D Flip-flops.

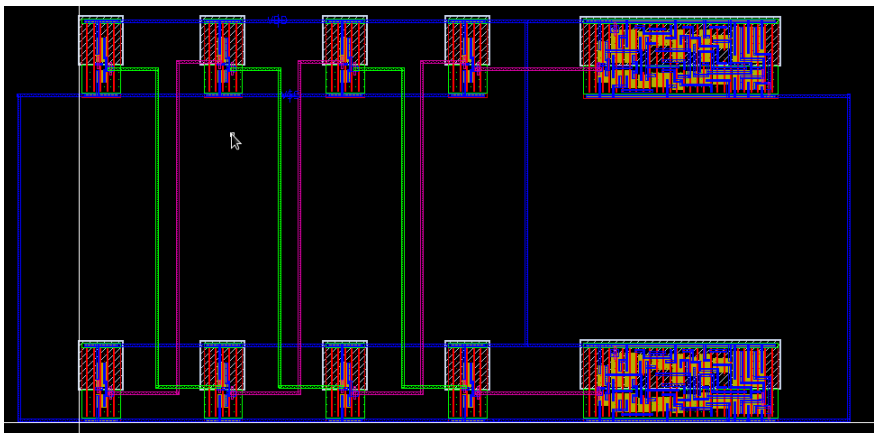
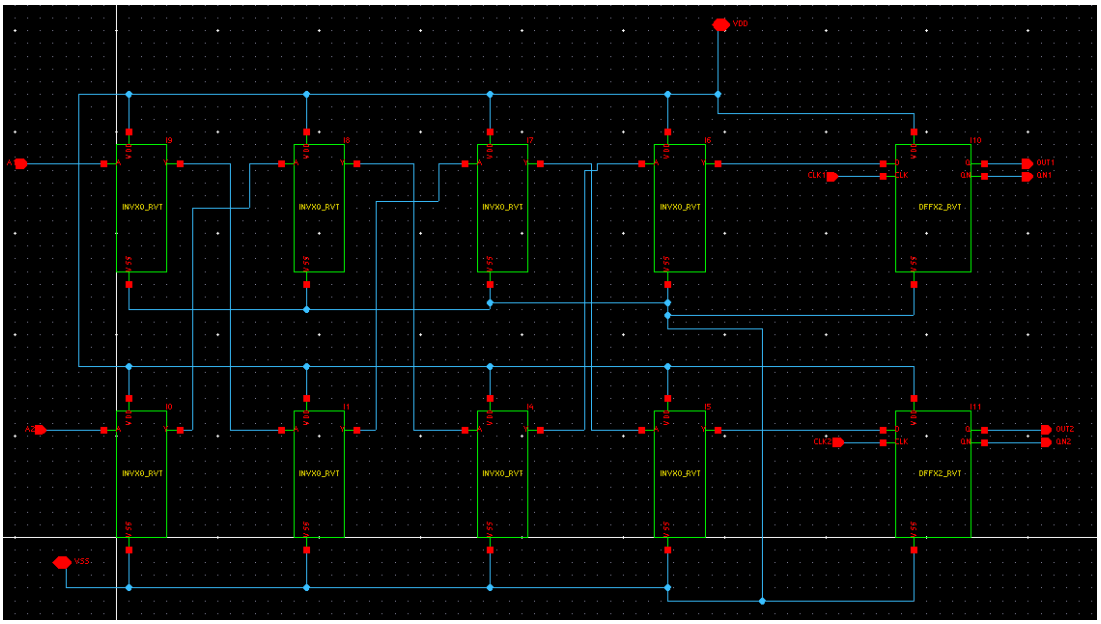


Figura 90. Circuito esquemático de ocho compuertas inversoras y dos D Flip-flops.



Luego de haber pasado las tres verificaciones, se procedió a analizar la extracción de parásitos con NanoTime corriendo en la línea de comandos el *script* de la siguiente manera:

```
nt_shell –file script_name.tcl
```

Este comando primero ejecuta el archivo con extensión ".tcl" y luego abre la línea de comandos. Existe otra forma de ejecutar el *script*:

```
nt_shell
source script_name.tcl
```

Este comando primero abre la línea de comando y luego ejecuta el archivo .tcl. Cualquiera de las dos formas es válida para ejecutar el *script* de NanoTime.

Por último se analizaron los resultados obtenidos.

2. Comparador de identidad de dos bits. Se diseñó un comparador de identidad de dos bits con *standard cells* y poder hacer las interconexiones cómo se muestra en la figura siguiente con la finalidad de tener capacitancias parásitas. Se eligió este circuito con el objetivo de realizar pruebas con compuertas que no fueran inversoras y poder validar los resultados obtenidos con NanoTime.

Figura 91. *Layout* del circuito comparador de identidad de dos bits.

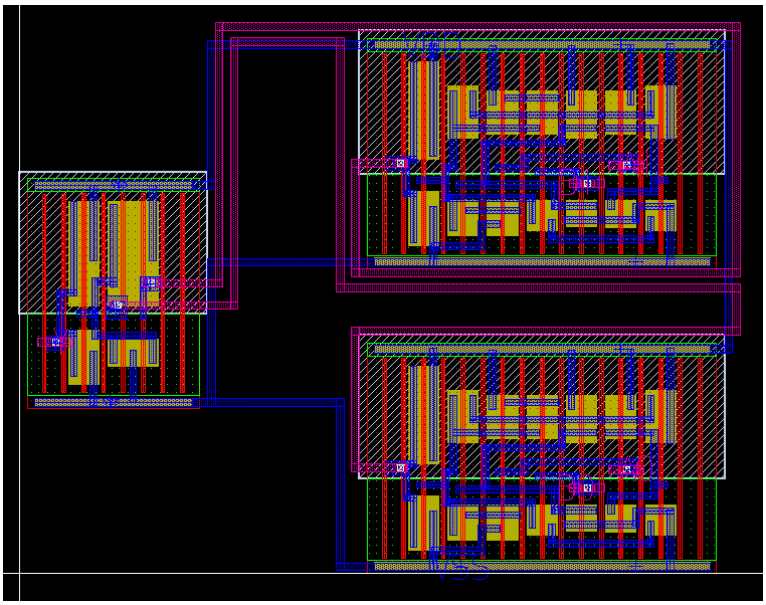
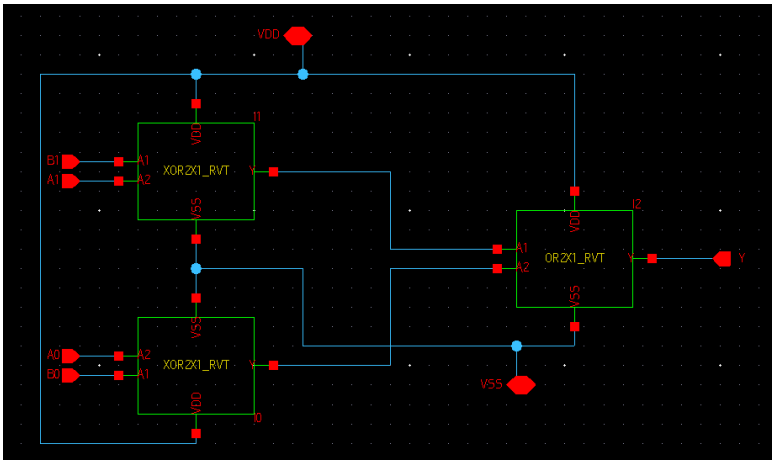


Figura 92. Circuito esquemático de un comparador de identidad de dos bits.



Al tener el *layout* y el esquemático del comparador de identidad de dos bits, se realizaron las verificaciones de DRC, LVS y LPE.

Luego de haber pasado las tres verificaciones, se analizó la extracción de parásitos con NanoTime ejecutando en la línea de comandos el *script* de la siguiente manera:

```
nt_shell -file script_name.tcl
```

Este comando primero ejecuta el archivo .tcl y luego abre la línea de comandos. Existe otra forma de ejecutar el *script*:

```
nt_shell
source script_name.tcl
```

Este comando primero abre la línea de comando y luego ejecuta el archivo .tcl. Cualquiera de las dos formas es válida para ejecutar el *script* de NanoTime.

Por último se analizaron los resultados obtenidos.

3. D Flip-flop. Se usó una *standard cell* de un D Flip-flop de la librería instalada de 32nm y se construyó el circuito esquemático como se muestra en las siguientes dos figuras. Se decidió hacer pruebas con una *standard cell* para comparar el resultado obtenido por NanoTime con el resultado que índica el manual de la librería que se encuentra dentro de los archivos de instalación de la librería.

Figura 93. *Layout* de una celda estándar D Flip-flop de flanco positivo.

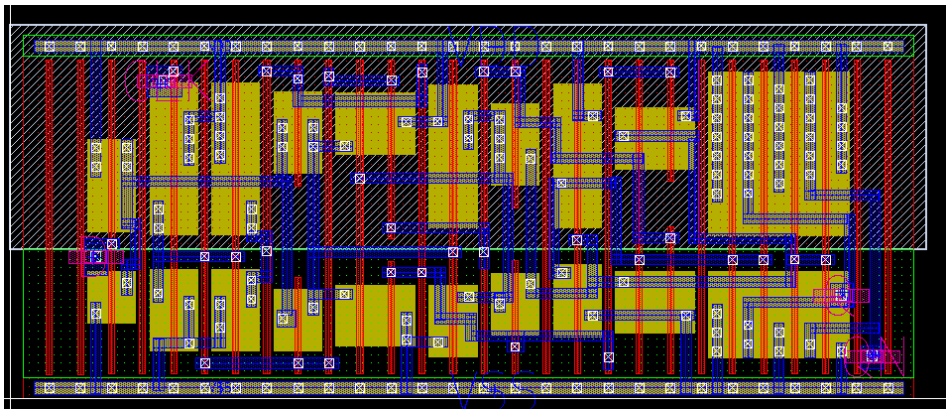
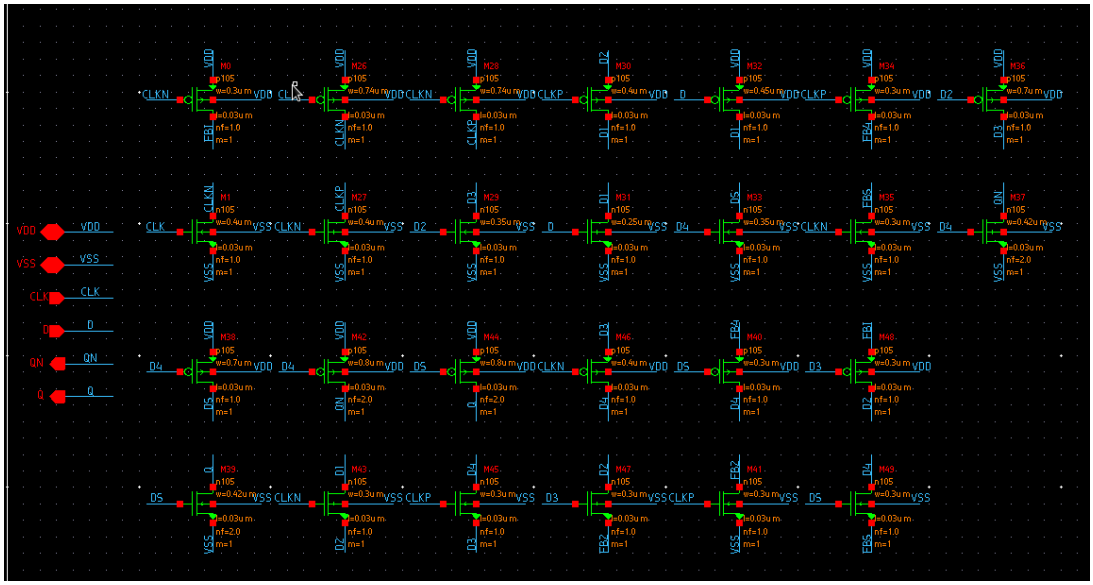


Figura 94. Circuito esquemático de un D Flip-flop estándar.



Al tener el *layout* y el esquemático del D Flip-flop, se procedió a realizar las verificaciones de DRC, LVS y LPE. Luego de haber pasado las tres verificaciones, se procedió a analizar la extracción de parásitos con NanoTime corriendo en la terminal (línea de comandos) el *script* de la siguiente manera:

```
nt_shell –file script_name.tcl
```

Este comando primero ejecuta el archivo .tcl y luego abre la línea de comandos. Existe otra forma de ejecutar el *script*:

```
nt_shell
source script_name.tcl
```

Este comando primero abre la línea de comando y luego ejecuta el archivo .tcl. Cualquiera de las dos formas es válida para ejecutar el *script* de NanoTime. Por último se analizaron los resultados obtenidos.

F. Obtención de curva de transferencia DC.

Para la investigación fue crucial obtener la curva de transferencia DC para definir los rangos de voltaje que serían valores lógicos válidos. Fue necesario contar con un circuito de una compuerta inversora con el flujo de diseño ya completado. En este caso se usó la celda estándar INVX1_RVT. Además, a este circuito se le extrajeron los parásitos como se explicó previamente con el *script* de StarRC modificado. Para conseguir la curva de transferencia DC se usó un *script* de HSPICE que incluía los parásitos extraídos del circuito más los siguientes comandos:

```
.DC var1 START=start1 STOP=stop1 STEP=incr1
.PRINT V(A) V(Y)
.PLOT DC (A,Y)
.alter
.PARAM ...
```

Esto permitió hacer un barrido en la entrada del circuito y definir el paso, el punto de inicio y el punto final. Debido a que esto se repitió para varios voltajes de operación, tanto en superumbral como en subumbral, se usó el comando .alter para modificar el valor del punto final. En cada simulación se graficaba el voltaje de entrada versus el voltaje de salida y estos datos se guardaban. Con esta información ya disponible post-simulación, se usó Microsoft Excel para calcular la pendiente de la curva en cada punto. Se identificaron los puntos con pendiente igual a -1, pues por definición son los límites del voltaje válido (Harris & Weste, CMOS VLSI Design: A Circuits and Systems Perspective, 2011). En la compuerta inversora, el voltaje en la entrada aceptado como un ‘1’ lógico es desde 0V hasta el punto donde la pendiente incrementó hasta -1. El voltaje

en la entrada aceptado como un ‘0’ lógico es desde el punto donde la pendiente disminuyó hasta -1 hasta VDD voltios.

G. Obtención de curva de rechazo de ruido.

La curva de rechazo de ruido es importante para complementar a la curva de transferencia DC. Se puede notar que la curva de rechazo de ruido sí relaciona el tiempo de duración del ruido con el voltaje de salida. Por esta razón también fue crucial obtener la curva de rechazo de ruido, pues un ruido en un nodo sería considerado violación sólo si tenía cierto voltaje pico por cierta cantidad de tiempo. Este procedimiento se realizó para una compuerta inversora y para un D Flip-flop de flanco negativo.

1. Compuerta inversora. Para obtener la curva de rechazo de ruido para la compuerta inversora se usó el mismo archivo de HSPICE que se usó para obtener su curva de transferencia DC. Sin embargo, se modificó para agregar los siguientes comandos de HSPICE:

```
.tran 1e-12 20e-9 SWEEP vpk_norm 0.05 1 0.05
.measure tran medicion trig=v(Y) val='0.95*vdd' fall=1 targ=v(Y) val=threshold cross=1
.alter .PARAM ...
```

Ya que la curva sí considera el tiempo, fue necesario hacer un análisis transciente, y dentro de cada corrida del análisis se hizo un barrido sobre una fuente. Se midió el tiempo que le tomaba a la salida de la compuerta cruzar por un umbral definido a partir de los valores encontrados con la curva de transferencia DC. Si el ruido no afectaba a la compuerta, esta medición fallaría. Por lo tanto, se graficó el primer voltaje de entrada y duración correspondiente que produjera una medición exitosa. La curva producida indicaría entonces la altura del pico del ruido necesaria para que cada duración de tiempo causara un error en la salida.

2. D Flip-flop de flanco negativo. Para obtener la curva de rechazo de ruido para el D Flip-flop de flanco negativo se completó el flujo de diseño para la celda estándar DFFNARX1_RVT. Con sus parásitos extraídos se agregaron los siguientes comandos de HSPICE al archivo:

```
.tran 5e-11 1000e-9 SWEEP vpk_norm 0.05 1 0.05
.meas tran medicionQ trig=v(Q) val='0.95*vdd' fall=1 targ=v(Q) val='0.05*vdd' fall=1
.alter para modificar el ancho del ruido
```

En este caso, el D Flip-flop se inicializaba con su salida en ‘1’ lógico y se esperaba que grabara un ‘0’ lógico. Por lo tanto, se esperaba que la medición fuera exitosa, pues esta medición contaba el tiempo desde que la salida cruzaba el valor de 95% de VDD hasta que cruzara el valor de 5% de VDD. El procedimiento fue el mismo que para la compuerta inversora.

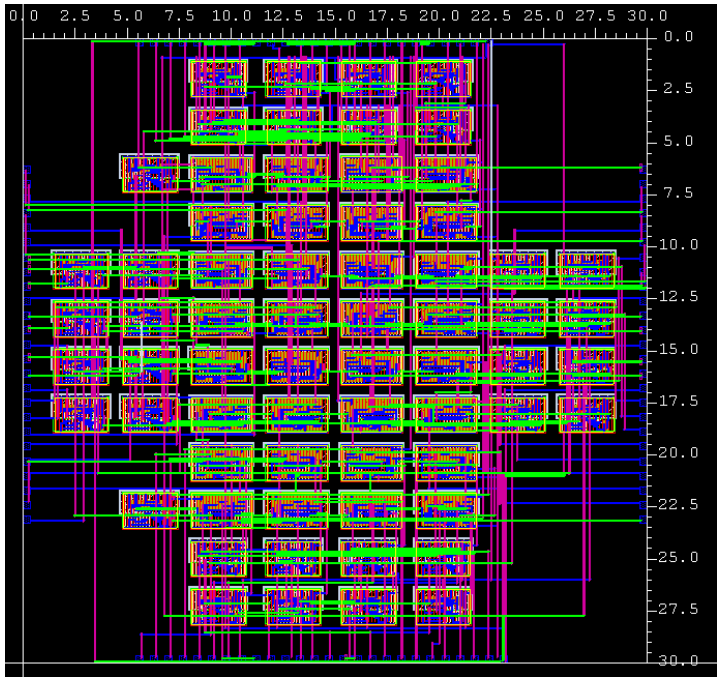
H. Análisis del ruido inyectado.

El análisis de ruido inyectado fue necesario para comparar directamente si se inyectaba más ruido en operación subumbral o superumbral. Para esto se construyó un circuito con tres compuertas inversoras con celdas estándar INVX1_RVT. En este circuito, dos compuertas eran agresoras y la tercera, rodeada por las otras dos, era la víctima de interés. Las entradas de las dos compuertas agresoras se controlaron simultáneamente y se hizo que conmutaran tanto en la misma dirección como en dirección opuesta a la entrada de la compuerta víctima. Se midió el ruido en el *interconnect* en las salidas de las compuertas. Este *interconnect* estaba en la capa de metal 3 y su longitud se varió desde 50µm hasta 1mm. Con la herramienta NanoTime se reportó la altura del pico del ruido para distintos voltajes de alimentación, tanto de superumbral como de subumbral. Esta altura se normalizó respecto al voltaje de alimentación para permitir una comparación directa, y se graficó.

I. Sumador/Restador Ripple-carry de 32 bits.

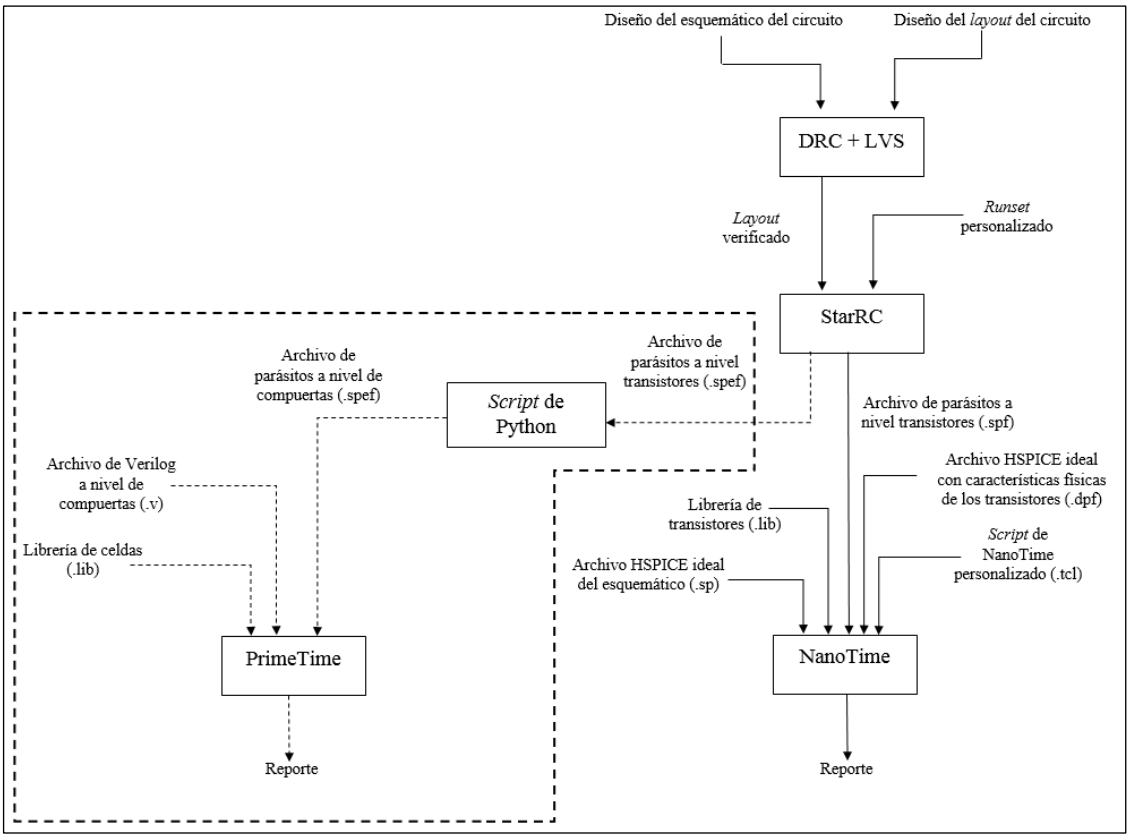
Con los resultados obtenidos de las curvas de transferencia DC y de rechazo de ruido, más el análisis del ruido inyectado, se definieron los parámetros para que NanoTime filtrara el ruido y reportara sólo el ruido que superara esos umbrales. Sin embargo, NanoTime no filtraba el reporte considerando también la duración del ruido, por lo que fue necesario hacer un *script* de Python en el que se introdujeran los datos de la curva de rechazo de ruido para filtrar el reporte de NanoTime. El diseño del Sumador/Restador *Ripple-carry* de 32 *bits* se muestra a continuación.

Figura 95. *Layout* del Sumador/Restador *Ripple-carry* de 32 *bits*.



VI. RESULTADOS

Figura 96. Flujo requerido para el análisis de *timing* de un circuito.



Cuadro 2. Retardo máximo obtenido con el circuito de la cadena de compuertas inversoras.

```
****************************************
Report : paths
        -path_type list
        -max
Design : CadenaNots
Version: L-2016.06-SP2
Date   : Fri Oct  7 12:38:04 2016
****************************************

            Path  Path
   Slack    Delay  Type    Startpoint                        Endpoint
-------- -------- ----- -- ------------------------------ -- ------------------------------
   9.806    0.232   D-L r  A2                             r  XI10.XMM33.main.GATE
   9.806    0.232   D-L r  A2                             r  XI10.XMM37.main.GATE
   9.807    0.232   D-L r  A1                             r  XI11.XMM33.main.GATE
   9.807    0.232   D-L r  A1                             r  XI11.XMM37.main.GATE
   9.821    0.217   D-L f  A2                             f  XI10.XMM38.main.GATE
   9.821    0.217   D-L f  A2                             f  XI10.XMM42.main.GATE
   9.822    0.217   D-L f  A1                             f  XI11.XMM38.main.GATE
   9.822    0.217   D-L f  A1                             f  XI11.XMM42.main.GATE
   9.825    0.175   C-O r  CLK2                           f  OUT2
   9.825    0.175   C-O r  CLK1                           f  OUT1
   9.847    0.153   C-O r  CLK2                           r  OUT2
   9.848    0.152   C-O r  CLK1                           r  OUT1
   9.870    0.130   C-O r  CLK2                           r  QN2
   9.870    0.130   C-O r  CLK1                           r  QN1
   9.873    0.165   C-L f  CLK2                           f  XI11.XMM38.main.GATE
   9.873    0.165   C-L f  CLK2                           f  XI11.XMM42.main.GATE
   9.876    0.162   C-L f  CLK1                           f  XI10.XMM38.main.GATE
   9.876    0.162   C-L f  CLK1                           f  XI10.XMM42.main.GATE
   9.877    0.161   D-L r  A2                             f  XI10.XMM36.main.GATE
   9.877    0.160   D-L r  A1                             f  XI11.XMM36.main.GATE
   9.886    0.114   C-O r  CLK2                           f  QN2
   9.886    0.114   C-O r  CLK1                           f  QN1
   9.898    0.140   C-L f  CLK2                           r  XI11.XMM33.main.GATE
   9.898    0.140   C-L f  CLK2                           r  XI11.XMM37.main.GATE
   9.899    0.139   C-L f  CLK1                           r  XI10.XMM33.main.GATE
   9.899    0.139   C-L f  CLK1                           r  XI10.XMM37.main.GATE
   9.905    0.132   D-L f  A2                             r  XI10.XMM29.main.GATE
   9.905    0.132   D-L f  A1                             r  XI11.XMM29.main.GATE
```

Cuadro 3. Resultados del retardo mínimo obtenido con el circuito de la cadena de compuertas inversoras.

```
****************************************
Report : paths
        -path_type list
        -min
Design : CadenaNots
Version: L-2016.06-SP2
Date   : Fri Oct  7 12:38:04 2016
****************************************

            Path  Path
   Slack    Delay  Type    Startpoint                          Endpoint
-------- -------- ----- -- ------------------------------ -- ------------------------------
   0.077    0.129   C-L f  CLK2                           r  XI11.XMM37.main.GATE
   0.077    0.129   C-L f  CLK2                           r  XI11.XMM33.main.GATE
   0.078    0.131   C-L f  CLK1                           r  XI10.XMM33.main.GATE
   0.078    0.131   C-L f  CLK1                           r  XI10.XMM37.main.GATE
   0.107    0.160   C-L f  CLK1                           f  XI10.XMM42.main.GATE
   0.107    0.160   C-L f  CLK1                           f  XI10.XMM38.main.GATE
   0.108    0.160   C-L f  CLK2                           f  XI11.XMM42.main.GATE
   0.108    0.160   C-L f  CLK2                           f  XI11.XMM38.main.GATE
  10.062    0.125   D-L f  A2                             r  XI10.XMM29.main.GATE
  10.064    0.127   D-L f  A1                             r  XI11.XMM29.main.GATE
  10.084    0.147   D-L r  A1                             f  XI11.XMM36.main.GATE
  10.085    0.148   D-L r  A2                             f  XI10.XMM36.main.GATE
  10.111    0.111   C-O r  CLK1                           f  QN1
  10.112    0.112   C-O r  CLK2                           f  QN2
  10.128    0.128   C-O r  CLK1                           r  QN1
  10.128    0.128   C-O r  CLK2                           r  QN2
  10.150    0.150   C-O r  CLK1                           r  OUT1
  10.150    0.150   C-O r  CLK2                           r  OUT2
  10.173    0.173   C-O r  CLK1                           f  OUT1
  10.173    0.173   C-O r  CLK2                           f  OUT2
```

Cuadro 4. Resultados del retardo máximo con el circuito comparador de identidad de dos *bits*.

```
****************************************
Report : paths
        -path_type list
        -max
Design : COMPARADOR_2bits
Version: L-2016.06-SP2
Date   : Fri Oct  7 12:35:08 2016
****************************************

            Path  Path
   Slack    Delay  Type    Startpoint                          Endpoint
-------- -------- ----- -- ------------------------------ -- ------------------------------
  -0.222    0.222   D-O f  A1                             r  Y
  -0.212    0.212   D-O f  A0                             r  Y
  -0.194    0.194   D-O r  B1                             r  Y
  -0.186    0.186   D-O f  B1                             r  Y
  -0.183    0.183   D-O r  B0                             r  Y
  -0.179    0.179   D-O r  A1                             r  Y
  -0.175    0.175   D-O f  B0                             r  Y
  -0.172    0.172   D-O r  A1                             f  Y
  -0.168    0.168   D-O r  A0                             r  Y
  -0.168    0.168   D-O r  B1                             f  Y
  -0.165    0.165   D-O r  A0                             f  Y
  -0.161    0.161   D-O r  B0                             f  Y
  -0.153    0.153   D-O f  A1                             f  Y
  -0.147    0.147   D-O f  A0                             f  Y
  -0.141    0.141   D-O f  B1                             f  Y
  -0.135    0.135   D-O f  B0                             f  Y
```

Cuadro 5. Resultados del retardo mínimo con el circuito comparador de identidad de dos *bits*.

```
****************************************
Report : paths
        -path_type list
        -min
Design : COMPARADOR_2bits
Version: L-2016.06-SP2
Date   : Fri Oct  7 12:35:08 2016
****************************************

            Path  Path
   Slack    Delay  Type    Startpoint                       Endpoint
-------- -------- ----- -- ------------------------------ -- ------------------------------
  20.121    0.121   D-O f  B0                             f  Y
  20.127    0.127   D-O f  B1                             f  Y
  20.131    0.131   D-O f  A0                             f  Y
  20.138    0.138   D-O f  A1                             f  Y
  20.143    0.143   D-O r  B0                             f  Y
  20.147    0.147   D-O r  A0                             r  Y
  20.149    0.149   D-O r  A0                             f  Y
  20.150    0.150   D-O r  B1                             f  Y
  20.154    0.154   D-O f  B0                             r  Y
  20.155    0.155   D-O r  A1                             f  Y
  20.155    0.155   D-O r  B0                             r  Y
  20.157    0.157   D-O r  A1                             r  Y
  20.164    0.164   D-O f  B1                             r  Y
  20.165    0.165   D-O r  B1                             r  Y
  20.181    0.181   D-O f  A0                             r  Y
  20.191    0.191   D-O f  A1                             r  Y
```

Cuadro 6. Resultados del retardo máximo con la *standard cell* del D Flip-flop.

```
****************************************
Report : paths
        -path_type list
        -max
Design : Flip_flop
Version: L-2016.06-SP2
Date   : Fri Oct  7 12:25:02 2016
****************************************

            Path  Path
   Slack    Delay  Type    Startpoint                       Endpoint
-------- -------- ----- -- ------------------------------ -- --------------------
  -8.178    0.178   C-O r  CLK                            f  Q
  -8.155    0.155   C-O r  CLK                            r  Q
  -8.133    0.133   C-O r  CLK                            r  QN
  -8.117    0.117   C-O r  CLK                            f  QN
  -8.036    0.072   D-L r  D                              f  XMM36.main.GATE
  -8.010    0.047   D-L f  D                              r  XMM29.main.GATE
  -0.085   -7.877   D-L f  D                              f  XMM38.main.GATE
  -0.085   -7.877   D-L f  D                              f  XMM42.main.GATE
  -0.076   -7.887   D-L r  D                              r  XMM33.main.GATE
  -0.076   -7.887   D-L r  D                              r  XMM37.main.GATE
   0.872    0.165   C-L f  CLK                            f  XMM38.main.GATE
   0.872    0.165   C-L f  CLK                            f  XMM42.main.GATE
   0.890    0.147   C-L f  CLK                            r  XMM33.main.GATE
   0.890    0.147   C-L f  CLK                            r  XMM37.main.GATE
```

Cuadro 7. Resultados del retardo mínimo con la *standard cell* del D Flip-flop

```
****************************************
Report : paths
        -path_type list
        -min
Design : Flip_flop
Version: L-2016.06-SP2
Date   : Fri Oct  7 12:25:02 2016
****************************************

            Path  Path
   Slack    Delay  Type    Startpoint                       Endpoint
-------- -------- ----- -- ------------------------------ -- --------------------
   0.079    0.132   C-L f  CLK                            r  XMM37.main.GATE
   0.079    0.132   C-L f  CLK                            r  XMM33.main.GATE
   0.102    0.155   C-L f  CLK                            f  XMM38.main.GATE
   0.102    0.155   C-L f  CLK                            f  XMM42.main.GATE
   9.976    0.041   D-L f  D                              r  XMM29.main.GATE
   9.995    0.061   D-L r  D                              f  XMM36.main.GATE
  10.111    0.111   C-O r  CLK                            f  QN
  10.126    0.126   C-O r  CLK                            r  QN
  10.150    0.150   C-O r  CLK                            r  Q
  10.171    0.171   C-O r  CLK                            f  Q
```

Figura 97. Curva de transferencia DC para una compuerta inversora con VDD de 1.3V.

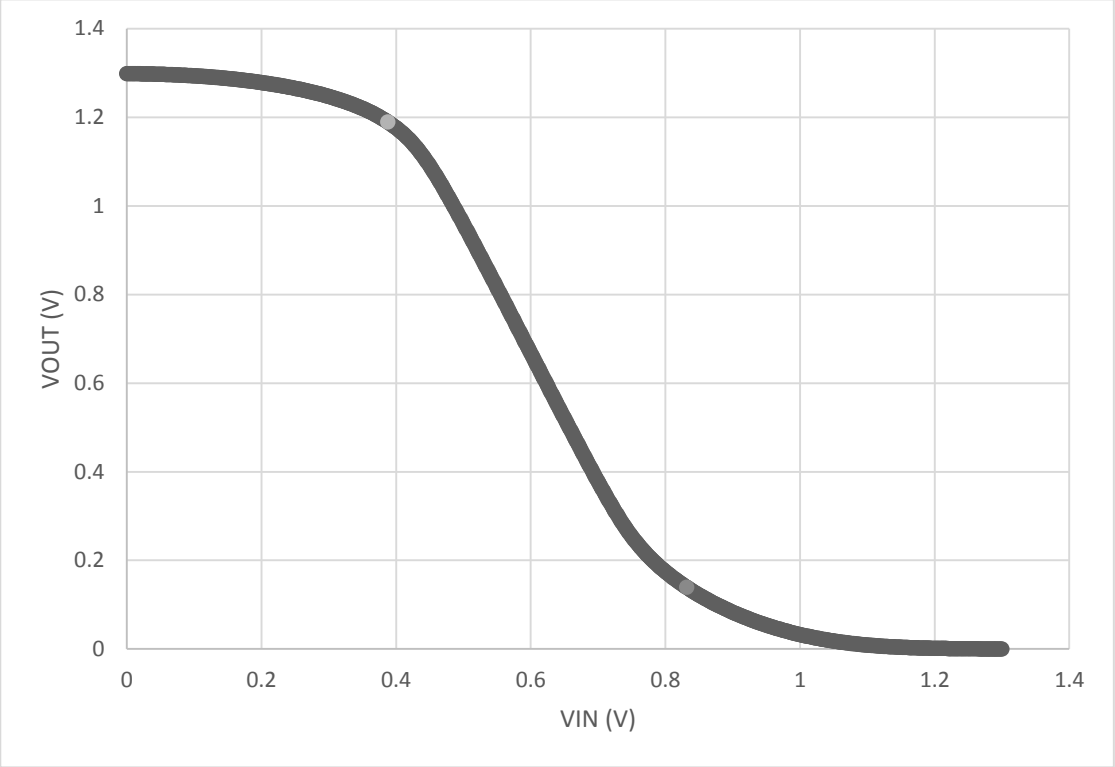


Figura 98. Curva de transferencia DC para una compuerta inversora con VDD de 0.5V.

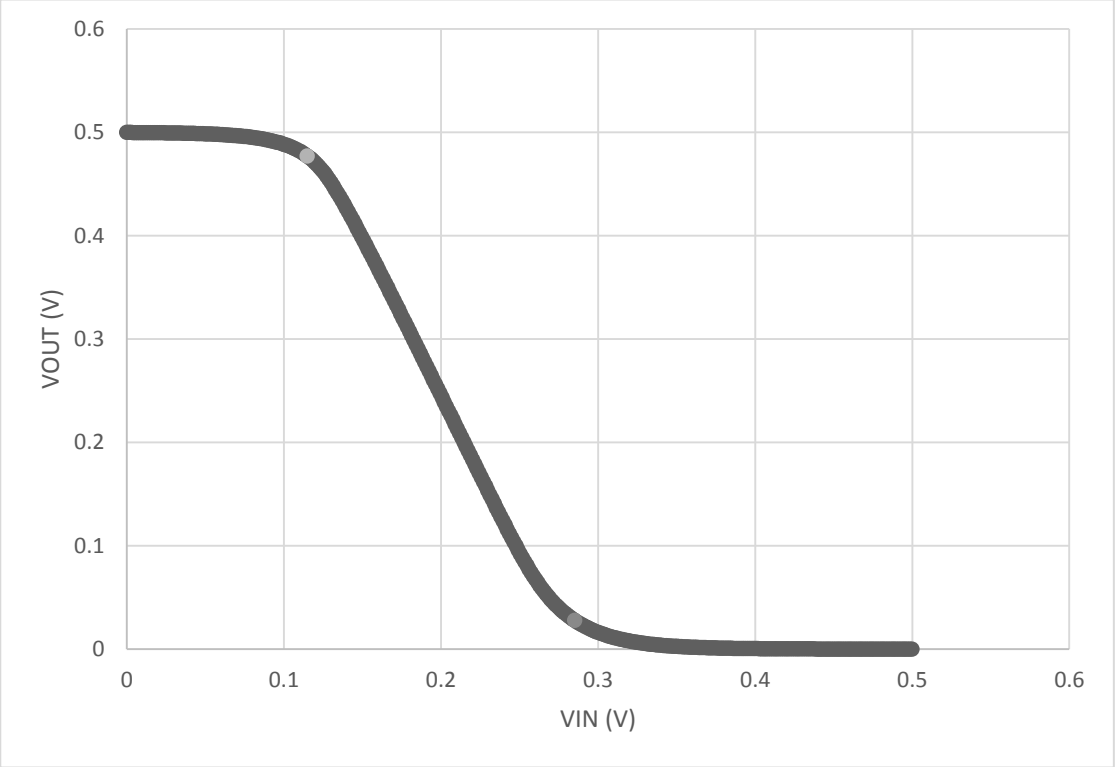


Figura 99. Curva de transferencia DC para una compuerta inversora con VDD de 0.292V

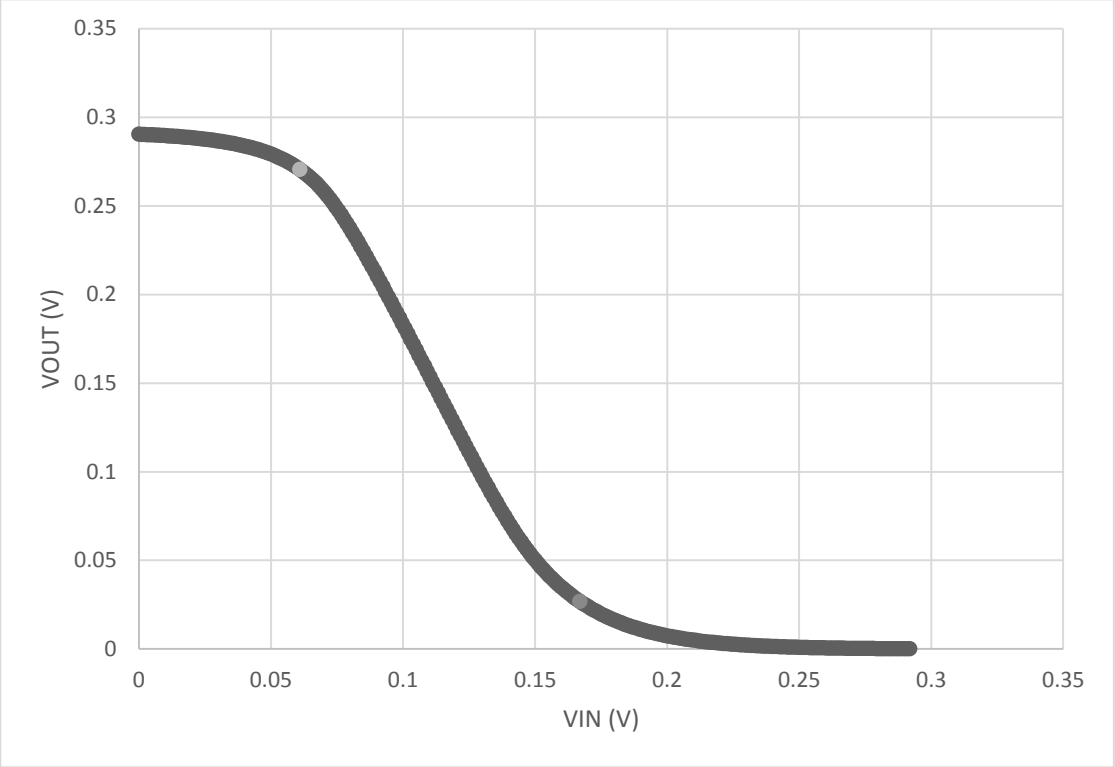


Figura 100. Curva de transferencia DC para una compuerta inversora con VDD de 0.2V.

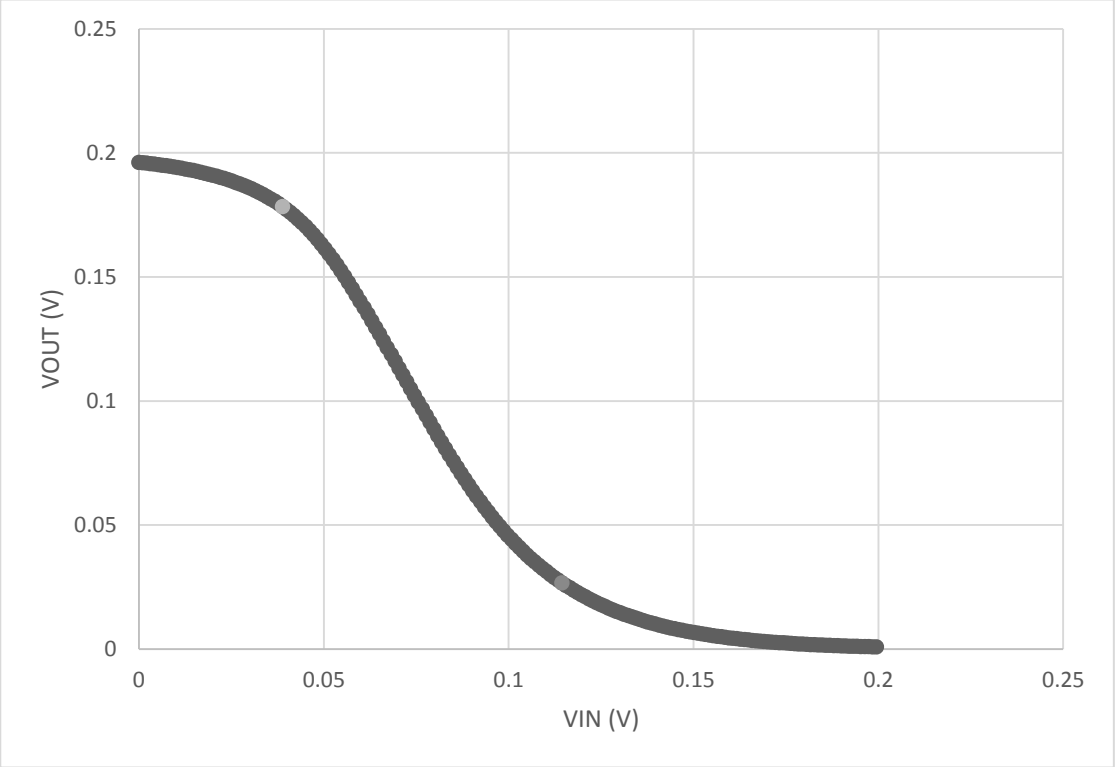


Figura 101. Curva de transferencia DC para una compuerta inversora con VDD de 0.1V.

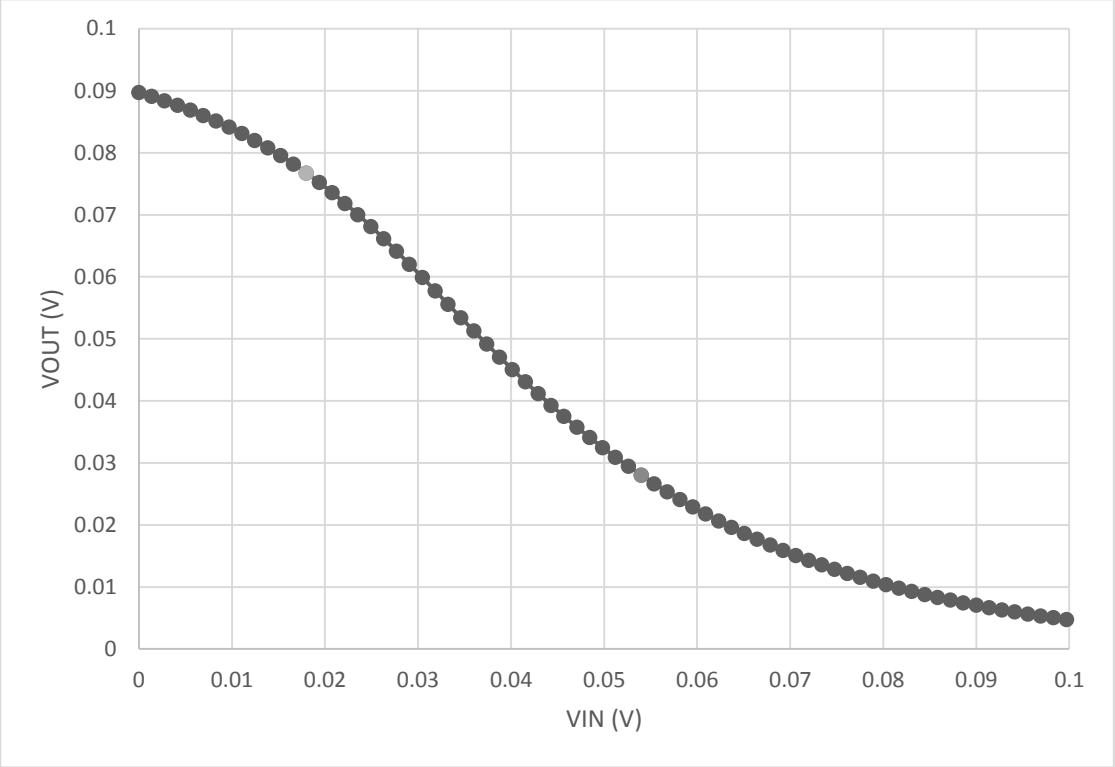


Figura 102. Curva de rechazo de ruido para una compuerta inversora.

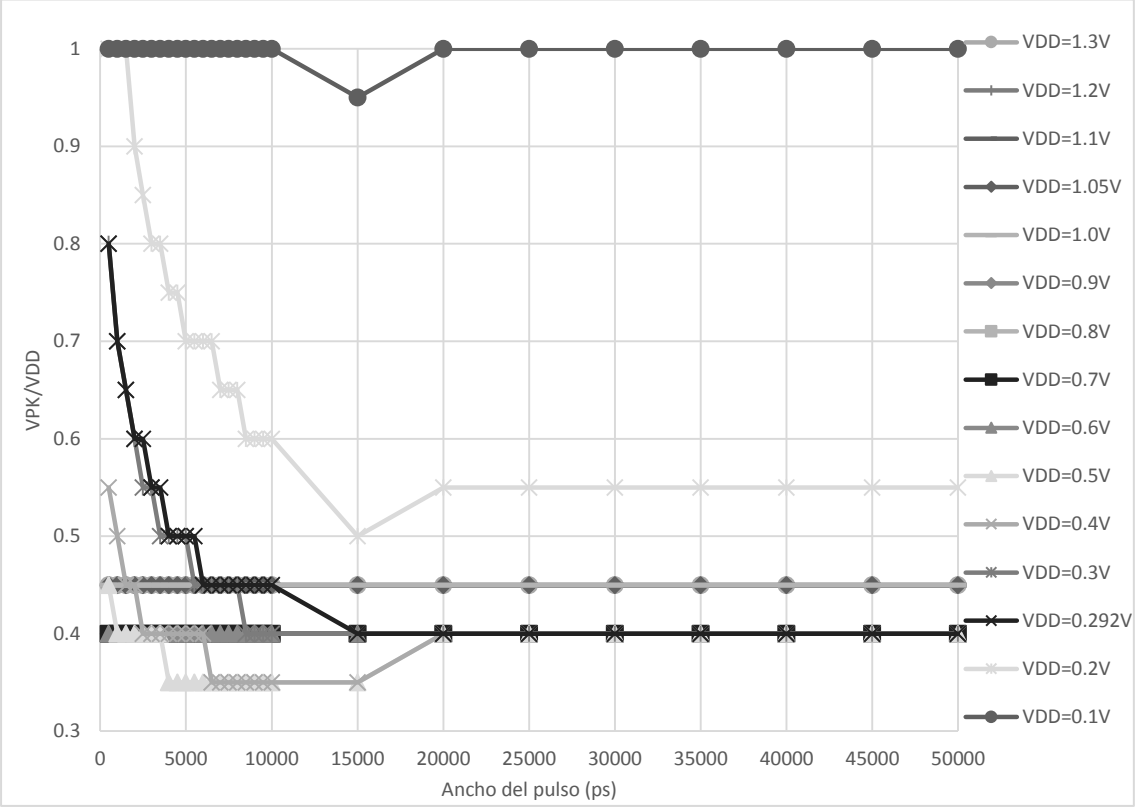


Figura 103.Curva de rechazo de ruido para un D Flip-flop de flanco negativo con tiempo de caída de 2ns en la señal de reloj.

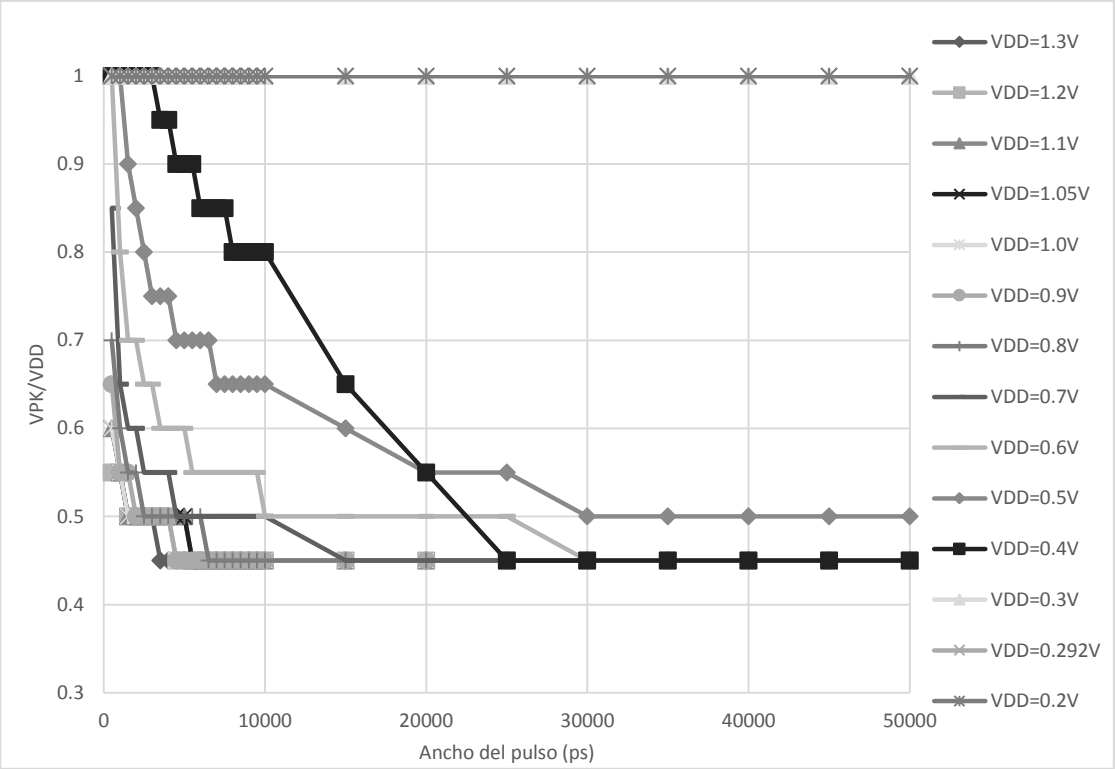


Figura 104. Grabación de datos fallida con un D Flip-flop de flanco negativo con tiempo de caída de 2ns en la señal de reloj. Se esperaba que la señal hiciera una transición de ‘1’ lógico a ‘0’ lógico.

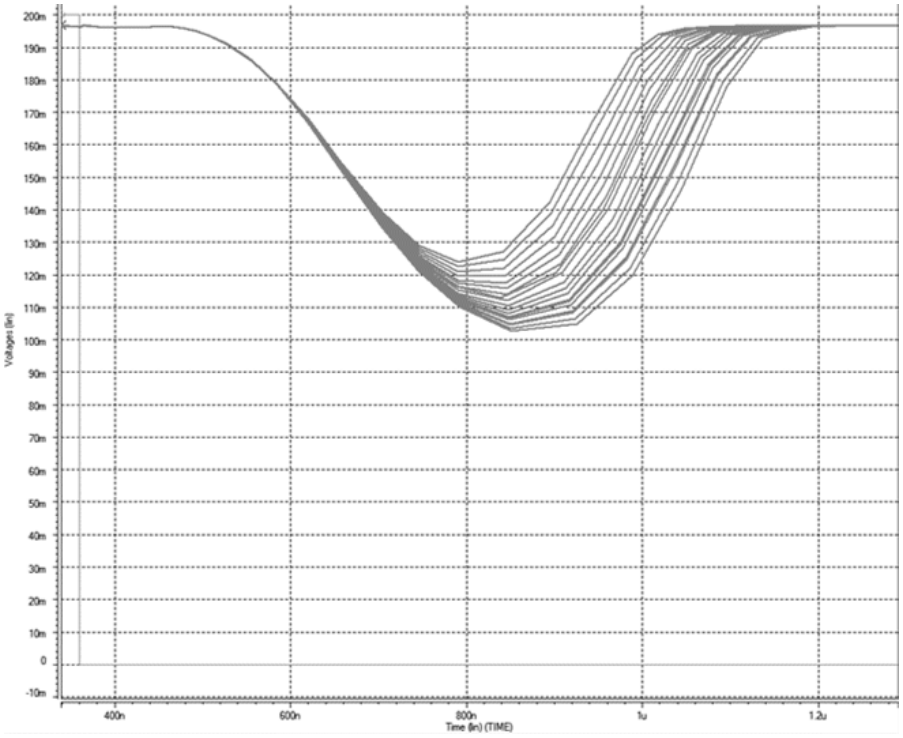


Figura 105. Grabación de datos exitosa con un D Flip-flop de flanco negativo con tiempo de caída de 100ns en la señal de reloj.

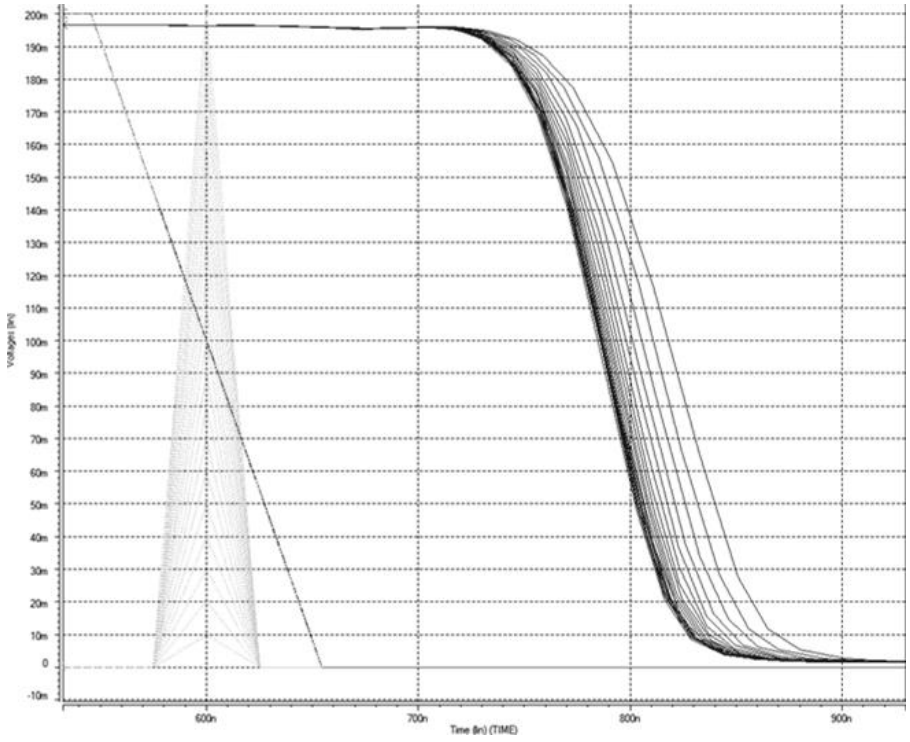


Figura 106. Ruido inyectado por encima del '1' lógico para distintas longitudes de *interconnect*.

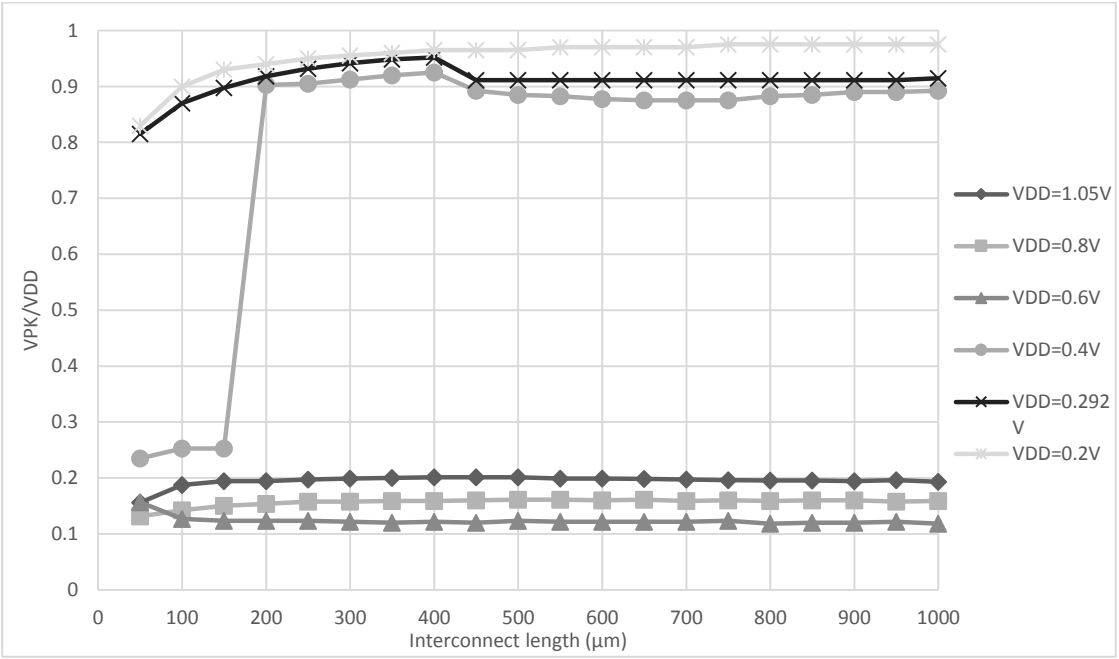
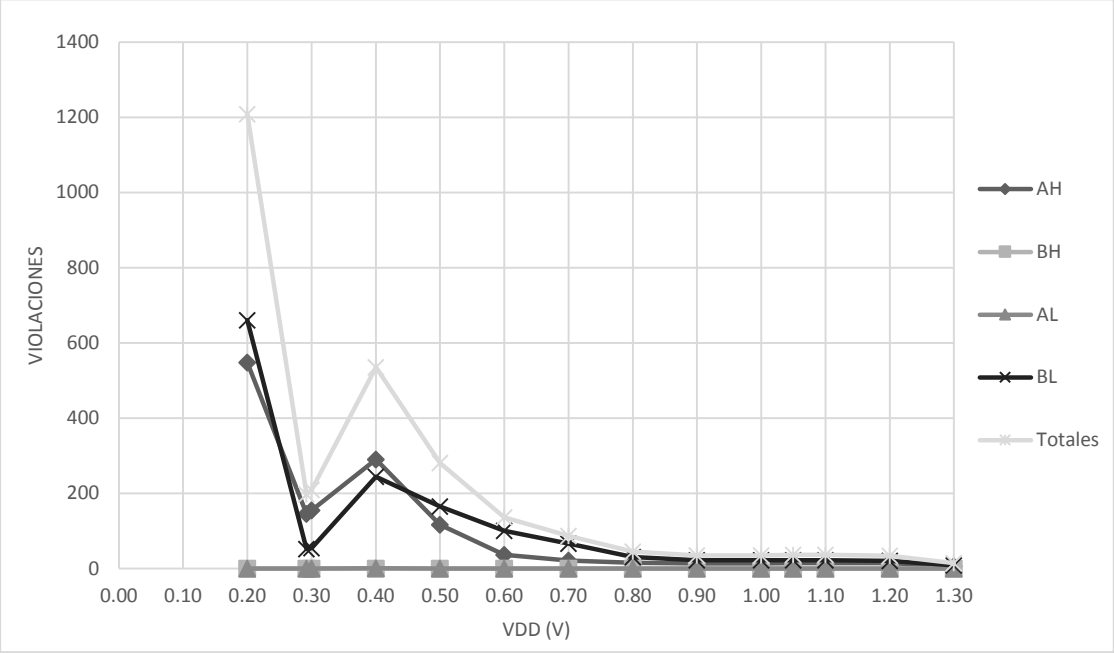


Figura 107. Violaciones en nodos de salida del Sumador/Restador de 32 *bits*.



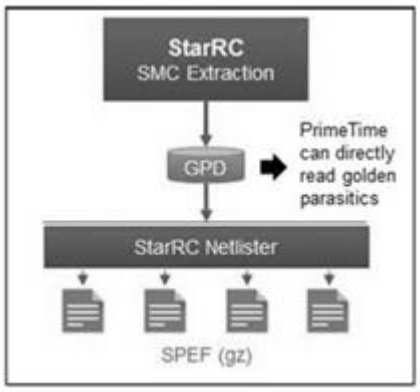
Cuadro 8.Cantidad de fallas de funcionamiento en las salidas del Sumador/Restador de 32 *bits*.

	Curva de transferencia DC para una not unitaria				Número de violaciones		
VDD	**AH**	**BH**	**AL**	**BL**	**BH**	**AL**	**Total de violaciones**
0.200	0.020	0.114	0.0389	0.020	0	0	0
0.292	0.0292	0.167	0.0609	0.0292	0	0	0
0.30	0.030	0.171	0.0637	0.030	0	0	0
0.40	0.040	0.229	0.0882	0.040	0	0	0
0.50	0.050	0.282	0.1150	0.050	0	0	0
0.60	0.060	0.343	0.1420	0.060	0	0	0
0.70	0.070	0.403	0.1700	0.070	0	0	0
0.80	0.080	0.467	0.2020	0.080	0	0	0
0.90	0.090	0.536	0.2350	0.090	0	0	0
1.00	0.100	0.610	0.2720	0.100	0	0	0
1.05	0.105	0.648	0.2910	0.105	0	0	0
1.10	0.110	0.678	0.3100	0.110	0	0	0
1.20	0.120	0.758	0.3480	0.120	0	0	0
1.30	0.130	0.832	0.3880	0.130	0	0	0

VII. ANÁLISIS DE RESULTADOS

La Figura 96 muestra el flujo requerido para analizar el *timing* de un circuito a nivel de transistor. Debe notarse que se intentó encontrar los pasos requeridos para hacer el mismo análisis a nivel de compuerta con la herramienta PrimeTime, pero no fue posible extraer los parásitos a nivel de compuerta con StarRC. El manual de usuario de StarRC no tenía documentación sobre cómo hacerlo, y se reducía simplemente a mencionar que sí es posible. En la página 4-8 de (Synopsys, StarRC User Guide and Command Reference, 2016) se presenta la Figura 108, que parece implicar que intermedio a la extracción de parásitos en formato SPEF se genera un archivo GPD (*Galaxy Parasitic Database*) que puede ser leído por PrimeTime. Sin embargo, esto sólo ocurre cuando la extracción se realiza a nivel de compuertas, y no se encontró una opción para habilitarlo. Esto ocurrió a pesar de que las compuertas utilizadas se instanciaron como celdas estándar.

Figura 108. Relación entre los archivos GPD y las herramientas StarRC y PrimeTime.



(Synopsys, StarRC User Guide and Command Reference, 2016)

Aun así, el intento de usar PrimeTime no fue completamente improductivo porque se encontró cómo se debía modificar el *runset* de StarRC para que tomara los nombres de los nodos del esquemático y no del diseño a nivel de silicio. También se encontró que las herramientas de Synopsys no son compatibles entre ellas si se usan sus opciones predefinidas. Por eso se analizó el manual de NanoTime y de StarRC para detectar qué opciones y variables se debían sincronizar. Esto se explicó previamente en la sección de Metodología. En el área encerrada en línea punteada de la Figura 96 se muestra el flujo que se usó para realizar el análisis de un circuito básico con PrimeTime. En proyectos futuros se debería remover el bloque de analizar los parásitos a nivel de transistor con un *script* de Python, y de la extracción de StarRC se debería obtener un archivo de parásitos a nivel de compuerta.

Se encontró la herramienta de NanoTime disponible en el sitio de Synopsys para el sitio de la Universidad del Valle de Guatemala. Esta herramienta puede analizar circuitos a nivel de transistor, a diferencia de PrimeTime, y también puede realizar un análisis de integridad de señales. Por la experiencia con PrimeTime, y porque NanoTime está mejor documentado que PrimeTime, la realización del *script* para el análisis fue

relativamente rápido. Sin embargo, sí hubo problemas al momento de abrir el archivo de parásitos SPEF generado por StarRC porque NanoTime reportaba errores en todas líneas. Además, hubo una confusión con los formatos SPF y DSPF, pues StarRC puede generar un archivo DSPF, a pesar de que su manual dice que genera un archivo SPF. Finalmente se abrió un archivo en formato DSPF guardado como "starrc_results.spf". Se encontraron algunos errores cuando se intentó abrir el archivo de parásitos DSPF, pero se concluyó que era debido a que las configuraciones del *runset* StarRC y del *script* de NanoTime aún no eran apropiadas. El error más común fue el de PARA-040, pues podía significar varios problemas. Este error indica que NanoTime descartó algunos capacitores parásitos porque tenían errores. La solución a esto fue forzar a StarRC a generar las instancias de transistores como subcircuitos con los siguientes comandos:

```
HN_NETLIST_SPICE_TYPE: n105 X
HN_NETLIST_SPICE_TYPE: p105 X
```

Además, se leyó el archivo DPF que se generó en la extracción de parásitos con el comando "read_device_parameters ideal_spice.dpf". Otra fuente de este error fue que la forma de reconocer la jerarquía de NanoTime no estaba sincronizada con la forma en que StarRC la identificó. Para esto se usó la opción "HIERARCHICAL_SEPARATOR: ." en StarRC y "set hierarchy_separator .", donde el "." es el separador de la jerarquía del diseño, en NanoTime.

Con los cuadros 6 y 7 se validó que tanto la extracción de parásitos como el *script* de NanoTime dieran resultados correctos. Se comparó el promedio de los resultados de propagación de CLK a Q para el D Flip-flop con la caracterización proveída por (Synopsys, Digital Standard Cell Library, 2012). Este promedio fue entre el tiempo de propagación cuando la señal en el puerto Q tenía un flanco de subida y cuando tenía un flanco de bajada. Por la existencia de los parásitos se esperaba que el promedio fuera mayor pero cercano a los 130ps reportados en (Synopsys, Digital Standard Cell Library, 2012). Se aseguró que las condiciones de operación fueran exactamente las mismas que las condiciones en que caracterizaron la *standard-cell* del D Flip-flop que se usó para que sí fuera válido.

Cuando el circuito estaba compuesto solamente por lógica combinacional, se esperaba que la señal de salida arribara de inmediato, como en la teoría cuando no se considera ningún retardo en las señales. Como se puede observar en los cuadros 4 y 5, el peor *slack* era negativo, lo cual indicó que la señal llegaba después de lo requerido. Los retardos mínimos se tomaron sobre cuánto antes llegó la señal desde un flanco positivo de un reloj MCLK que se generó con período de 20µs, hasta su siguiente flanco positivo. Por esta razón el *slack* es bastante positivo, porque los retardos de la lógica combinacional eran casi despreciables a comparación del período del reloj MCLK. Debe notarse que este reloj no se conectó a ningún puerto del circuito y sólo se usó su período como referencia.

Como se puede observar en las Figuras 97 a 99, al reducir el voltaje de alimentación desde 1.3V, que es el potencial máximo de operación permitido para la tecnología acorde a Synopsys (Synopsys, Digital Standard Cell Library, 2012), hasta 0.292V, que es el voltaje de umbral de la tecnología según (Cao, 2011), la curva de transferencia DC de la compuerta inversora se corre hacia la izquierda. Los puntos resaltados en las Figuras 97 a 101 son los puntos donde la pendiente es igual a -1. Por lo tanto, ya que la curva se mueve hacia la izquierda al reducir el voltaje de alimentación, la tolerancia al ruido en la entrada disminuye. Esto afecta principalmente al '0' lógico, pues el rango de voltajes que la compuerta toma como un '0' lógico en su entrada disminuye considerablemente. En las Figuras 100 y 101 se muestra la curva de transferencia DC para voltajes de alimentación de 0.2V y 0.1V, respectivamente. Se debe resaltar que en la Figura 101, la compuerta inversora no hizo una transición completa pues su salida no llegó al voltaje de alimentación. Por lo tanto, el voltaje de alimentación mínimo que se debería usar es 0.2V.

En las Figuras 102 y 103 se muestra la curva de rechazo de ruido para una compuerta inversora y un D Flip-flop de flanco negativo, respectivamente. Como puede notarse, conforme se reduce el voltaje de alimentación en ambos casos, la curva se corre hacia arriba. Esto significa que tanto la compuerta inversora como el D Flip-flop se vuelven más tolerantes al ruido. Para un ruido de duración dada por el eje horizontal de las Figuras 102 y 103, el ruido debe tener un pico de voltaje más alto para causar una salida errónea en la compuerta inversora o una grabación incorrecta de un dato en el D Flip-flop. Es bastante importante mencionar que para el voltaje de alimentación de 0.2V, el D Flip-flop no grabó datos en ningún caso. Esto causó dudas sobre si el D Flip-flop funcionaba en subumbral. Con un tiempo de transición de 2ns, el comportamiento de la salida del D Flip-flop se muestra en la Figura 104. Es posible que un nodo interno del D Flip-flop no hiciera una transición completa y que la lógica CMOS lo restaurara a un '1' lógico. Esto evitaría que la transición de '1' a '0' lógico ocurriera. Se comprobó que el D Flip-flop sí funcionaba en subumbral al aumentar el tiempo de transición de la señal de reloj a 100ns, como se muestra en la Figura 105.

Se buscó comprobar si el ruido inyectado se comportaba de la misma forma que en el estudio de (Nanua & Blaauw, 2007) para la tecnología de 65nm, donde los agresores inyectaban más ruido en las víctimas en operación subumbral que en operación superumbral. Como se muestra en la Figura 106, esta tendencia se mantiene para la tecnología de 32nm que se estudió, pues los voltajes de operación subumbral presentaban curvas de inyección de ruido en niveles más altos que los voltajes de operación superumbral. Al unir todos los resultados existió la duda de si la operación subumbral era más robusta que la operación superumbral. Aunque se inyectaba más ruido en operación subumbral acorde a la Figura 106, se toleraba más ruido acorde a las Figuras 102 y 103. Para comprobar qué efecto era más fuerte, se estudió el Sumador/Restador de 32 *bits*. Como se muestra en la Figura 107, al reducir el voltaje de superumbral a subumbral ocurrieron más violaciones de estado en nodos. Esto causaría una mayor cantidad de *glitches* en la lógica combinacional en subumbral. Sin embargo, como se muestra en el cuadro 8, las salidas del Sumador/Restador no tenían ruido de tal magnitud que se grabaría como un dato en un D Flip-flop, por lo que no habría fallas de funcionamiento.

VIII. CONCLUSIONES

1. Se desarrolló un procedimiento que permitió cuantificar las violaciones debido al *crosstalk* y las violaciones de *timing* con la herramienta NanoTime de Synopsys.
2. Se instaló y utilizó efectivamente la herramienta de NanoTime para análisis de *timing*.
3. Se realizó y validó el *script* de NanoTime comparando los resultados con HSPICE y con lo reportado por Synopsys para sus celdas estándar.
4. Se logró detectar errores de *timing* como resultado de que algunas señales tenían un *slack* negativo.
5. El escalamiento de los transistores no es lineal al reducir el voltaje de alimentación, pues el NMOS se vuelve más fuerte que el PMOS.
6. Conforme se reduce el voltaje de alimentación se espera que haya menos tolerancia al ruido cuando la entrada es un '0' lógico.
7. El voltaje pico normalizado del ruido inyectado incrementa conforme se reduce el voltaje de operación de condiciones superumbral a subumbral, y aumenta con interconexiones más largas.
8. El D Flip-flop de flanco negativo no graba datos con cualquier reloj cuando está operando en subumbral.
9. Para las 3208 *nets* del circuito Sumador/Restador de 32 *bits* se encontraron más violaciones en nodos en operación subumbral que en operación superumbral.
10. No se encontraron posibles fallos de funcionamiento en las salidas del Sumador/Restador de 32 *bits*.

IX. RECOMENDACIONES

1. Se recomienda leer los manuales de las herramientas que se piense usar para determinar los requisitos que tienen. Esto tiene un impacto en la forma en que se configura el *runset* o *script* de las herramientas que generan los archivos que se usan posteriormente, como fue el caso con StarRC y NanoTime.
2. Se recomienda comprender el ambiente de CentOS para que la instalación de las herramientas sea más sencilla. Se deben localizar varios archivos y carpetas específicas que son de importancia, como el archivo ".bashrc" para definir variables de entorno y la carpeta "synopsys" que contiene las instalaciones de las herramientas.
3. Investigar por qué ocurre el *gate skewing* conforme se reduce el voltaje de operación y evaluar formas de compensarlo para mejorar la operación en *subthreshold.*
4. Investigar de dónde surge el comportamiento extraño para el voltaje de 0.4V.

X. BIBLIOGRAFÍA

Cao, Y. (2011). Predictive Technology Model of Conventional CMOS Devices. En Y. Cao, *Predictive Technology Model for Robust Nanoelectronic Design* (págs. 7-22). Springer Science+Business Media.

de los Santos, J. A. (2014). <<*Diseño de un sumador/restador completo de 32 bits con tecnología CMOS en un proceso de 28 nanómetros usando aplicaciones de diseño de la empresa Synopsys* >>. Tesis Universidad del Valle de Guatemala.

Fuketa, H., Takahashi, R., Takamiya, M., Nomura, M., Shinohara, H., & Sakurai, T. (2012). Increase of Crosstalk Noise Due to Imbalanced Threshold Voltage between NMOS and PMOS in Sub-Threshold Logic Circuits. *IEEE*, 4.

Gayathri. (8 de diciembre de 2012). *What is crosstalk analysis/signal integrity analysis in static timing analysis.* Obtenido de Technology@Tdzire: http://tech.tdzire.com/what-is-cross-talk-analysissignal-integrity-analysis-in-static-timing-analysis/

Gayathri. (20 de noviembre de 2012). *Why Coupling Capacitance Increases When CMOS Technology Shrinks*. Obtenido de Technology@Tdzire: http://tech.tdzire.com/why-coupling-capacitance-increases-when-cmos-technology-shrinks/

Harris, D. M., & Weste, N. H. (2011). *Cmos VLSI Design a Circuits and Systems Perspective.* United States of America: Addison-Wesley.

Harris, D. M., & Weste, N. H. (2011). *CMOS VLSI Design: A Circuits and Systems Perspective.* United States of America: Addison-Wesley.

Kaeslin, H. (2008). *Digital Integrated Circuit Design From VLSI Architectures to CMOS Fabrication.* United States of America: Cambridge University Press, New York.

Kamakoti, V., & Balachandran, S. (9 de octubre de 2013). *CAD for VLSI Design - II*. Obtenido de SlideShare: http://www.slideshare.net/chenna_kesava/nptel-cad206-capcitances

Nanua, M., & Blaauw, D. (2007). Investigating Crosstalk in Sub-Threshold Circuits. *IEEE Computer Society*, 1-6.

Roy, K. (2009). *HSPICE, Nanosim, and NanoTime Simulation Introduction.* West Lafayette, Indiana, USA: Purdue University.

Shomalnasab, G., Heys, H. M., & Zhang, L. (s.f.). *Interconnect Capacitive Modeling in Submicron and Nano Technologies.* St. John´s, Canada: Memorial University of Newfoundland, Electrical and Computer Engineering Department.

Synopsys. (3 de septiembre de 2009). *Handling of Macro-Model Parasitics in NanoTime.* Obtenido de Solvnet.synopsys.com: https://solvnet.synopsys.com/retrieve/028079.html

Synopsys. (15 de diciembre de 2010). *Setting Up the Back-Annotation Flow in NanoTime Using StarRC Parasitics Netlist.* Obtenido de Solvnet.synopsys.com: solvnet.synopsys.com/retrieve/031751.html

Synopsys. (2012). *Digital Standard Cell Library.* Synopsys Armenia Educational Department.

Synopsys. (junio de 2016). *Custom Compiler Help.* Obtenido de Solvnet.synopsys.com: solvnet.synopsys.com/dow_retrieve/latest/custom_compiler/custom_compiler_olh/index.htm

Synopsys. (junio de 2016). *NanoTime.* Obtenido de Solvnet.synopsys.com: https://solvnet.synopsys.com/dow_retrieve/latest/ni/nanotime.html#NanoTime%20and%20NanoTime%20Ultra

Synopsys. (junio de 2016). *PrimeTime Online Help.* Obtenido de Solvnet.synopsys: https://solvnet.synopsys.com/dow_retrieve/latest/ptolh/Default.htm

Synopsys. (junio de 2016). *StarRC User Guide and Command Reference.* Obtenido de Solvnet.synopsys.com: https://solvnet.synopsys.com/dow_retrieve/latest/stug/ni/stug.pdf

Synopsys. (septiembre de 2016). *Synopsys Installation Guide.* Obtenido de Solvnet.synopsys.com: https://solvnet.synopsys.com/Install/installation_guide.jsp?id=166&releasedate=2016-09-06

Synopsys. (2016). *Worldwide University Program*. Obtenido de solvnet.synopsys.com: http://www.synopsys.com/community/universityprogram/Pages/default.aspx

XI. ANEXOS

A. Archivo ".bashrc" para creación de variables de entorno en CentOS

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
     . /etc/bashrc
fi
# User specific aliases and functions
LM_LICENSE_FILE=27020@192.168.6.124; export LM_LICENSE_FILE
PATH=/usr/synopsys/installer:$PATH
export PATH
PATH=/usr/synopsys/CustomCompiler/bin:$PATH
export PATH
export SAED32_28_PDK=/usr/synopsys/iPDK/SAED32_28_iPDK/
source /usr/synopsys/PyCell/quickstart/bashrc
PATH=/usr/synopsys/HSPICE/I-2013.12-SP2-1/hspice/bin/:$PATH
export PATH
export ICV_HOME_DIR=/usr/synopsys/IC_Validator
PATH=/usr/synopsys/IC_Validator/bin/AMD.64:$PATH
export PATH
PATH=/usr/synopsys/CustomExplorer/bin/:$PATH
export PATH
PATH=/usr/synopsys/StarRC_2016/bin:$PATH
export PATH
PATH=/usr/synopsys/PrimeTime/bin/:$PATH
export PATH
PATH=/usr/synopsys/NanoTime/bin/:$PATH
export PATH
export SYNOPSYS_CUSTOM_SITE=/usr/synopsys/NanoTime/admin/setup
PATH=/usr/synopsys/Hercules/bin/amd64/:$PATH
export PATH
export SAED_PDK_90=/usr/synopsys/iPDK/SAED90_Ipdk
```

B. *Runset* de StarRC para compatibilidad con NanoTime

```
BLOCK: XO_2_0
TCAD_GRD_FILE: /usr/synopsys/iPDK/SAED32_28_iPDK/starrc/nominal/saed32nm_1p9m_nominal.nxtgrd
ICV_RUNSET_REPORT_FILE: ./pex_runset_report
COUPLE_TO_GROUND: NO
COUPLING_MULTIPLIER: 1
MILKYWAY_EXTRACT_VIEW: YES
CASE_SENSITIVE: YES
NETLIST_FORMAT: SPF
NETLIST_NODE_SECTION: YES
NETLIST_CONNECT_SECTION: YES
NETLIST_SUBCKT: YES
NETLIST_PASSIVE_PARAMS: YES
NETLIST_NODENAME_NETNAME: NO
NETLIST_DELIMITER: :
NETLIST_IDEAL_SPICE_FILE: ideal_spice.dpf
INTRANET_CAPS: NO
REDUCTION: NO
XREF: YES
EXTRACT_VIA_CAPS: NO
IGNORE_CAPACITANCE: NONE
KEEP_VIA_NODES: NO
MAGNIFY_DEVICE_PARAMS: NO
METAL_FILL_POLYGON_HANDLING: IGNORE
MODE: 200
MOS_GATE_DELTA_RESISTANCE: NO
REMOVE_DANGLING_NETS: NO
REMOVE_FLOATING_NETS: NO
TRANSLATE_RETAIN_BULK_LAYERS: YES
SKIP_CELLS: !*
HIERARCHICAL_SEPARATOR: .
EXTRA_GEOMETRY_INFO: NODE
HN_NETLIST_SPICE_TYPE: n105 X
HN_NETLIST_SPICE_TYPE: p105 X
OA_LIB_NAME:
OA_VIEW_NAME: starrc
OA_MARKER_SIZE: 0.1
OA_REMOVE_SPICECARD_PREFIX: NO
OA_DEVICE_MAPPING_FILE: /usr/synopsys/iPDK/SAED32_28_iPDK/saed32nm_1p9m_device.map
```

C. *Script* de Python

```
def gate_number(string):
    for i in xrange(len(string)-1, 0, -1):
        if not string[i].isdigit():
            return string[i+1:]
spef_input_name = 'starrc_results.spef'; spef_input_file = open(spef_input_name, 'r');
flag = False; name_map = []; lines = [];
for line in spef_input_file:
    if not '//' in line:
        if '*NAME_MAP' in line:
            flag = True
        elif flag and line != '\n':
            if '*PORTS' in line:
                flag = False
            elif 'X' in line:
                    name_map.append(line)
    lines.append(line)
names = {}
for name in name_map:
    s = name.split(' '); names[s[0]] = s[1].split('|')[0];
spef_output_name = 'starrc_results_modified.spef';
spef_output_file = open(spef_output_name, 'w'); flag = False
for line in lines:
    writeable = line
    if not flag and '*CONN' in line:
        flag = True
    if flag:
        for key in names.keys():
            if key in writeable:
                key_index = writeable.index(key);
                end_index = writeable.find(' ', key_index)
                gate = gate_number(names[key])
                if 'GATE' in writeable[key_index:end_index]:
                    s = 'u'+str(gate)+':A'
                    writeable = writeable.replace(writeable[key_index:end_index], s)
                elif 'DRN' in writeable[key_index:end_index]:
                    s = 'u'+str(gate)+':Y'
                    writeable = writeable.replace(writeable[key_index:end_index], s)
    spef_output_file.write(writeable)
spef_input_file.close(); spef_output_file.close();
```

D. Script de PrimeTime

```
# Read Design Data

read_lib saed32rvt_tt0p78v25c.lib
set link_path "* saed32rvt_tt0p78v25c"
read_verilog RingOsc3.v
link_design RingOsc3
set si_enable_analysis true
# Read parasitics
read_parasitics -keep_capacitive_coupling starrc_results_modified.spef

#Constraint report to be able to set the input/output delay
report_constraint

# Set timing constraints

# clock definition
create_clock -name mclk1 -period 50.0 [get_ports CLK]

# input delays
#set_input_delay  0.0 [get_ports {}] -clock mclk1
#set_input_delay  0.0 [get_ports {A2}] -clock mclk2

# input drivers
set_driving_cell -lib_cell INVX0_RVT -pin o [get_ports  {net17}] -input_transition_fall
80.0 -input_transition_rise 80.0
#set_driving_cell -lib_cell INVX0_RVT -pin o [get_ports  {A2}] -input_transition_fall
80.0 -input_transition_rise 80.0

# output delays
set_output_delay  0.0 [get_ports {Q}] -clock mclk1
#set_output_delay  0.0 [get_ports {OUT2}] -clock mclk2

# output loads
set_load -pin_load  4.0 [get_ports {Q}]
#set_load -pin_load  4.0 [get_ports {OUT2}]

#Path timing report, reports the path with the worst setup slack.
report_timing
```

E. Script de NanoTime

```
#************************************************************
#* NETLIST PHASE
#************************************************************
set search_path {.}
set link_path {*}
set link_case ""
set hierarchy_separator .
set parasitics_xref_layout_instance_prefix ld_
set parasitics_fingered_device_chars \@
set parasitics_enable_drain_source_swap true
set link_transistor_drain_pin_name DRN
set link_transistor_gate_pin_name GATE
set link_transistor_source_pin_name SRC
set link_transistor_bulk_pin_name BULK
set link_enable_wrapper_subckt_parasitics false
set parasitics_accept_node_name_net_name true
set parasitics_enable_annotation_to_embedded_rc true
set parasitics_enable_drain_source_swap true
set parasitics_ground_incomplete_coupling_cap true
set parasitics_suppress_dpf_inheritance multi
set si_enable_noise_analysis true
set si_enable_analysis true
set parasitics_read_variation true
set topo_auto_search_class {mux inverter flip-flop}
# crosstalk parasitic options
#set parasitics_coupling_cap_variation_max 0.0
#set parasitics_coupling_cap_variation_min 0.0
#set si_enable_aggressor_logic_pessimism_reduction true
# crosstalk parasitic filtering variables
# si_filter_total_aggr_xcap 0.0
# si_filter_total_aggr_xcap_to_gcap_ratio 0.0
# si_aggressive_filtering_total false
# si_filter_per_aggr_xcap 0.0
# si_filter_per_aggr_xcap_to_gcap_ratio 0.0
# si_filter_per_aggr_to_average_aggr_xcap_ratio 0.0
# si_aggressive_filtering_per_aggr false
#set sim_transistor_wrapper_subckts # solvnet article 028079
set oc_global_voltage 1.0
```

```
  register_netlist -format spice {cadena_up.sp tech.sp}
  link_design -keep_capacitive_coupling CadenaNots
  #*************************************************************
  #* CLOCK DEFINITION AND TOPOLOGY RECOGNITION PHASE
  #*************************************************************
  set_supply_net VDD
  set_voltage 1.05 VDD
  set_supply_net -gnd VSS
  set_port_direction -input [get_ports A*]
  set_port_direction -output [get_ports {OUT* QN*}]
  create_clock -period 20 -name MCLK [get_ports CLK*]
  set_input_transition 0.1 -clock MCLK [get_ports CLK*]
  read_device_parameters CadenaNots_ideal.dpf
  set topo_find_clock_driven_data_inputs true
  match_topology -structure_types {flip_flop inverter}
  check_topology
  #*************************************************************
  #* TIMING CONSTRAINT SPECIFICATION PHASE
  #*************************************************************
  #read_spice_model ... # solvnet article 028079
  #set_technology ... # solvnet article 028079
  set_input_delay -clock MCLK 10 [get_ports A*]
  set_output_delay -clock MCLK 10 [get_ports {OUT* QN*}]
  # logic constraints...
  set parasitics_enable_mapping_unresolved_pins true
  # do not use the -complete_with or complete_net_parasitics options (solvnet article
028079)
  read_parasitics -keep_capacitive_coupling starrc_results.spf
  # -complete_with zero in accordance with solvnet article 028079
  check_design -complete_with zero
  #*************************************************************
  #* PATH TRACING (TIMING ANALYSIS) PHASE
  #*************************************************************
  trace_paths
  #*************************************************************
  #* ANALYSIS REPORTING PHASE
  #*************************************************************
  report_paths -max
  report_paths -min
  #*************************************************************
```

F. HSPICE ideal de un circuito prueba (cadena de compuertas inversoras)

```
*  Generated for: HSPICE
*  Design library name: andcas_pruebas
*  Design cell name: CadenaNots_tb
*  Design view name: schematic
.option search='/usr/synopsys/iPDK/SAED32_28_iPDK/hspice'
.option PARHIER = LOCAL
.option RUNLVL = 5
.option ARTIST=2 PSF=2
.temp 25
.lib 'saed32nm.lib' TT
*Custom Compiler Version L-2016.06-5
********************************************************************************
* Library          : SAED_PDK_32_28
* Cell             : INVX0_RVT
* View             : schematic
* View Search List : auCdl schematic symbol
* View Stop List   : auCdl
********************************************************************************
.subckt INVX0_RVT A VDD VSS Y
*.PININFO A:I VDD:B VSS:B Y:O
XMM2 Y A VDD VDD p105 w=0.52u l=0.03u nf=1.0 m=1
XMM3 Y A VSS VSS n105 w=0.27u l=0.03u nf=1.0 m=1
.ends INVX0_RVT
********************************************************************************
* Library          : SAED_PDK_32_28
* Cell             : DFFX2_RVT
* View             : schematic
* View Search List : auCdl schematic symbol
* View Stop List   : auCdl
********************************************************************************
.subckt DFFX2_RVT CLK D Q QN VDD VSS
*.PININFO CLK:I D:I Q:O QN:O VDD:B VSS:B
XMM36 D3 D2 VDD VDD p105 w=0.7u l=0.03u nf=1.0 m=1
XMM48 D2 D3 FB1 VDD p105 w=0.3u l=0.03u nf=1.0 m=1
XMM34 FB4 CLKP VDD VDD p105 w=0.3u l=0.03u nf=1.0 m=1
XMM46 D4 CLKN D3 VDD p105 w=0.4u l=0.03u nf=1.0 m=1
XMM32 D1 D VDD VDD p105 w=0.45u l=0.03u nf=1.0 m=1
XMM44 Q D5 VDD VDD p105 w=1.6u l=0.03u nf=2.0 m=1
XMM30 D1 CLKP D2 VDD p105 w=0.4u l=0.03u nf=1.0 m=1
```

```
XMM42 QN D4 VDD VDD p105 w=1.6u l=0.03u nf=2.0 m=1
XMM28 CLKP CLKN VDD VDD p105 w=0.74u l=0.03u nf=1.0 m=1
XMM40 D4 D5 FB4 VDD p105 w=0.3u l=0.03u nf=1.0 m=1
XMM26 CLKN CLK VDD VDD p105 w=0.74u l=0.03u nf=1.0 m=1
XMM38 D5 D4 VDD VDD p105 w=0.7u l=0.03u nf=1.0 m=1
XMM0 FB1 CLKN VDD VDD p105 w=0.3u l=0.03u nf=1.0 m=1
XMM37 QN D4 VSS VSS n105 w=0.84u l=0.03u nf=2.0 m=1
XMM49 D4 D5 FB5 VSS n105 w=0.3u l=0.03u nf=1.0 m=1
XMM35 FB5 CLKN VSS VSS n105 w=0.3u l=0.03u nf=1.0 m=1
XMM47 D2 D3 FB2 VSS n105 w=0.3u l=0.03u nf=1.0 m=1
XMM33 D5 D4 VSS VSS n105 w=0.35u l=0.03u nf=1.0 m=1
XMM45 D4 CLKP D3 VSS n105 w=0.3u l=0.03u nf=1.0 m=1
XMM31 D1 D VSS VSS n105 w=0.25u l=0.03u nf=1.0 m=1
XMM43 D1 CLKN D2 VSS n105 w=0.3u l=0.03u nf=1.0 m=1
XMM29 D3 D2 VSS VSS n105 w=0.35u l=0.03u nf=1.0 m=1
XMM41 FB2 CLKP VSS VSS n105 w=0.3u l=0.03u nf=1.0 m=1
XMM27 CLKP CLKN VSS VSS n105 w=0.4u l=0.03u nf=1.0 m=1
XMM39 Q D5 VSS VSS n105 w=0.84u l=0.03u nf=2.0 m=1
XMM1 CLKN CLK VSS VSS n105 w=0.4u l=0.03u nf=1.0 m=1
.ends DFFX2_RVT
********************************************************************************
* Library          : andcas_pruebas
* Cell             : CadenaNots
* View             : schematic
* View Search List : auCdl schematic symbol
* View Stop List   : auCdl
********************************************************************************
.subckt CadenaNots A1 A2 CLK1 CLK2 OUT1 OUT2 QN1 QN2 VDD VSS
*.PININFO A1:I A2:I CLK1:I CLK2:I OUT1:O OUT2:O QN1:O QN2:O VDD:B VSS:B
XI6 net20 VDD VSS net47 INVX0_RVT
XI7 net24 VDD VSS net21 INVX0_RVT
XI8 net28 VDD VSS net25 INVX0_RVT
XI9 A1 VDD VSS net29 INVX0_RVT
XI5 net21 VDD VSS net48 INVX0_RVT
XI4 net25 VDD VSS net20 INVX0_RVT
XI1 net29 VDD VSS net24 INVX0_RVT
XI0 A2 VDD VSS net28 INVX0_RVT
XI11 CLK2 net48 OUT2 QN2 VDD VSS DFFX2_RVT
XI10 CLK1 net47 OUT1 QN1 VDD VSS DFFX2_RVT
.ends CadenaNots
.end
```