Diseño de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC: Simulación en *PrimeSim* y análisis de errores.

Israel Antonio Arévalo González

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC: Simulación en *PrimeSim* y análisis de errores.

Trabajo de graduación presentado por Israel Antonio Arévalo González
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

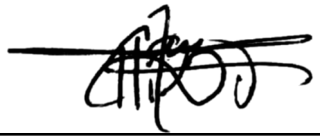
Diseño de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC: Simulación en *PrimeSim* y análisis de errores.

Trabajo de graduación presentado por Israel Antonio Arévalo González
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2023

Vo.Bo.:



(f) _____

MSc. Carlos Esquit

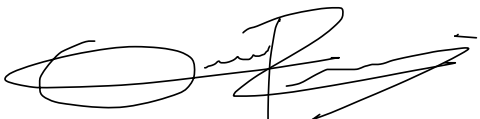
Tribunal Examinador:

(f) _____

MSc. Carlos Esquit

(f) _____

Ing. Jonathan de los Santos



(f) _____

Ing. Juan Ricardo Girón

Fecha de aprobación: Guatemala, 5 de enero de 2023.

Prefacio

Antes que nada, quisiera dedicar este trabajo a mi familia, quienes siempre me han apoyado a lo largo de estos 5 años y han sido un gran ejemplo para mí. Además, darle las gracias a Dios por permitirme llegar a este punto de mi vida.

Este trabajo forma parte de una línea de investigación dentro de la Universidad del Valle de Guatemala que tiene la finalidad de impulsar el desarrollo del campo de nanoelectrónica en la región. Al tratarse de un proyecto intergeneracional, este no podría ser realizado sin los avances y resultados obtenidos en las investigaciones de las promociones pasadas. Me gustaría agradecer a la universidad por darme la oportunidad de pertenecer a una comunidad educativa que busca la excelencia. Al Ms.C Carlos Esquit y al Ing. Jonathan de los Santos, por su apoyo incondicional y asesoría brindada para la realización de este trabajo y por su deseo continuo deseo de realizar investigaciones de primer nivel. Finalmente, le doy gracias a mis profesores por tener la paciencia y la disposición de trasladarme sus conocimientos.

Índice

Lista de figuras

Lista de cuadros

Resumen

En la Universidad del Valle de Guatemala se encuentra activa una línea de investigación en nanoelectrónica en la que se está diseñando un circuito que muestra una secuencia de caracteres ASCII. Actualmente, el diseño del circuito integrado se encuentra casi completo. Sin embargo, antes de enviar a fabricar el chip, es necesario realizar simulaciones para verificar que este se comporte correctamente. Este trabajo muestra una serie de simulaciones usando la herramienta de *PrimeSim* HSPICE, que pueden ser utilizadas para verificar el correcto funcionamiento del chip; de igual manera, pretende explorar otro tipo de simulaciones que pueden ser utilizadas o ser tomadas como referencia para verificar futuros diseños dentro de la línea de investigación que lleguen a ser más complejos.

Abstract

At the Universidad del Valle de Guatemala there is an active line of research in nanoelectronics in which a circuit that displays a sequence of ASCII characters is being designed. Currently, the design of the integrated circuit is almost complete. However, before sending the chip to the foundry, it is necessary to carry out simulations to verify that it behaves correctly. This work shows a series of simulations using the *PrimeSim* HSPICE tool, which can be used to verify the correct operation of the chip; likewise, it intends to explore other types of simulations that can be used or taken as a reference to verify future designs within the line of research that become more complex.

CAPÍTULO 1

Introducción

En el año 1947 se marcó un punto de inflexión en el desarrollo tecnológico debido a la invención del transistor, esto dio inicio a lo que en la actualidad se conoce como era digital. Han pasado más de 70 años desde este evento y el transistor se ha vuelto parte fundamental de la vida cotidiana de las personas, puesto que la mayoría de los dispositivos que se utilizan hoy día, poseen transistores para ejecutar su función.

A medida que ha pasado el tiempo, se han logrado madurar los procesos de fabricación de los circuitos integrados; lo que a su vez ha derivado en la posibilidad de reducir constantemente el tamaño de los transistores llegando a escalas nanométricas. Debido a la complejidad de los diseños modernos, se utiliza VLSI para cumplir con los requerimientos de los procesos de diseño modernos. A pesar de ser una línea de investigación que lleva existiendo más de medio siglo, no es muy trabajada en la región latinoamericana. La Universidad del Valle de Guatemala ha identificado este aspecto desde el 2019, ha impulsado la investigación en nanoelectrónica, de tal forma que llegado el 2022 se cuenta con un diseño casi terminado del primer nanochip: diseñado en el país.

Debido a que la fabricación de un nanochip es un proceso que utiliza muchos recursos y puede tomar meses para completarse, resulta necesaria la simulación del diseño para verificar que este se comporte de la forma en que se espera, con el objetivo de identificar errores de diseño y poder corregirlos antes de que sea enviado a fabricar.

CAPÍTULO 2

Antecedentes

Este trabajo forma parte de una línea de investigación en la que han participado estudiantes de distintas promociones de ingeniería electrónica de la Universidad del Valle de Guatemala. La posibilidad de realizar investigaciones en el área de nanoelectrónica comenzó con el convenio realizado en el 2014 entre la universidad y la empresa desarrolladora de software Synopsys, quien nos ha permitido usar sus herramientas de software para diseño electrónico. Gracias a esto y al trabajo de investigación del ingeniero Jonathan de los Santos [1], se han podido adaptar estos programas al equipo de cómputo de la universidad para que los estudiantes de último año de ingeniería electrónica puedan aprender a profundidad sobre nanoelectrónica utilizando herramientas que son de vanguardia en el sector de los semiconductores.

El tener acceso a estas herramientas, suscitó la idea de realizar el primer nanochip del país y la región centroamericana; con lo cual se logró llegar a un acuerdo con Interuniversity Microelectronics Centre (IMEC) para que apoyara a la universidad con la fabricación de este circuito integrado. Debido a que IMEC tiene convenio de fabricación con Taiwan Semiconductor Manufacturing Company (TSMC), se obtuvo acceso a sus librerías de diseño que ha permitido el desarrollo del *layout:* del chip. Gracias a los convenios expuestos anteriormente, se logró abrir la ventana para iniciar el desarrollo de un circuito integrado desde cero. A lo largo de toda esta línea de investigación han aportado varios estudiantes del departamento, obteniendo una diversidad de resultados y proponiendo recomendaciones que se han tomado en cuenta para mejorar el proceso.

Cabe resaltar que algunos de los antecedentes que serán expuestos en esta sección, fueron replicados con el objetivo de ser tomados como punto de partida para la elaboración de este trabajo de graduación.

2.1. Primera iteración (año 2019)

Este año, al comenzar con la línea de investigación, se comenzó a investigar sobre las distintas aplicaciones provistas por Synopsys junto con su determinada función. Además, se definió un flujo de diseño tomando en cuenta investigaciones similares realizadas por otras instituciones. Gracias a esto, en el trabajo de graduación de Steven Rubio [2], se muestra un flujo que relaciona la mayoría de las etapas con la herramienta de Synopsys encargada de ello.

Por otra parte, en el trabajo de graduación de Luis Nájera [3] se plantea diseñar un circuito integrado que mostrara un texto por medio de caracteres ASCII. El cual contaría con 12 pines, 1 pin para Vdd y otro para Vss, 1 pin para la señal de reloj, y 8 pines de salida que mostrarían el código ASCII de cada carácter del texto en formato binario para que pudiera ser analizado por un microcontrolador externo.

El circuito para mostrar el texto fue concebido como una máquina de estados finitos para simplificar su diseño, por lo que, se realizaron tablas de verdad y por medio de *Logic Fryday* se obtuvo el esquemático. Seguido de esto, se realizó el primer paso del flujo de diseño que consiste en la generación de HDL del circuito. Este, originalmente, se planteó de manera *Structural*. Sin embargo, debido a su alta complejidad y descubrimiento que el siguiente paso, síntesis lógica, puede realizarse con HDL *behavioral*, se optó por emplear el segundo.

Finalmente, se utilizó la herramienta de *Design Vision* para sintetizar el HDL: a nivel de celdas y se utilizó la herramienta de *Formality* para verificar que el *netlist:* obtenido en esta etapa tuviera el mismo funcionamiento que su respectivo archivo de HDL.

Como recomendaciones de esta primera iteración se obtuvieron la adquisición del software provisto por *Cadence* para la elaboración de un flujo de diseño alterno al obtenido usando las herramientas de Synopsys. Además, se propuso la simulación del HDL por medio de un FPGA: para verificar, de forma física, su correcto funcionamiento.

2.2. Segunda iteración (año 2020)

En esta etapa se comenzó a investigar sobre cómo usar la herramienta de *VCS* para comprobar el funcionamiento del HDL y lo obtenido en la etapa de la síntesis lógica. Debido a la cantidad de pasos que conllevó realizar todo este proceso, se comenzó a automatizar este proceso por medio de una serie de scripts que permiten generar los archivos necesarios y obtener la síntesis lógica 1 y el *core:* del circuito 2. Esto abrió la puerta a buscar automatizar no solo esta etapa, sino todo el flujo de diseño planteado en el trabajo de Steven Rubio.

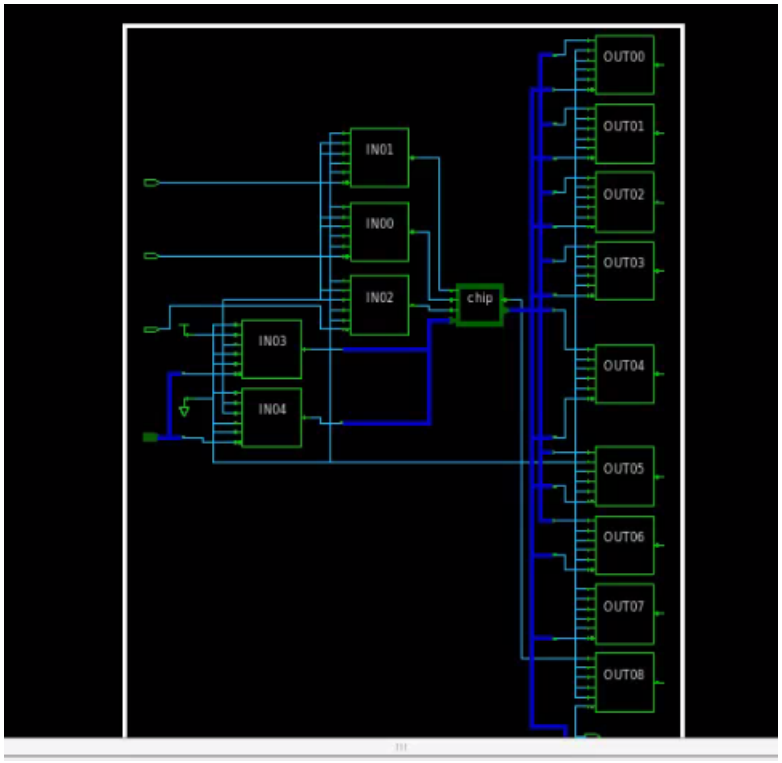


Figura 1: Síntesis lógica con I/O en *Design Vision* utilizando el script de automatización de esta etapa.

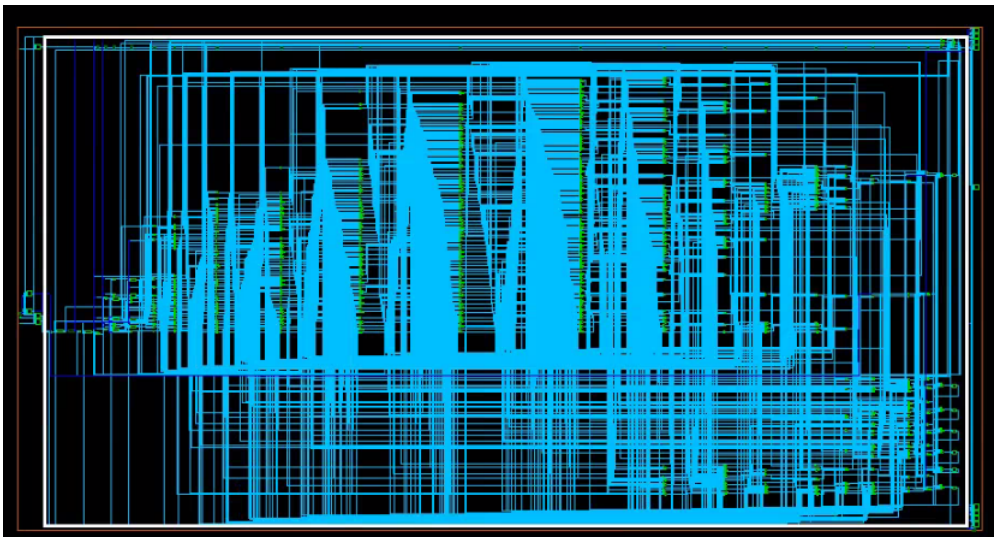


Figura 2: Síntesis lógica con I/O del *core* del chip *Design Vision* utilizando el script de automatización de esta etapa.

Es por lo que, se decidió crear un código en Python, al cual se le ingresara un texto, y proporcionara un archivo *verilog* el cual contuviera el HDL de la máquina de estados finitos que mostrara el texto manteniendo las especificaciones del circuito integrado establecido anteriormente. Descartando de esta manera la necesidad de utilizar *Logic Fryday*. Por otra

parte, siguiendo con el flujo de diseño, se comenzó a realizar la síntesis física por medio de la herramienta de *IC Compiler 1*; cuyo reto principal fue la creación del anillo de entradas y salidas. Una vez cumplido esto, se procedió a investigar sobre cómo utilizar la herramienta de *IC Validator* para realizar las verificaciones de diseño; lo que llevó a utilizar los *runsets* provistos por TSMC para realizar las verificaciones de Antena, ERC y LVS. Con lo obtenido se pudieron corregir errores de diseño y garantizar que el *layout* sea fabricable según las especificaciones del proceso de 180 nm. Puede encontrarse más información en los trabajos de Marvin Flores [4] y Ricardo Girón [5].

Originalmente, se tenía previsto que en esta iteración se enviaría a fabricar el circuito integrado y se le realizarían pruebas de rendimiento. Sin embargo, con la pandemia del Covid-19, se atrasó este proceso. Puesto que, se eliminaron las asistencias presenciales a los laboratorios. Para adaptarse a esta nueva realidad, se tuvo que gestionar con Synopsys, nuevos accesos a las computadoras de los laboratorios para que los estudiantes pudieras acceder a estas por medio de Splashtop:. Sin embargo, esto demoró hasta finales de julio de ese año.

Uno de los principales conocimientos obtenidos a lo largo de este año fue la existencia de *IC Compiler 2*, el cual es una opción más moderna que de *IC Compiler 1* para realizar la síntesis física. Esta nueva opción no se había explorado porque mucha de la documentación sobre flujo de diseño tomada como referencia, solamente exponían a *IC Compiler 1*. Por consiguiente, se propuso que en próximas investigaciones se busque adaptar esta nueva herramienta al flujo de diseño y se descarte el uso de *IC Compiler 1*.

2.3. Tercera iteración (año 2021)

Tomando en cuenta las recomendaciones presentes en los trabajos de graduación del año 2020. Se definió migrar el proceso de síntesis física a *IC Compiler 2*. Por tanto, el reto principal fue indagar sobre el funcionamiento de esta nueva herramienta y realizar la síntesis física 40 del circuito obtenido en la etapa de la síntesis lógica. De igual manera, implementar las verificaciones de antena [6], DRC [7] y ERC [8] para garantizar la correcta fabricación.

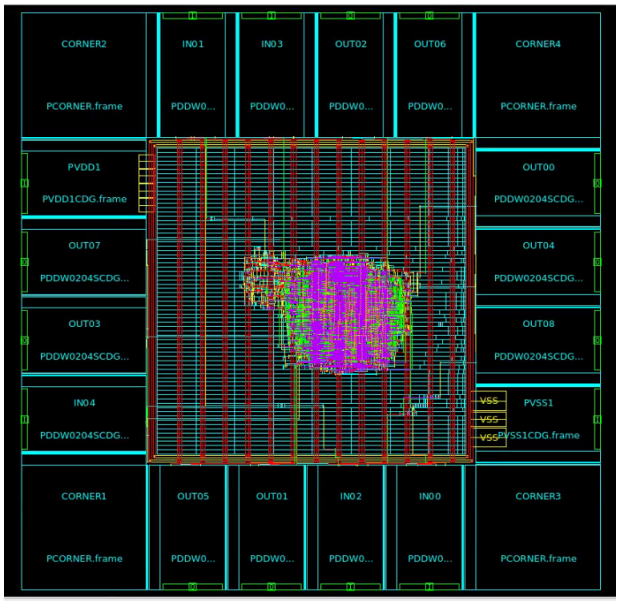


Figura 3: Síntesis física obtenida usando *IC Compiler 2*.

En esta ocasión se logró que el *layout* obtenido pasara sin errores la verificación de Antena y ERC. No obstante, al momento de depurar los errores mostrados en el DRC se encontraron 6 errores de densidad [4]. Se determinó que estor errores son producto del *runset* usando para ejecutar la prueba. Por tanto, se solicitó apoyo a TSMC para que proporcionaran el *runset* adecuado. Sin embargo, el *runset* proporcionado solo puede ser ejecutado usando Hercules, herramienta que ha sido discontinuada por Synopsys debido a la existencia de *IC Validator* la cual es mejor. Es por lo que, se determinó que en futuras investigaciones se propusieran posibles soluciones al problema.

```
                    LAYOUT ERRORS RESULTS: ERRORS

                  ##### ####  ####   ###  ####   ###
                  #     #   # #   # #   # #   # #
                  ####  ####  ####  #   # ####   ###
                  #     #  #  #  #  #   # #         #
                  ##### #   # #   #  ###  #   # ####

======================================================================

Library name:    EL_GRAN_JAGUAR.ndm
Structure name:  chip_IO
Generated by:    IC Validator RHEL64 S-2021.06-SP3-2.7200131 2022/01/06
Runset name:     /home/nanoelectronica/Documents/Gran_Jaguar_DENSITY/Sintesis_fisica/requisitos_tsmc/ICVLM18_LM16_LM152_6M.215a_pre041518
User name:       nanoelectronica
Time started:    2022/05/06 05:11:29PM
Time ended:      2022/05/06 05:15:01PM

Called as: icv -icc2 -f NDM -i EL_GRAN_JAGUAR.ndm -p /home/nanoelectronica/Documents/Gran_Jaguar_DENSITY/Sintesis_fisica -c chip_IO -clf /home/nanoelectronica/Documents/Gran_Jaguar_DENSITY/Sintesis_fisica/DRC/
slnFile.txt -icc_density_blockage -icc2_error_categories -I /home/nanoelectronica/Documents/Gran_Jaguar_DENSITY/Sintesis_fisica/DRC -icc2_error_browser INST -icc2_error_cell signoff_check_drc.err -rc /home/
nanoelectronica/Documents/Gran_Jaguar_DENSITY/Sintesis_fisica/DRC/signoff_check_drc.rc -I /home/nanoelectronica/Documents/Gran_Jaguar_DENSITY/Sintesis_fisica/DRC /home/nanoelectronica/Documents/
Gran_Jaguar_DENSITY/Sintesis_fisica/requisitos_tsmc/ICVLM18_LM16_LM152_6M.215a_pre041518
CLF: -sln "16 18 28 31 33 38 17 27 29 32 39 "

                        ERROR SUMMARY

 M1.R.1 : Min M1 area coverage < 30%
   density ..................................................... 1 violation found.

 M2.R.1 : Min M2 area coverage < 30%
   density ..................................................... 1 violation found.

 M3.R.1 : Min M3 area coverage < 30%
   density ..................................................... 1 violation found.

 M4.R.1 : Min M4 area coverage < 30%
   density ..................................................... 1 violation found.

 M5.R.1 : Min M5 area coverage < 30%
   density ..................................................... 1 violation found.

 M6.R.1 : Min M6 area coverage < 30%
   density ..................................................... 1 violation found.
```

Figura 4: Errores de densidad obtenidos al realizar DRC.

Por otra parte, se investigó sobre cómo mejorar y automatizar el proceso para realizar la prueba de LVS tomando en cuenta las recientes actualizaciones. Por tanto, se comenzó a utilizar la herramienta de *NetTran Tool* para obtener los *netlist* de esquemático. El principal reto de esta tarea fue encontrar una manera de llevar el orden de los archivos que se van

generando y lo que se van necesitando en cada etapa. [9]

Otro avance relevante obtenido en esta etapa del proceso fue la utilización de *StarRC* para la extracción de parásitos del circuito obtenido después de obtener la síntesis física. El objetivo de esto fue por simular en *HSPICE* el comportamiento del circuito a nivel transistor para garantizar que el *layout* no solamente es fabricable; sino que también funciona de la forma esperada. Para ello fue necesaria una investigación completa del funcionamiento de esta herramienta.

En esta tarea, se logró obtener un archivo '.sp' que contiene un subcircuito con las capacitancias y resistencias parásitas del circuito. Sin embargo, este subcircuito no cuenta con entradas ni salidas, por lo que sería necesario realizar un rastreo de conexiones para encontrar estos nodos antes de realizar la simulación. Otra limitante fue que dentro del subcircuito general se hizo referencia a otros subcircuitos que no están creados. Por lo que, para poder simular, primero se deben crear los *decks*, según las especificaciones de TSMC, de estos subcircuitos. Ambas tareas fueron asignadas para futuras investigaciones.

Cabe recalcar que una vez logrado lo expuesto anteriormente junto con los avances obtenidos en iteraciones pasadas, exceptuando los 6 errores de densidad en el DRC. Se cuenta con un flujo de diseño, con cada una de sus etapas automatizadas para diseñar un chip desde su archivo HDL hasta su GDS, el cual puede ser enviado a fabricar según las especificaciones del proceso de 180 nm de *TSMC*. Llegado a este punto, faltaría simular el circuito obtenido en extracción de parásitos para verificar el correcto funcionamiento del *layout* obtenido.

CAPÍTULO 3

Justificación

La invención del transistor en 1947 marcó un hito en el desarrollo tecnológico, dado que marcó el punto de partida a lo que se conoce como era digital. Desde ese momento, los dispositivos electrónicos, cada vez se vuelven más relevantes. En la actualidad, después de más de 70 años desde la invención del transistor, este elemento sigue siendo la parte elemental de cada dispositivo electrónico. Además, el avance de esta tecnología nos ha permitido hacer transistores a escala nanométrica, permitiendo que, en un espacio reducido, como lo es el procesador de una computadora, quepan billones de transistores.

A medida que crecía el número de transistores necesarios para hacer los dispositivos electrónicos de vanguardia, los ingenieros cada vez se enfrentaban a diseños más complejos, lo que hacía que los tiempos de desarrollo aumentaran o que fuera más sencillo cometer errores. Ante esto surgieron herramientas EDA lo que a su vez permitió el inicio de VLSI logrando así, mejorar el proceso de diseño.

Con los avances obtenidos gracias a las iteraciones anteriores dentro de esta línea de investigación, se ha logrado obtener el layout de un circuito integrado que, a su vez, cumple con las restricciones de diseño para la tecnología de 180nm, exceptuando 6 errores de densidad expuestos en la sección de antecedentes. Llegado a este punto, resulta necesaria la simulación del circuito obtenido para verificar que su comportamiento sea el esperado. En otras palabras, que se muestre la secuencia de ASCII de la forma esperada.

Para efectuar la simulación se emplea el archivo '.sp' obtenido en la etapa de extracción de parásitos para simular el comportamiento del circuito integrado a nivel transistor, tomando en cuenta sus resistencias y capacitancias parásitas. Sin embargo, este archivo cuenta con subcircuitos de HSPICE, que, si bien, son referenciados en el archivo, no son creados por este. Por lo cual, al ejecutar la simulación solamente con este '.sp' genera error puesto que los subcircuitos no han sido creados. Por lo que, si se desea simular el trabajo obtenido en iteraciones anteriores, se deben crear los decks en HSPICE siguiendo las especificaciones provistas por TSMC para que cada uno de estos subcircuitos se comporte acorde a lo especificado por el fabricante.

Una vez creados los decks de HSPICE, se puede simular el funcionamiento del chip. Para ello se debe usar la herramienta de *PrimeSim*, sin embargo, es una herramienta reciente, por lo que la única documentación que se tiene de esta es la provista por el fabricante. Por tanto, resulta sumamente importante investigar sobre como usar la herramienta para adaptarla a las necesidades de la investigación. Además, elaborar manuales de referencia para que futuras generaciones puedan tener acceso a documentación de forma inmediata. Una vez se haya simulado correctamente el circuito para validar su correcto funcionamiento, se obtiene la certeza que todo lo que se ha hecho en la línea de investigación funciona correctamente. Con lo cual, se obtendrá una metodología de diseño válida para futuras creaciones del departamento.

Cabe mencionar que una vez se haya logrado fabricar este circuito integrado, se conseguirá marcar un hito en la historia de Guatemala, dado que al ser un proceso sumamente complejo y poco conocido en la región; demostrará que en el país se tiene la capacidad para ejecutar procesos e investigación de primer nivel. Esto abre una ventana muy grande de oportunidades de desarrollo tecnológico en el país, al igual que el inicio de la industria de semiconductores; el cual sería un gran logro, ya que fabricar un chip es un proceso que, por lo general, solo tienen acceso los países desarrollados debido a sus altos niveles de dificultad y de costos.

CAPÍTULO 4

Objetivos

4.1. Objetivos generales

- Crear el primer circuito integrado funcional del país y la región usando tecnología de 180nm y obtener un flujo de diseño eficiente y automatizado que pueda ser utilizado por estudiantes de la UVG para futuros proyectos.
- Crear Decks en HSPICE para simular correctamente el circuito integrado obtenido en la etapa de extracción de parásitos; para verificar su correcta funcionalidad usando la nueva herramienta de *PrimeSim*

4.2. Objetivos específicos

- Replicar resultados obtenidos en trabajos de graduación de promociones anteriores relacionadas con el chip de El Gran Jaguar.
- Crear decks en HSPICE para la simulación y corrección de posibles errores de diseño en el chip de El Gran Jaguar.
- Crear una base de datos de decks en HSPICE para la simulación de futuros proyectos.
- Simular en HSPICE el circuito obtenido en la etapa de extracción de parásitos y corregir posibles errores.
- Encontrar y analizar errores mostrados al realizar las verificaciones de diseño al layout del chip y retroalimentar a los equipos encargados de corregirlos.
- Conocer a profundidad el funcionamiento de la herramienta de *PrimeSim HSPICE* y sus herramientas de simulación.
- Crear un manual de inicio rápido y una serie de videos cortos para que futuras generaciones puedan entender la nueva herramienta de *PrimeSim*.

CAPÍTULO 5

Alcance

El alcance de este proyecto consiste en documentar una serie de análisis que se pueden realizar por medio de *PrimeSim* HSPICE y que pueden ser útiles para analizar el circuito de El Gran Jaguar y verificar su correcto funcionamiento. De igual manera, se busca explorar herramientas de simulación que puedan ser utilizadas en *PrimeSim* HSPICE de tal forma que en un futuro pueda servir de guía para realizar análisis y simulaciones más complejas de los diseños que sean hechos por el departamento de ingeniería electrónica, mecatrónica y biomédica, producto de la línea de investigación de nanoelectrónica.

CAPÍTULO 6

Marco teórico

6.1. Electrical Design Automation

Debido al alto grado de complejidad que conlleva el diseño electrónico, desde mediados de los años 70, se ha iniciado el desarrollo de herramientas computacionales que permitan la realización de esta tarea.

6.1.1. Simulation Program with Integrated Circuits Emphasis

Esta herramienta fue desarrollada por Laurecen Nagel y Donald Pederson en la Universidad de Berkeley en 1973. También es conocido, por sus siglas, como *SPICE* y representa a un conjunto de software de *EDA*, que se encarga de simular circuitos electrónicos analógicos a su escala más fundamental, en la que operan los transistores, capacitores, resistencias, diodos, entre otros.

La metodología para emplear esta herramienta consiste en describir un circuito por medio de texto, usando la sintaxis de *SPICE*, y seleccionar un tipo de análisis a realizar; siendo los más comuntes: análisis transisnte, análisis AC y análisis DC. Gracias a esto, se pueden obtener datos sobre la frecuencia de operación, análisis de timing y consumo de potencia del circuito que está siendo analizado.

6.2. Synopsys

El software que posee la Universidad del Valle de Guatemala para llevar a cabo el flujo de diseño del chip es proveído por la empresa *Synopsys*. Por medio de ellos, se tiene acceso a un conjunto de herramientas especializadas que permiten completar cada etapa dentro del flujo de diseño de un chip.

Para realizar simulaciones y análisis de circuitos empleando *SPICE*, *Synopsys* provee una plataforma conocida como *PrimeSim*. Esta está conformada por herramientas que hacen funciones específicas. Pero, al combinar la información generada en cada una, se obtiene información muy detallada y realista sobre el comportamiento de un determinado circuito.

Las herramientas que conforman *PrimeSim* o pueden utilizarse para generar análisis en esta misma plataforma son: *PrimeSim Spice, PrimeSim Continuum, PrimeSim HSPICE, PrimeSim Pro, PrimeSim XA, PrimeSim Reliability Analysis, PrimeSim Custom Fault, PrimeWave Design Environment* y *Custom Compiler*.

6.2.1. PrimeSim Pro

Es parte esencial de *PrimeSim Continuum*. Su valor agregado radica en su capacidad para realizar análisis complejos a diseños de DRAM y de memoria Flash. Además, la arquitectura de software en la que se ha basado está sumamente optimizada para manejar la memoria de un chip complejo y diseños CMOS de sensores de imagen con un alto ancho de banda, redes de distribución, gran potencia y estrictos requerimientos de densidad. [10]

Cabe recalcar que *PrimeSim Pro* está completamente integrado con *PrimeWave* y funge un papel fundamental para realizar el análisis de fiabilidad de *PrimeSim*. [10]

6.2.2. PrimeSim SPICE

Es un simulador de circuitos de alto rendimiento para aplicaciones analógicas, radiofrecuencia o de señales mixtas. Esta herramienta soporta análisis de ruido de alta frecuencia, manejo eficiente de parámetros de dispersión (parámetros-S) mientras ofrece un análisis avanzado de capacidades para aplicaciones no periódicas y dependientes del tiempo; así como aplicaciones que dependen de la frecuencia. [11]

Además, este simulador es el motor principal para el análisis de fiabilidad de *PrimeSim* (*PrimeSim Reliability Analysis*). Por otra parte, para diseños RF como mezcladores, VCOs (Voltage Controlled Oscillators), PLLs (Phase-locked loop), LNAs (low noise amplifiers); o para diseños analógicos o de señales mezcladas como convertidores de datos, moduladores Delta-Sigma, *PrimeSim SPICE* ofrece diversos análisis en dominio del tiempo, dominio de la frecuencia y de ruido. También, puede ser integrado con VCS permitiendo simulaciones de alto rendimiento con *SPICE*, *Verilog*, *SystemVerilog, Verilog-AMS* y *VHDL*. [11]

6.2.3. PrimeSim XA

Forma parte integral de *PrimeSim Continuum*. Su función principal es el análisis y verificación de diseños de SRAM y SoC de señales mezcladas. Debido a la inteligencia de simulación, tiene la capacidad de detectar y reconocer dispositivos, topologías y jerarquías para emplear técnicas de simulación que se adapten al diseño. Entre sus principales ventajas se encuentra su certificación para un amplio rango de procesos de fabricación; entre ellos, los procesos de *FinFET* mantenidos por *TSMC*; también cuenta con un conjunto completo

de las capacidades del análisis de MonteCarlo. [12]

Cabe recalcar que está basado en la librería de modelos de *HSPICE*. Además, está integrado completamente con el simulador digital *VCS*. [12]

6.2.4. PrimeSim Custom Fault

Esta herramienta surge de la necesidad de realizar verificaciones de seguridad y pruebas de cobertura a circuitos integrados usados en aplicaciones de seguridad crítica, como lo puede ser en la industria automotriz o aeronáutica, para reducir, eficientemente, fallas de diseño y mejorar la seguridad. Para cumplir con esta tarea, cuenta con una interfaz gráfica, en la cual se pueden realizar pruebas de análisis de falla de silicio, verificación de seguridad funcional o prueba de cobertura de manufactura. Además, cuenta con tecnología AWRS (*Adaptive Weighted Random Sampling*), la cual emplea métodos de muestreo para reducir la cantidad de simulaciones de fallas drásticamente. [13]

6.2.5. PrimeSim HSPICE

PrimeSim HSPICE surge en nuevas versiones lanzadas por *Synopsys*, para sustituir *HSPICE*. Esta nueva herramienta se encarga de hacer simulaciones de circuitos y ofrece modelos de *MOS* que han sido certificados por los fabricantes de circuitos integrados. Asimismo, cuenta con una diversidad de algoritmos de análisis. Gracias a esto, se pueden realizar simulaciones de integridad de señal, caracterizar celdas y memorias y analizar circuitos analógicos y digitales, tomando en cuenta las propiedades físicas y eléctricas presentes en los transistores de los fabricantes. [14]

Esta herramienta está completamente integrada a *PrimeWave*, con lo cual se mejora el flujo de diseño y trabajo.

Sección de ayuda

Se puede acceder a la documentación de *PrimeSim HSPICE* en formato PDF usando el siguiente comando:

```
% hspice -doc
```

También se puede acceder a la documentación en línea de la plataforma por medio de un buscador usando formato HTML por medio del comando:

```
% hspice -help
```

Archivos de demostración

Al momento de instalar *PrimeSim HSPICE*, se pueden localizar archivos de demostración en la siguiente dirección:

```
$installdir/demo/hspice/
```

Donde *$installdir* corresponde al directorio donde se instaló *PrimeSim HSPICE*. Dentro de esta carpeta se encuentran los directorios mostrados en el Cuadro 15 y la forma de ejecutar estos ejemplos depende del sistema operativo que se esté utilizando. Si se está en el ambiente de Windows, se debe utilizar *PrimeSim HSPICE User Interface* (HSPUI). Mientras que, si se está utilizando en Linux, se debe ejecutar el siguiente comando desde la terminal:

```
%> hspice -i archivo_demo_seleccionado -o archivo_salida
```

Directorio	**Descripción**
/spiceADE_integ	Tutorial de integración de PrimeSim HSPICE con Cadence® Virtuoso® Analog Design Environment
/apps	Aplicaciones generales
/back_annotation	Utilización de las BA_options
/behave	Componentes de comportamiento analógico
/bench	Pruebas de rendimiento estándar
/bisect	Optimización de bisección
/bjt	Componentes bipolares
/cchar	Características de celdas prototipo
/ciropt	Optimización a nivel de circuito
/devopt	Optimización a nivel de dispositivos
/encryption	Encriptación tradiciones, 8-bits y 3DES
/fft	Análisis de Fourier
/filters	Filtros
/ibis	Ejemplos de IBIS
/lstb	Análisis de estabilidad de retroalimentación
/mag	Transformadores y componentes de núcleo magnético
/mos	Componentes MOS
/rf_examples	Ejemplos de circuitos RF
/si	Aplicaciones de integridad de señal
/sources	Fuentes de alimentación dependientes e independientes
/sparam	Aplicaciones de parámetro-S
/tline	Filtros y líneas de transmisión
/twline	Líneas de transmisión de elementos W y solucionadores de campo.
/variability	Bloque de variación, Monte Carlo, y ejemplos de discordancia AC/DC
/veriloga	Ejemplos de Verilog-A

Cuadro 1: Árbol de directorios de archivos de demostración proporcionados por PrimeSim.

Invocar simulación de HSPICE

Al momento de invocar una simulación de *HSPICE*, *PrimeSim HSPICE* ejecuta los siguientes pasos:

1. Se realiza la invocación de los archivos de entrada y salida. Por ejemplo, al ejecutar el comando:

   ```
   %> hspice -i demo.sp -o salida.lis
   ```

   se invoca la herramienta en el archivo demo.sp y se direccionan las salidas de la simulación al archivo salida.lis

2. La herramienta de *PrimeSim HSPICE* determina la mejor arquitectura de ejecucion y determina la versión de la herramienta.

3. La herramienta de *PrimeSim HSPICE* lee las varaibles de ambiente relacionadas con la licencia del producto y verifica la validez de estas. De no ser autorizada la licencia, se cancela la ejecución de la simulación.

4. *PrimeSim HSPICE* lee el correspondiente archivo *hspice.ini*

5. Se abre el *netlist* de entrada (siguiendo el ejemplo inicial seria demo.sp), si este archivo no existe se genera error. Seguido de esto, se abre el arhivo de salida (salida.lis).

6. Se abren los archivos que fueron incluidos en el *netlist* de entrada mediante la instrucción .INCLUDE o .LIB

7. Se realiza la lectura de las condiciones iniciales definidas con las instrucciones .IC y .NODESET. Además, se encuentra un punto de operación cuya inforación se puede guardar usando el comando .SAVE

8. Se ejecuta el barrido especificado de diseño y produce un conjunto de archivos de salida.

9. Mediante la utilización del comando .ALTER, se pueden alterar las condiciones de la simulación y repetir análisis.

10. Mediante la combinación de las teclas Ctrl + Z se puede suspender la ejecución de un análisis usando *PrimeSim HSPICE*.

11. Una vez finalizada la simulación, *PrimeSim HSPICE* cierra todos los archivos abiertos.

Tipos de simulación

La herramienta de *PrimeSim HSPICE* puede simular circuitos en el dominio del tiempo y la frecuencia. Asimismo, provee capacidades de análisis que permiten el desarrollo de circuitos de última generación. En el Cuadro 2 se muestra un listado de los tipos de análisis que puede realizar esta herramienta.

Análisis	Descripción
.AC	Análisis de barrido AC
.ACMATCH	Analiza el efecto de variaciones locales en la respuesta AC del circuito.
.ACPHASENOISE	Análisis de ruido de fase AC
.BIASCHK	Comprobación de sesgo dinámico
.DC	Análisis de barrido DC
.DCMATCH	Analiza el efecto de las variaciones locales en las características DC de un circuito
.DCSENS	Análisis DC de sensibilidad utilizando variaciones especificadas de variación por medio de bloques de variación
.DISTO	Análisis de distorsión
.ENV	Análisis de envolvimiento
.ENVFFT	Análisis de envolvimiento FFT
.ENVOSC	Análisis de envolvimiento para osciladores
.FFT	Análisis de FFT
.FOUR	Análisis de Fourier
.HB	Análisis de balance armónico
.HBAC	Análisis AC de balance armónico multi-tono
.HBLIN	Traducción de frecuencia y extracción de parámetros-S
.HBLSP	Análisis de señal larga de parámetros-S
.HBNOISE	Análisis de ruido de balance armónico multi-tono
.HBOSC	Análisis de balance armónico para osciladores
.HBXF	Análisis de la función de transferencia de balance armónico
.LIN	Análisis de red lineal
.LSTB	Análisis de estabilidad de lazo
.MOSRA	Análisis de fiabilidad de MOSFET
.NOISE	Análisis de ruido de señal pequeña
.OP	Análisis de punto de operación
.PHASENOISE	Análisis de ruido de fase
.PTDNOISE	Análisis de ruido periódico en el dominio del tiempo
.PZ	Análisis de polos y ceros
.SAMPLE	Análisis de ruido de muestreo de datos
.SENS	Determina la sensibilidad de una señal DC pequeña de una variable de salida para los parámetros del circuito.
.SN	Análisis de disparo de Newton
.SNAC	Análisis AC de disparo de Newton
.SNFT	Análisis de transformada discreta de Fourier usando el método de disparo de Newton
.SNNOISE	Análisis de ruido usando el método de disparo de Newton
.SNOSC	Análisis de disparo de Newton para un oscilador
.SNXF	Análisis de la función de transferencia usando el método de disparo de Newton
.STATEYE	Análisis estadístico de diagrama de ojo
.TF	Calcula valores de señales pequeñas para funciones de transferencia en simulaciones AC y DC
.TRAN	Análisis transiente
.TRANNOISE	Análisis de ruido transiente

Cuadro 2: Tipos de análisis provistos por *HSPICE*

Análisis de Monte Carlo

Monte Carlos es un análisis que consiste en parametrizar una o más características de un circuito, variar de forma aleatoria los valores de dichas características y realizar una simulación en PrimeSim HSPICE por cada vez que se realice una variación de estos parámetros. Cada una de estas ejecuciones se conoce como barrido y permiten la generación de gráficos y tablas que posteriormente pueden ser analizadas para estudiar las condiciones de operación del circuito.

El algoritmo de Monte Carlo, posee tres formas para variar el valor de alguna característica del circuito. Estas son: gaussiana, uniforme y de límite.

Sintaxis del análisis

- Definir el o los parámetros que se desean variar y asignarle el tipo de distribución al que estará expuesto.
- En el *netlist*, reemplazar el parámetro del elemento o modelo que se desea estudiar, por el parámetro que se estará variando mediante Monte Carlo.
- Incluir la opción de SWEEP y MONTE en la instrucción encargada del análisis.

La simulación de Monte Carlo debe ser utilizada en conjunto con análisis transitorio, DC o AC para que esta pueda ser ejecutada correctamente. La diferencia entre utilizar únicamente alguno de los análisis citados anteriormente en comparación a utilizarlos en conjunto con Monte Carlo, es que esta última, permite hacer una simulación más completa de como se comportará el circuito diseñado ante la variación de diversos factores. Esto resulta súmamente útil cuando se desea evaluar como se comportará el diseño en un ambiente no controlado, lo cual es más realista que hacer una simulación ideal.

Elementos

PrimeSim HSPICE posee declaraciones de elementos que describen el *netlist* de dispositivos y fuentes. Según la sintaxis, cada elemento debe comenzar por medio de una única letra que identifica el tipo de dispositivo. El Cuadro 3 muestra cada una de estas letras únicas y el elemento que representan.

Elemento	Descripción
B	Buffer IBIS
C	Capacitor
D	Diodo
E	Fuente de voltaje dependiente de voltaje
F	Fuente de corriente dependiente de corriente
G	Fuente de corriente dependiente del voltaje
H	Fuente de voltaje dependiente de la corriente
I	Fuente de corriente independiente
J	JFET o MESFET
K	Inductor mutuo
L	Inductor
M	MOSFET
P	Puerto
Q	Transistor BJT
R	Resistencia
S	Parámetro S
T	Línea de transmisión ideal
U	Línea de transmisión con pérdidas
V	Fuente de voltaje independiente
W	Línea de transmisión con pérdidas (preferida)
X	Subcircuito

Cuadro 3: Listado de elementos brindados por *HSPICE*

Modelos

Los modelos de dispositivos son plantillas que contienen la definición de varias versiones de cada tipo de elemento soportado por las *netlist* de la herramienta *PrimeSim HSPICE*. Por ejemplo, transistores, diodos, capacitores, entre otros. La principal ventaja de la utilización de estos elementos son la alta eficiencia y velocidad para la creación y simulación de un diseño de circuito. Dentro del *netlist* cada elemento que referencia a un modelo es considerado como una instancia de este.

PrimeSim HSPICE permite la creación de modelos por medio de un archivo '.sp'; o bien, se puede instanciar alguno de los modelos que ya vienen incluidos en la herramienta. Los cuales se encuentran listados en las tablas 4 y 5.

Capacitores	
CMC Varactor	Level=7, Modelo varicap CMC
Resistencias	
CMC R2 Resistor	Level=2, Modelo de resistencia R2 CMC
CMC R3 Resistor	Level=5, Modelo de resistencia R3 CMC
Diodos	
Non-Geometric Junction Diode	Level=1, Modelo de diodo de junción no geométrico
Fowler-Nordheim Diode	Level=2, Modelo de diodo Fowler-Nordhiem
Geometric Junction Diode	Level=3, Modelo de diodo de junción geométrico
JUNCAP1 Diode	Level=4, Modelo de diodo de junción con capacitancia
JUNCAP2 Diode	Level=6, Modelo de diodo de junción con capacitancia
Philips D500 Diode	Level=5, Modelo de diodo Philips D500 advanced
CMC Diode	Level=7, Modelo de diodo CMC
JFETs y MESFETs	
Level 1 JFET	Level=1, Modelo de JFET
Level 2 JFET	Level=2, Modelo de JFET
Level 3 MESFET	Level=3, Modelo de MESFET
TriQuint MESFET	Level=7, Modelo de TriQuint MESFET
Materka MESFET	Level=8, Modelo de Materka MESFET
BJTs	
Level 1 BJT	Level=1, Modelo de BJT
Level 2 BJT	Level=2, Modelo de BJT
VBIC	Level=4, Modelo de VBIC
Philips Bipolar MEXTRAM Level 503	Level=6, Modelo de BJT Philips Bipolar
Philips Bipolar MEXTRAM Level 504	Level=6, Modelo de BJT Philips Bipolar
HICUM	Level=8, Modelo de BJT HICUM
VBIC99	Level=9, Modelo de BJT VBIC99
Philips MODELLA Bipolar	Level=10, Modelo de BJT Philips MODELLA Bipolar
UCSD HBT	Level=11, Modelo de BJT UCSD HBT
HICUM/L0	Level=13, Modelo de BJT HICUM/L0

Cuadro 4: Modelos de diodos, resistencias, capacitores, BJTs, MESFETs y JFETs provistos por *PrimeSim HSPICE*

MOSFETs	
Schichman-Hodges	Level=1, Modelo de Schichman-Hodges
MOS2 Grove-Frohman (SPICE 2G)	Level=2, Modelo de MOS2 Grove-Frohman
MOS3 Empirical (SPICE 2G)	Level=3, Modelo de MOS3 Empirical
Grove-Frohman Level 2	Level=4, Modelo de Grove-Frohman
AMI-ASPEC	Level=5, Modelo de AMI-ASPEC
Lattin-Jenkins-Grove (ASPEC)	Level=6, Modelo de Lattin-Jenkins-Grove (ASPEC)
Lattin-Jenkins-Grove (SPICE)	Level=7, Modelo de Lattin-Jenkins-Grove (SPICE)
Advanced Level 2 MOS2 Grove-Frohman	Level=8, Modelo de Grove-Frohman (SPICE 2G)
BSIM	Level=13, Modelo BSIM
SOSFET	Level=27, Modelo SOSFET
BSIM Derived	Level=28, Modelo derivado BSIM
Cypress Depletion	Level=38, Modelo de depleción de ciprés
BSIM2	Level=39, Modelo BSIM2
HP a-Si TFT	Level=40, Modelo HP a-Si TFT
BSIM3 Version 2 MOSFET	Level=47, Modelo BSIM3 Version 2
BSIM3 Version 3 MOSFET	Level=49 y Level=53 Modelo BSIM3 versión 2
Philips MOS9	Level=50, Modelo Philips MOS9
BSIM4 MOSFET	Level=54, Modelo BSIM4
EPFL-EKV MOSFET	Level=55, Modelo EPFL-EKV
BSIM3 SOI	Level=57, Modelo BSIM3 SOI
UFSOI MOSFET	Level=58, Modelo UFSOI MOSFET
BSIM3 SOI FD	Level=59, Modelo BSIM3 SOI FD
BSIM3 SOI DD	Level=60, Modelo BSIM3 SOI DD
RPI a-Si TFT	Level=61, Modelo RPI a-Si TFT
RPI Poli-Si TFT	Level=62, Modelo RPI Poli-Si TFT
Philips MOS11	Level=63, Modelo Philips MOS11
STARC HiSIM MOSFET	Level=64, Modelo STARC HiSIM
SSIMSOI	Level=65 SSIMSOI Model
PrimeSim HSPICE HVMOS	Level=66, Modelo PrimeSim HSPICE HVMOS
STARC HiSIM2 MOSFET	Level=68, Modelo STARC HiSIM2
PSP	Level=69, Modelo PSP
BSIMSOI4	Level=70, Modelo BSIMSOI4
Level 71 TFT	Level=71, Modelo TFT
BSIM-CMG MOSFET	Level=72, Modelo BSIM-CMG
HiSIM_HV	Level=73, Modelo HiSIM_HV
Level=74 MOS Model 20	Level=74, Modelo MOS 20
LETI-UTSOI MOSFET	Level=76, Modelo LETI-UTSOI
Level 77 BSIM6 MOSFET	Level=77, Modelo BSIM6
BSIM-IMG	Level=78, Modelo BSIM-IMG
EKV3	Level=79, Modelo EKV3

Cuadro 5: Modelos de MOSFETs provistos por *PrimeSim HSPICE*

Factores de escala

La herramienta de *PrimeSim HSPICE* utiliza el metro, kilogramo y segundo como unidades de medidas por defecto. Sin embargo, es capaz de aceptar la unidad *mil* (0.001 pulgadas) como unidad de longitud de entrada. Por otra parte, para finalidades de simplificar valores de longitud, peso o tiempo muy grandes o muy pequeños, se cuenta con factores de escala 6 que facilitan el diseño y análisis de simulaciones.

Factor de escala	**Prefijo**	**Símbolo**	**Factor de multiplicación**
T	tera	T	1e+12
G	giga	G	1e+9
ME, MEG, X, or Z	mega	M, ME, X, or Z	1e+6
K	kilo	k	1e+3
MI or MIL	n/a	MI or MIL	25.4e-6
U	micro		1e-6
N	nano	n	1e-9
P	pico	p	1e-12
F	femto	f	1e-15
A	atto	a	1e-18
DB	DB	db	10(valor/20)
MIN	MIN	min	60
HR	HR	hr	3600
DAY	DAY	day	86400
YR	YR	yr	31536000

Cuadro 6: Factores de escala soportados en *PrimeSim HSPICE*

Estructura de netlist

Los *netlist* utilizados en *PrimeSim HSPICE* pueden ser diseñados usando un editor de texto o generados por medio de *netlisters* a partir de un esquemático. Cabe recalcar que *PrimeSim HSPICE* acepta *netlist* jerarquicos (dependen de referencias hacia otros *netlist*) y *netlist* planos (no dependen de *netlist* externos).

Para diseñar un netlist es necesario seguir un orden para que *PrimeSim HSPICE* puede interpretar el *SPICE* de forma correcta y evitar errores de simulación. Para ello, se recomienda seguir una estructura determinada 7 para sacar el mayor provecho a la herramienta.

Sección	Comando	Descripción
Título	.TITLE	Esta sección es opcional y define el título del netlist de entrada.
Configuración	.OPTION	Establece las condiciones para la simulación
	.IC o NODESET	Valores iniciales del circuito y subcircuito
	.PARAM	Establece los valores de los parámetros del netlist
	.GLOBAL	Sets the node name globally in the netlist.
Fuentes de alimentación	Fuentes y entradas digitales	Establece las señales de entrada del circuito (voltaje y corriente)
Netlist	.SUBCKT, .ENDS .MACRO, .EOM	Elementos de circuito para simulación, definición de subcircuitos y macros.
Análisis	.TRAN, .AC, .DC,...	Establece los tipos de análisis a realizar
	.SAVE	Guarda información del punto de operación
	.LOAD	Carga información del punto de operación
	.DATA	Crea una tabla para análisis basado en datos.
	.TEMP	Establece un análisis de temperatura
Inclusión de modelos, librerías y archivos	.INCLUDE	Inclusión de archivos generales
	.MODEL	Descripciones de modelos de elementos
	.LIB	Librería de modelos
	.MALIAS	Asigna un alias a diodo, BJT, JFET o MOSFET.
	.OPTION SEARCH	Busca la dirección de archivos o librerías incluidas
Salida	.PRINT, .PROBE	Declaraciones de variables de salida
	.MEASURE	Declaración para evaluar y reportar funciones definidas por el usuario de un circuito.
Bloques de alteración	.ALTER	Secuencia para el análisis de casos en línea.
	.DEL LIB	Remueve selección de librerías previas
Fin del netlist	.END	Final del netlist (obligatorio)

Cuadro 7: Estructura recomendada de un archivo netlist de entrada

Archivos de salida

Al momento de ejecutar una simulación en *PrimseSim HSPICE*, dependiendo de esta, se generarán una serie de archivos que poseen información. Esta puede variar desde una medición realizada en un momento específico de la simulación, o una serie de datos que pueden llegar a ser vistos en herramientas externas como *WaveView* o *PrimeWave*.

Descripción	Extensión
Resultado de las mediciones del análisis AC	.ma#
Resultados del análisis AC (obtenidas con las instrucción .POST)	.ac#
Resultados del análisis de Monte Carlo	.mc#
Resultados de minería de datos	.mpp0
Resultado de las mediciones de análisis DC	.ms#
Resultados del análisis DC (obtenidos con las instrucción .POST)	.sw#
Grafo de datos del análisis de FFT	.ft#
Información del punto de operación (.OPTION OPFILE)	.dp#
Condiciones iniciales de voltajes en nodos	.ic#
Listado de salidas	.lis
Estado de salidas	.st#
Tabla de salida (.DCMATCH OUTVAR)	.dm#
Resultados del análisis de polos y ceros	.pz#
Resultados de las mediciones de una análisis StatEye	.mste#
Mediciones del transiente inicial de un análisis StatEye	.mtNp{f}#
Análisis del transiente inicial de un análisis StatEye	.trNp{f}#
Resultados en el dominio del tiempo del análisis StatEye	.stet#
Resultado de voltajes del análisis StatEye	.stev#
Listado cruzado de subcircuitos	.pa#
Resultado de mediciones de análisis transiente	.mt#
Análisis de resultados de análisis transiente (con las instrucción.POST)	.tr#
Archivos para utilizar con Synopsys WaveView/SX tools (.OPTION WDF)	*_wdf.tr#, *_wdf.sw#, o *_wdf.ac#

Cuadro 8: Archivos de salida generados por *PrimeSim HSPICE* donde el caracter # representa un número aleatorio.

6.2.6. PrimeSim Continuum

El constante aumento en la complejidad de los circuitos integrados a diseñar ha generado que aumenten las etapas del flujo de diseño, provocando que este se torne cada vez más difícil de manipular. A medida que se modernizan las tecnologías de fabricación, aumenta la cantidad de parásitos y variaciones de procesos en los nodos del circuito. Ante esta problemática surge *PrimeSim Continuum*, el cual provee una plataforma unificada que permite acelerar el flujo de diseño. Para ello, utiliza los motores computacionales que ofrecen cada una de las aplicaciones de *PrimeSim*. [15]

6.2.7. PrimeSim Reliability Analysis

Esta herramienta surge de la necesidad de los diseñadores de circuitos integrados de garantizar que sus creaciones completamente fiables en actividades donde la seguridad es crítica. Ejemplos de estos casos se dan en la industria espacial, automotriz o aeronáutica,

en donde un error de transistor podría derivar en una catástrofe. [16]

Ante esto, *PrimeSim Reliability Analysis*, provee un entorno amigable con el usuario para realizar pruebas y análisis que unifica las características del método de producción de los fabricantes, junto con tecnologías de análisis de fiabilidad certificadas por estos fabricantes. Por consiguiente, cuenta con análisis de electro migración, *IR drop*, Montecarlo, envejecimiento de *MOS*, simulación de fallas analógicas y *ERC*. Con esto se garantiza una verificación de fiabilidad a lo largo de todo el ciclo de vida del circuito integrado. [16]

6.2.8. PrimeWave Design Environment

Es un ambiente comprensible y flexible para configurar y ejecutar análisis a diseños de memorias, circuitos de radiofrecuencia, circuitos analógicos y señal mezclada, que fueron generados empleando las aplicaciones de diseño personalizado de Synopsys. Su interfaz gráfica le permite al usuario ejecutar y comparar simulaciones distintas. Además, permite realizar análisis para validar la integridad de señales determinadas. [17]

Entre sus características principales se encuentra su unificado flujo de trabajo entre todas las herramientas de *PrimeSim* para todos los tipos de análisis; así como, su integración con *Custom Compiler* para análisis para circuitos de radiofrecuencia, analógicos y de señal mezclada. Además, capacidad para interpretar scripts basados en TCL para la programación de *testbenches* complejos para realizar regresiones y procesamiento de resultados. [17]

6.2.9. Custom Compiler

Esta plataforma representa el núcleo de diseño de Synopsys. Su diseño está hecho para satisfacer los requerimientos más grandes para el diseño de circuitos usando FinFET. *Custom Compiler* cuenta con un entorno gráfico para simplificar el trabajo del usuario. Este cuenta con opciones especializadas para la corrección de errores, configuración de simulaciones, diseño de señales mezcladas, análisis y reporte de resultados. Además, cuenta con un espacio para entrada de archivos de tipo esquemático (*schematic*) por medio de un editor especializado; el cual, a su vez, permite el cableado automático, generación de símbolos, edición de parámetros, entre otros. Este, de igual forma, cuenta con pruebas para la detección de errores de alimentación de potencia, conexiones. También permite reportar cambios que se han hecho entre esquemáticos. [18]

Por otra parte, también cuenta con un editor de texto, el cual soporta código de *verilog*; y que puede usarse en conjunto con el editor de esquemáticos para diseños de circuitos.

Custom Compiler se encuentra basado en la base de datos OpenAccess. Gracias a esto, posee una gran variedad de APIs que permiten personalizar la experiencia de usuario e integrar herramientas externas para realizar aplicaciones externas, tales como: extracción de parásitos, síntesis física, verificaciones, simulación de circuitos, entra otras. Además, soporta extensiones de lenguaje que incluye Python, TCL o C++. [18]

6.3. Análisis de integridad de señal con PrimeSim

El rendimiento de un circuito integrado no depende únicamente de la cantidad de transistores que pueden ser agrupados dentro de un chip. Debido a la cercanía de los componentes dentro del empaquetado, también depende de la frecuencia de reloj de operación, líneas de transmisión, entre otras. Para garantizar que las señales que recorren los *interconnetcs* del chip son correctas, se realizan análisis de integridad de señal en donde se tienen en cuenta los fenómenos físicos intrínsecos de los transistores y metales al momento de transmitir una señal.

Para realizar una correcta simulación de integridad de señal, la herramienta de *PrimeSim HSPICE*, modela los siguientes aspectos:

- Celdas conductoras tomando en cuenta las capacitancias parásitas de los pines y las inductancias de los cables en los paquetes.
- Líneas de transmisión o *interconnetcs*.
- Celdas receptoras con capacitancias parásitas en los pines e inductancias de cable de paquete.
- Terminaciones u otros componentes eléctricos en las líneas de transmisión.

Según el manual de *PrimeSim HSPICE* [19], los problemas más comunes de integridad de señal que se presentan en diseños de alta velocidad se encuentran en el Cuadro 9.

Problema	**Causas**	**Solución**
Noise: delta I (current)	Múltiples drives cambiando de estado simultáneamente, dispositivos de alta velocidad generan cambios grandes de corriente.	Ajustar o evaluar el tamaño, posición y valor de los capacitores de desacople.
Noise: coupled (crosstalk)	Tramos paralelos demasiado cerca.	Establecer reglas de diseño para longitudes de líneas paralelas.
Noise: reflective	Discordancia de impedancias.	Reducir el número de conectores y seleccionar mejores conectores correctos de impedancia.
Delay: path length	Colocación y enrutamiento deficientes, demasiadas o pocas capas.	Escoger una tecnología de empaquetado de alta densidad.
Propagation speed	Medio dieléctrico.	Escoger un dieléctrico con menor constante dieléctrica.
Delay: rise time degradation	Pérdida de resistividad y discordancia de impedancias.	Ajustar ancho, grosor y longitud de las líneas de transmisión.

Cuadro 9: Errores comunes de integridad de señal en circuitos de alta velocidad. *HSPICE*

La simulación de un circuito de un sistema digital se vuelve necesaria cuando características analógicas de las señales del circuito se tornan relevantes. Para obtener una correcta

simulación de la integridad de este tipo de señales sobre todo en circuitos que operan a alta frecuencia, se debe analizar cuidadosamente el ruido y los delays de las señales. El ruido tiene varias fuentes; sin embargo, las más relevantes son:

- Ruido por terminación de una línea o interconnect
- Ruido por rebote en el nodo de tierra generado cuando los cables del circuito no pueden formarse en líneas de transmisión.
- Ruido de líneas de acople formado entre líneas que son están adyacentes.

Por otra parte, exceder la capacidad de ruido establecida para un circuito puede que no cause que este falle. Alcanzar este valor máximo solamente se vuelve problemático cuando *PrimeSim HSPICE* acepta una entrada digital. Por consiguiente, si se logra desacoplar el sistema, el simulador puede aceptar un nivel mayor de ruido. Los métodos más comunes para lograr esto son:

- Tener múltiples nodos de tierra y planos de alimentación en el *printed circuit board* (PCB), *multichip module* (MCM), and *pin grid array* (PGA).
- Capacitores de desacoplamiento.
- Separación de las líneas de señal por medio de líneas de tierra.
- Resistencias en serie a los buffers de salida.
- Conducción de línea de par trenzado.

A medida que aumenta la frecuencia de operación de un chip, las longitudes de las líneas o interconnects se vuelve un factor crítico para analizar la integridad de las señales. Cuando esto sucede, se debe tomar en cuenta dos factores. El primero es la velocidad de la señal en un circuito impreso, la cual es de 6 in/ns aproximadamente. Segundo, los sistemas críticos de alambrado, los cuales son:

- Nivel de alambrado del circuito integrado.
- Alambrado de la placa impresa del circuito.
- Líneas de alimentación, coaxial o par trenzado.

Para estimar la longitud de un cable o interconnect cuando la línea de transmisión deja de ser despreciable, se puede usar la siguiente ecuación:

$$D_c = (t_r) * \frac{v}{8} \tag{1}$$

Donde D_c corresponde a la distancia crítica, t_r representa el tiempo de subida de la señal y v es la velocidad de la señal en el medio.

6.3.1. .LIN análisis

El comando .LIN extrae el ruido y los parámetros de transferencia lineales para una red general multipuerto. Cuando se utiliza junto con el comando .AC se habilitan un conjunto de mediciones lineales por puerto. Estas son:

- Parámetros de dispersión multipuerto [S]
- Parámetros de ruido
- Factores de estabilidad
- Factores de ganancia
- Coeficientes de coincidencia

El análisis .LIN es similar a varios análisis AC provistos por *PrimeSim HSPICE* pero permite calcular automáticamente una serie de parámetros de señal pequeña y transferencia de ruido entre las terminales identificados mediante elementos de puerto (P).

6.3.2. .SNLIN análisis

El comando .SNLIN invoca el análisis SNLIN de *PrimeSim HSPICE*. Este tiene dos principales funciones. La primera es la extracción de parámetros de ruido de un circuito accionado periódicamente que presenta efectos de traducción de frecuencia. La segunda es la extracción de parámetros de dispersión de traducción de frecuencia en un circuito accionado periódicamente que exhibe efectos de traducción de frecuencia, como mezcladores.

6.4. Sintaxis *HSPICE*

6.4.1. Definición de subcircuitos

Para definir un subcircuito en el *netlist* se utiliza el comando .SUBCKT. Se recomienda usar esta opción cuando se tiene la descripción de un circuito común que se está utilizando varias veces en el *netlist*. La forma de utilizar este comando es la siguiente:

```
.SUBCKT subnam n1 n2 n3 ... [param=val]
.ENDS
```

Donde las variables usadas tienes los siguientes significados:

- *subnam*: Nombre de referencia de un modelo de subcircuito.
- *n1..* : Nombre de nodos externos al subcircuito, no se puede usar el nodo de tierra (0, gnd, ground, gnd!); ni nodos que fueron asignados usando el comando .GLOBAL

- *param*: Nombre del parámetro utilizado en el subcircuito.

Es importante que al final de un subcircuito se utilice el comando .ENDS para denotar la finalización del subcircuito y evitar errores de simulación.

6.4.2. Mediciones

En una gran variedad de análisis se vuelve útil poder modificar en tiempo real parámetros del análisis u obtener información para definir resultados de simulaciones sucesivas en *PrimeSim HSPICE*. Por tanto, el comando .MEASURE permite imprimir especificaciones eléctricas de un circuito definidas por el usuario. Estas especificaciones pueden ser:

- Delay
- Propagación
- Tiempo de subida de una señal
- Tiempo de bajada de una señal
- Voltaje pico a pico
- Voltaje mínimo y máximo dentro de un período definido.

También se puede utilizar el comando .MEASURE con la función error o el parámetro GOAL para optimizar los parámetros de los componentes del circuito. Dependiendo de la aplicación y tipo de simulación que se esté realizando, se pueden utilizar diferentes formatos [10].

Tipo de MEASURE	Análisis soportados
.MEASURE (Potencia, delays, tiempo de subida y bajada)	AC, DC, TRAN, SN, SNOSC, SNAC, SNFD, HB, HBOSC, HBTRAN (o HBTR), y HBAC
.MEASURE (FIND y WHEN)	AC, DC, TRAN, HB, HBOSC, HBAC, HBNOISE, HBTRAN, HBXF, SN, SNOSC, SNFD, SNAC, SNNOISE, SNXF, ENV, y ENVOSC
.MEASURE (Resultados Continuos)	DC_CONT, AC_CONT, y TRAN_CONT
.MEASURE (Evaluación de ecuación/ expresión aritmética)	AC, DC, TRAN, HB, HBOSC, HBTRAN, HBAC, SN, SNOSC, SNFD, y SNAC
.MEASURE (AVG, EM_AVG, INTEG, MIN, MAX, PP, y RMS)	AC, DC, TRAN, HB, HBOSC, HBAC, HBTRAN, HBAC, SN, SNOSC, SNFD, SNAC, SNXF, ENV, y ENVOSC
.MEASURE (Ventana de mediciones múltiple para máximos y mínimos)	AC, DC, y TRAN
.MEASURE (Función Integral)	AC, DC, TRAN, SN, SNOSC, SNFD, SNAC, HBNOISE, y SNNOISE
.MEASURE (Función derivada)	AC, DC, TRAN, SN, SNOSC, SNFD, SNAC, SNNOISE, HB, HBOSC, HBTRAN (o HBTR), HBAC, y HBNOISE
.MEASURE (Función error)	AC, DC, TRAN, SN, SNOSC, SNFD, SNAC, HB, HBOSC, HBTRAN (o HBTR), y HBAC

Cuadro 10: Análisis soportados según el tipo de .MEASURE que se utilice

.MEASURE (Potencia, delays, tiempo de subida y bajada)

Por medio de este se puede medir cambios en variables independientes del circuito, como el tiempo de subida, tiempo de bajada, entre otros. La sintaxis de este formato es la siguiente:

```
.MEASURE [AnalysisType] ResultName
+ TRIG TrigSpec TARG [FROM=val] [TO=val]TargSpec
+ [GOAL=val][MINVAL=val][WEIGHT=val][PRINT= 0|1][FROM=val]
+ [TO=val]
```

En donde, los significados de las variables usadas se detallan en el Cuadro 11

Argumento	Descripción
AnalysisType	Tipo de análisis realizado
ResultName	Nombre asociado al valor medido
TRIG	Inicio de las especificaciones de trigger (instante de inicio de la medición)
TARG	Inicio de las especificaciones del target (instante de finalización de la medición)
TrigSpec	Especificación del valor y elemento que disparará el trigger.
TargSpec	Especificación del valor y elemento que representa el target de la medición
GOAL=val	Valor de medida deseado en el cálculo de la función error para optimización
MINVAL	Valor mínimo que puede tener el parámetro GOAL
WEIGHT	Función de peso para el error obtenido
PRINT	Si: •print=0 no se imprime el resultado de la medición en el archivo de salida. •print=1 (Por defecto) imprime el resultado de la medición en el archivo de salida
FROM... TO...	Permite añadir condiciones XRANGE.
trig_ var	Valor de incremento de contador.
trig_ var	Especifica el nombre de la variable de entrada que determina el tiempo de inicio de medición.
PREVIOUS	Al usar la opción PREVIOUS, el último evento de target previo al evento de trigger se computa.
REVERSE	La opción REVERSE se usa para revertir la dirección de la medición.
TD	Tiempo de simulación que debe pasar antes que PrimeSim habilite la medición.
AT=val	Caso especial de especificación de trigger. val es: •Time para análisis TRAN. •Frequency para análisis AC. •Parameter para análisis DC. •SweepValue análisis de discordancia DC.

Cuadro 11: Explicación de las variables del comando .MEASURE (Potencia, delays, tiempo de subida y bajada)

.MEASURE (FIND y WHEN)

Este tipo de comando .MEASURE puede medir variables dependientes e independientes. De igual modo, variables dependientes al ocurrir un evento específico. Este comando tiene 2 posibles sintaxis dependiende su función; una utilizado la opción WHEN y otra con la opción FIND. Estas son las siguientes:

```
.MEASURE [AnalysisType] result WHEN out_var=val [TD=val]
```

```
+ [FROM=val] [TO=val]
+ [RISE= r|LAST][FALL= f|LAST][CROSS= c|LAST][REVERSE]
+ [[GOAL=val]|GOALMAX|GOALMIN] [MINVAL=val] [WEIGHT=val]
+ [PRINT= 0|1]

.MEASURE [AnalysisType] result FIND out_var1
+ WHEN out_var2=val [TD=val] [FROM=val] [TO=val]
+ [RISE= r|LAST][FALL= f|LAST] [CROSS= c|LAST] [REVERSE]
+ [[GOAL=val]|GOALMAX|GOALMIN][MINVAL=val] [WEIGHT=val]
+ [PRINT= 0|1]
```

Donde las definiciones de las variables se muestran en el Cuadro 12

Argumento	Descripción
AnalysisType	Tipo de análisis realizado
result	Nombre asociado al valor medido
WHEN	Función WHEN
FIND	Función FIND
out_ var1, out_ var2	Variables que establecen condiciones para iniciar la medición.
CROSS=c *RISE=r* *FALL=f*	Indica la cantidad de flancos de subida, bajada deben suceder para realizar la medición.
GOAL=val	Valor de medida deseado en el cálculo de la función error para optimización
MINVAL	Valor mínimo que puede tener el parámetro GOAL
WEIGHT	Función de peso para el error obtenido
PRINT	Si: •print=0 no se imprime el resultado de la medición en el archivo de salida. •print=1 (Por defecto) imprime el resultado de la medición en el archivo de salida
FROM... TO...	Permite añadir múltiples condiciones de trigger a mediciones utilizando la función WHEN.
GOALMAX/GOALMIN	Utiliza el método de bisección para obtener el método máximo y mínimo de la medición.
LAST	Palabra reservada para determinar que la medición se realice después del último evento.
PREVIOUS	Al usar la opción PREVIOUS, el último evento de target previo al evento de trigger se computa.
REVERSE	La opción REVERSE se usa para revertir la dirección de la medición.
TD	Tiempo de simulación que debe pasar antes que PrimeSim habilite la medición.
AT=val	Caso especial de especificación de trigger. val es: •Time para análisis TRAN. •Frequency para análisis AC. •Parameter para análisis DC. •SweepValue análisis de discordancia DC.

Cuadro 12: Explicación de las variables del comando .MEASURE (FIND y WHEN)

.MEASURE (Error Function)

La función de error relativo brinda un reporte de la diferencia relativa entre dos variables de salida. Se puede utilizar este formato en optimización y ajustes de curvas de datos muestreados. Para calcular el error relativo *PrimeSim HSPICE* utiliza las funciones ERR, ERR1, ERR2 y ERR3. Con este formato se puede especificar un grupo de parámetros que varíen hasta que coincidan con los valores calculados y los datos medidos. La sintaxis del comando es la siguiente:

```
.MEASURE [AnalysisType] result
+ ERRfun meas_var calc_var
+ [MINVAL=val] [IGNOR|YMIN=val]
+ [YMAX=val] [WEIGHT=val] [FROM=val] [TO=val] [PRINT= 0|1]
```

Donde el significado de los parámetros se encuentran en el Cuadro 13.

Argumento	**Descripción**
AnalysisType	Tipo de análisis realizado
result	Nombre asociado al valor medido
ERRfun	Función error a usar: ERR, ERR1, ERR2, o ERR3.
meas_var	Nombre de cualquier variable de salida o parámetro en el comando de datos.
calc_var	Nombre de la variable de salida simulada o parámetros en el comando para comparar con meas_var.
IGNOR/YMIN	Si el valor absoluto de meas_var es menor que IGNOR, entonces el cálculo de ERRfun no se considera en este punto.
YMAX	Si el valor absoluto de meas_var es mayor que YMAX, entonces el cálculo de ERRfun no se considera en este punto.
MINVAL	entonces el cálculo de ERRfun no se considera en este punto.
WEIGHT	Función de peso para el error obtenido
PRINT	Si: •print=0 no se imprime el resultado de la medición en el archivo de salida. •print=1 (Por defecto) imprime el resultado de la medición en el archivo de salida
FROM... TO...	Permite añadir múltiples condiciones de trigger a mediciones utilizando la función WHEN.

Cuadro 13: Explicación de las variables del comando .MEASURE (Error Function)

.MEASURE (Ventana de mediciones múltiple para máximos y mínimos)

La ventaja de este tipo de .MEASURE sobre los demás es que permite especificar múltiples ventanas de mediciones. La sintaxis del comando es la siguiente:

```
.MEASURE [AnalysisType]
+ measure_result MAX|MIN
+ measure_variable XRANGE=(xlow1 xhigh1
+                          xlow2 xhigh2
+                          ...
+                          xlown xhighn)
```

Donde el significado de los parámetros se encuentran en el Cuadro 14.

Argumento	Descripción
AnalysisType	Tipo de análisis realizado
measure_result	Nombre asociado al valor medido
ERRfun	Función error a usar: ERR, ERR1, ERR2, o ERR3.
measure_variable	Nombre de cualquier variable de salida o parámetro en el comando de datos.
XRANGE	Partición explícita de un intervalo de medición en subintervalos.
xlow1, xhigh1...	Parejas para límite superior e inferior para subintervalos válidos.

Cuadro 14: Explicación de las variables del comando .MEASURE (Ventana de mediciones múltiple para máximos y mínimos)

6.4.3. Sentencia IF

La sentencia if le permite saber a *PrimeSim HSPICE* si debe ejecutar una serie de comandos agrupados en un bloque condicional. La sintaxis para utilizar esta herramienta es:

```
.IF (condicion1)
  (instrucciones1)
.ELSEIF (condicion2)
  (instrucciones2)
.ELSE
  (instrucciones3)
.ENDIF
```

Operadores relacionales

Para definir las condiciones se pueden utilizar operadores relacionales. Los que son soportados por la herramienta *PrimeSim HSPICE* son los siguientes:

- == corresponde a igual que
- < corresponde a menor que
- <= corresponde a menor o igual que
- > corresponde a mayor que
- >= corresponde a mayor o igual que
- && corresponde a un AND lógico
- || corresponde a un OR lógico

6.4.4. .OPTION

Por medio del comando .OPTION se pueden cambiar los aspectos de una simulación en *PrimeSim HSPICE*; incluyendo aspectos como: tipos de salida, exactitud, velocidad y convergencia. Se puede definir un amplio número de opciones en un único comando .OPTION; y también se pueden utilizar varios comando .OPTION en un mismo archivo de *netlist*. La mayoría de opciones para este comando tiene 0 como valor predeterminado; a excepción de aquellas opciones cuyos valores correspondan a una tolerancia o parámetros de análisis. La sintaxis para utilizar este comando es:

```
.OPTION opt=val
```

Donde *opt* corresponde al nombre de la opción y *val* representa el valor numérico asignado a la opción. Para obtener un listado sobre los tipos de opciones que son aceptados por *PrimeSim* se puede consultar el manual de usuario en [19].

CAPÍTULO 7

Construcción de Decks

En esta sección se muestran los resultados de simulación obtenidos, mediante *PrimeSim* HSPICE y *WaveView*, de los decks que fueron realizados con el objetivo de simular el archivo spice de extracción de parásitos. El código en HSPICE de los decks realizados se puede encontrar en la sección de anexos 12.1.

Para obtener los resultados, se diseñó un deck en HSPICE 12.1 que cuyo objetivo fuera configurar un análisis transitorio que permitiera observar como variaba la salida de los decks, según el cambio de señal en sus entradas. Por otra parte, para simular el comportamiento de los transistores, se usaron las librerías para tecnología CMOS de 180 nm desarrolladas por la Universidad Estatal de Arizona (ASU) que se encuentran en [20].

7.1. Decks realizados

7.1.1. IAO21D0BWP7T

Tabla de verdad			
Entradas			Salida
A1	A2	B	ZN
0	0	x	0
x	x	1	0
1	x	0	1
x	1	0	1

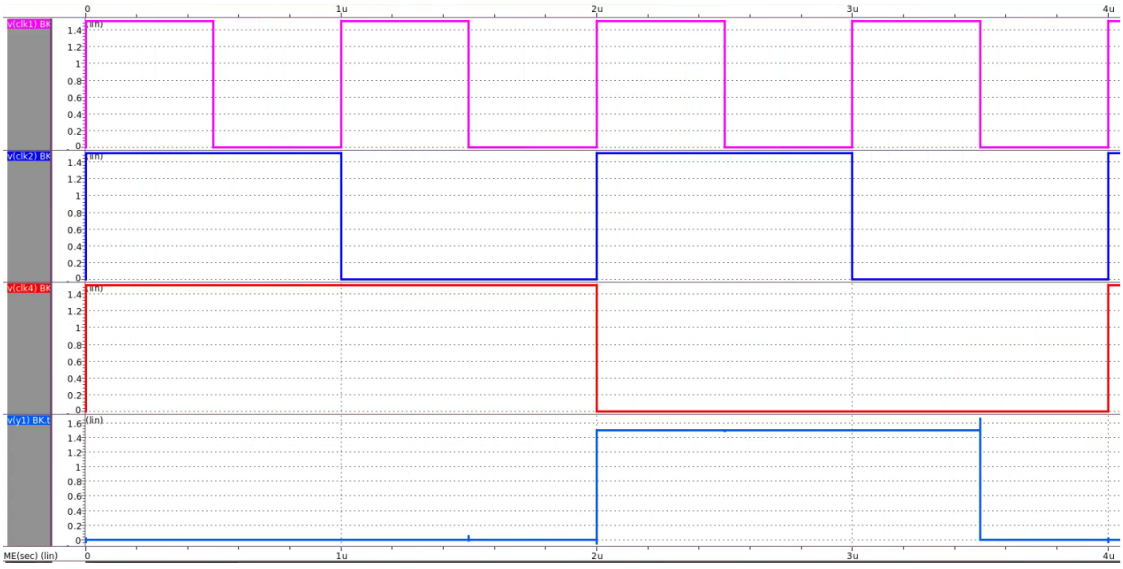


Figura 5: Resultados de la simulación del circuito IAO21D0BWP7T realizado en HSPICE.

7.1.2. AOI211D1BWP7T

Tabla de verdad				
Entradas				Salida
A1	A2	B	C	ZN
1	1	x	x	0
x	x	1	x	0
x	x	x	1	0
0	x	0	0	1
x	0	0	0	1

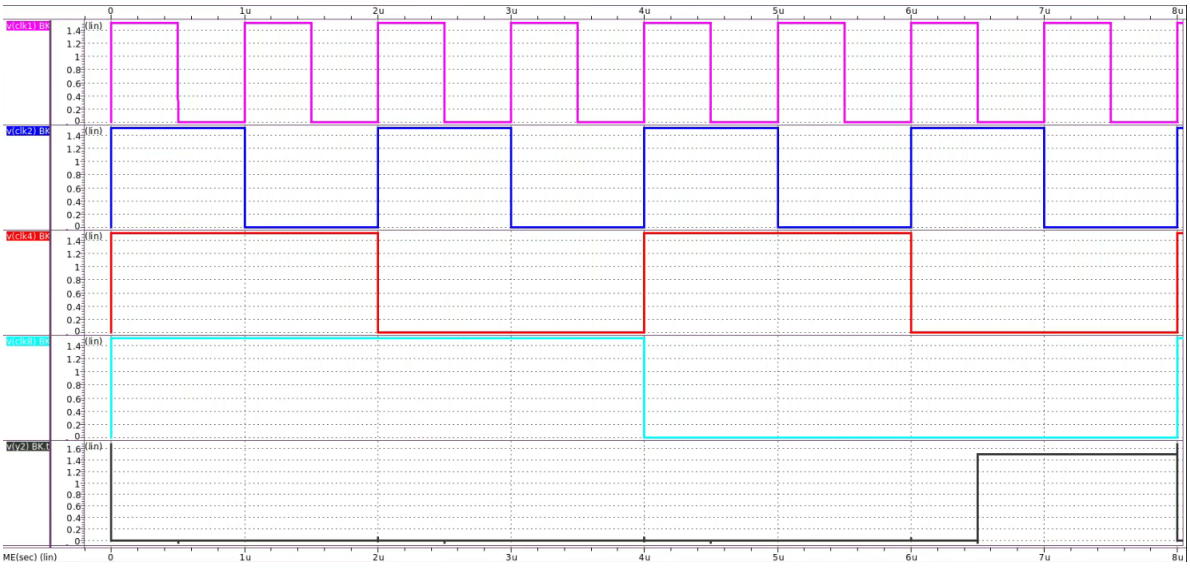


Figura 6: Resultados de la simulación del circuito AOI211D1BWP7T realizado en HSPICE.

7.1.3. AN2XD1BWP7T

Tabla de verdad		
Entradas		Salida
A1	A2	ZN
1	1	1
0	x	0
x	0	0

Figura 7: Resultados de la simulación del circuito AN2XD1BWP7T realizado en HSPICE.

7.1.4. DFCND0BWP7T

Tabla de verdad				
Entradas			Salidas	
CDN	CP	D	Q	QN
0	x	x	0	1
1	FP	0	0	1
1	FP	1	1	0
1	0	x	Q	QN
1	1	x	Q	QN

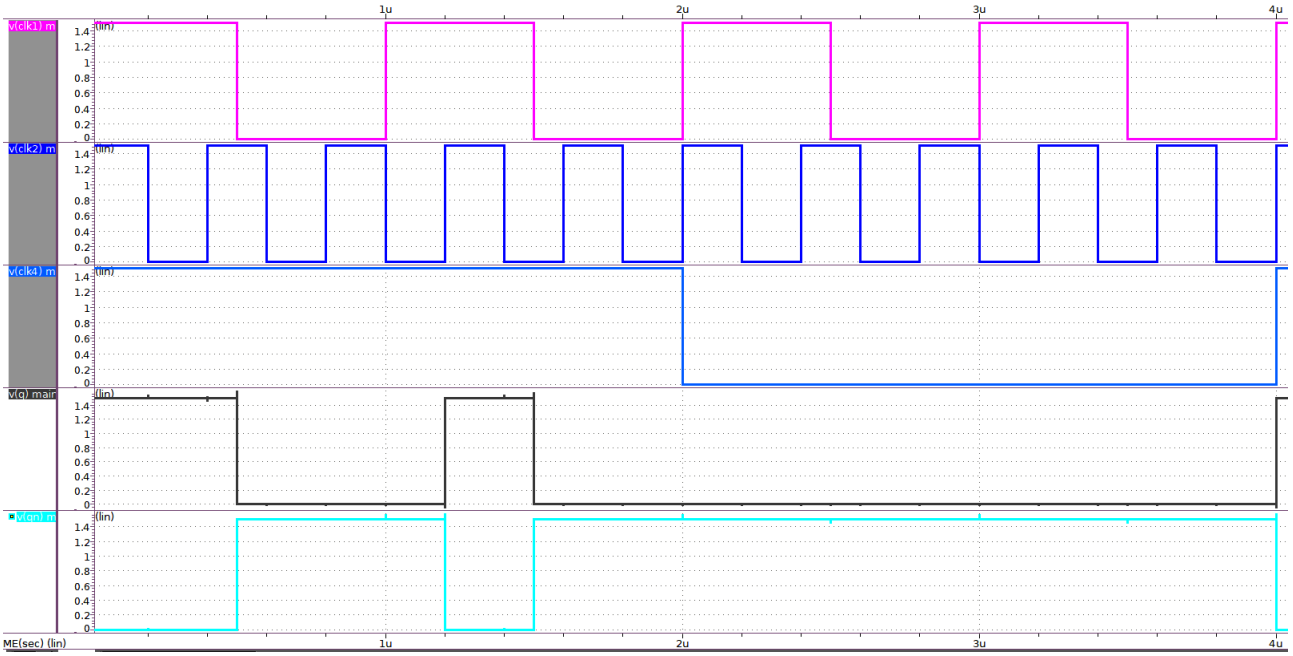


Figura 8: Resultados de la simulación del circuito DFCND0BWP7T realizado en HSPICE.

7.1.5. AOI31D0BWP7T

Tabla de verdad				
Entradas				Salida
A1	A2	A3	B	ZN
1	1	1	x	0
x	x	x	1	0
0	x	x	0	1
x	0	x	0	1
x	x	0	0	1

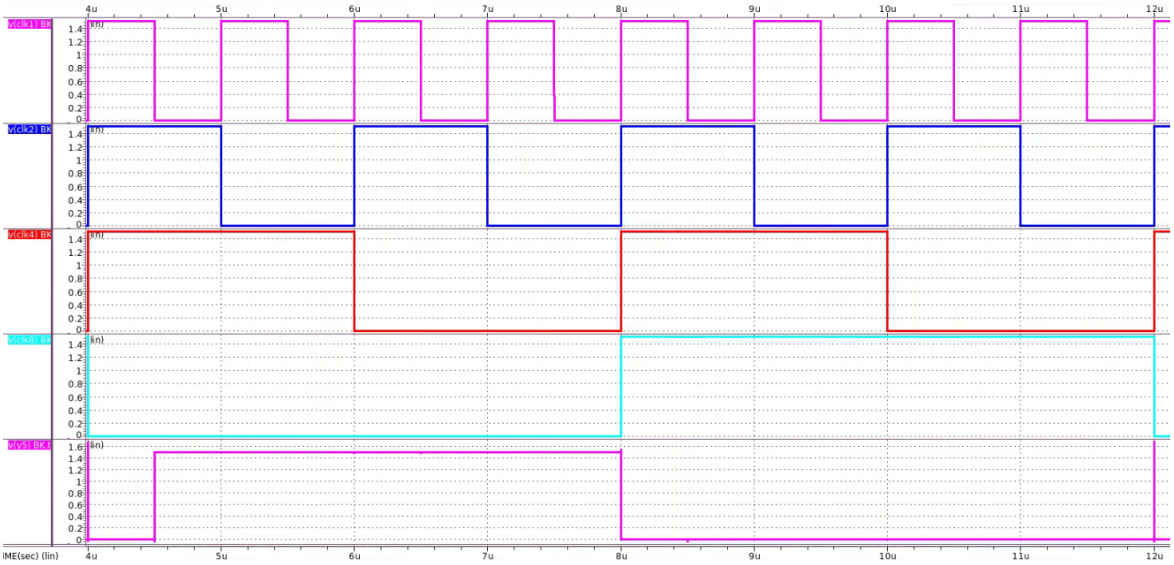


Figura 9: Resultados de la simulación del circuito AOI31D0BWP7T realizado en HSPICE.

7.1.6. AO211D0BWP7T

Tabla de verdad				
Entradas				Salida
A1	A2	B	C	ZN
1	1	x	x	1
x	x	1	x	1
x	x	x	1	1
0	x	0	0	0
x	0	0	0	0

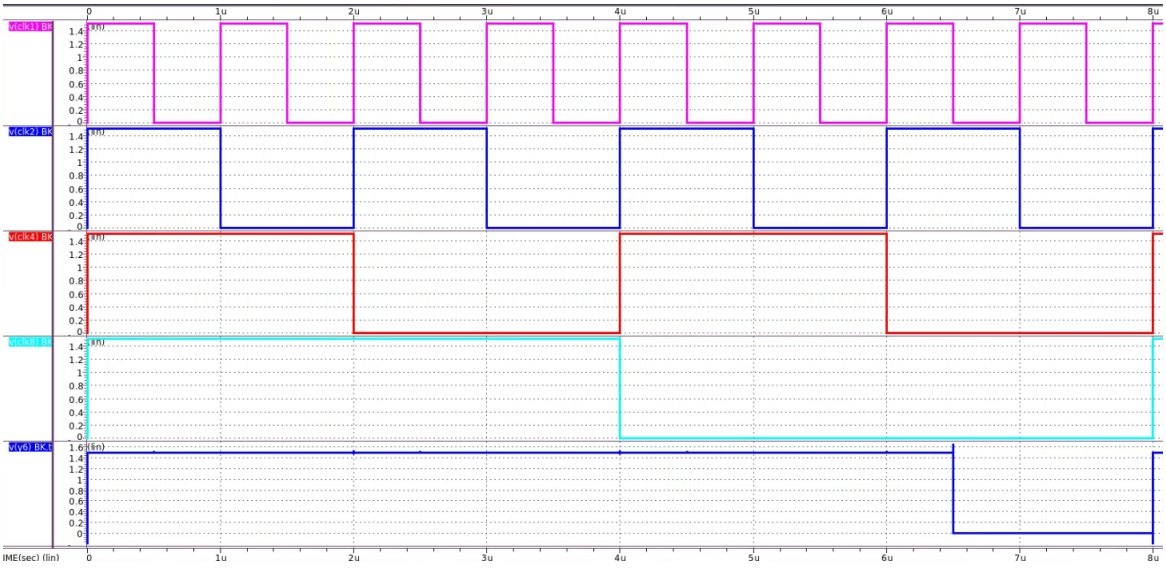


Figura 10: Resultados de la simulación del circuito AO211D0BWP7T realizado en HSPICE.

7.1.7. AOI22D0BWP7T

Tabla de verdad				
Entradas				Salida
A1	A2	B1	B2	ZN
1	1	x	x	0
x	x	1	1	0
0	x	x	0	1
x	0	x	0	1
0	x	0	x	1
x	0	0	x	1

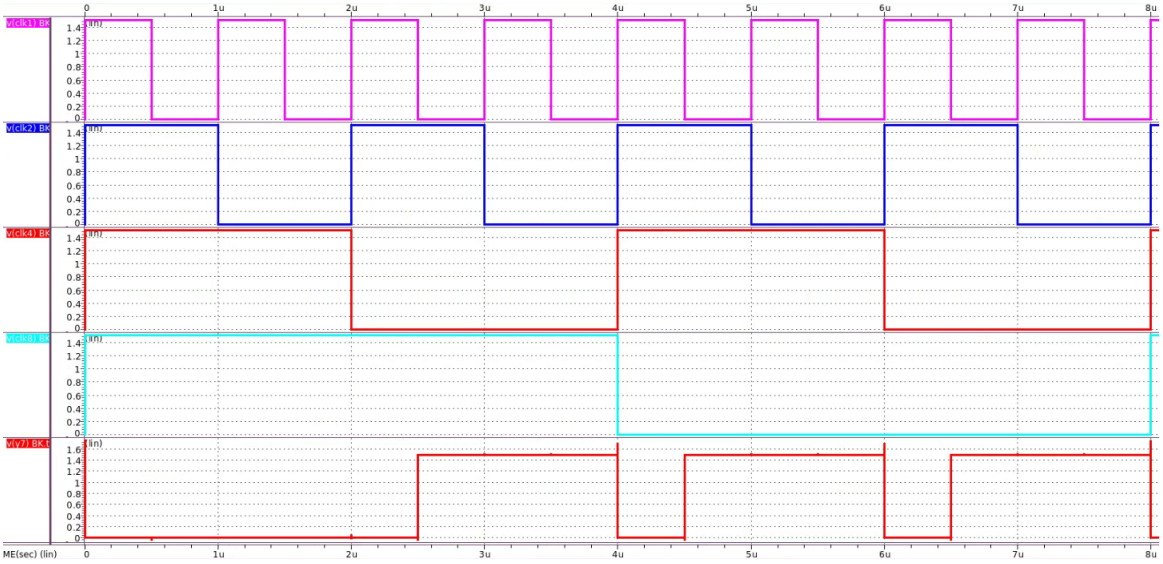


Figura 11: Resultados de la simulación del circuito AOI22D0BWP7T realizado en HSPICE.

7.1.8. TIEHBWP7T

Salida
Z
1

Se tomó la decisión de no crear este *deck* por el hecho que la salida siempre debe estar conectada a un 1 lógico, por lo que la conexión se realizará de forma manual para mayor simplicidad al momento de hacer la conexión de los *decks* con los *blackboxes* generados en la síntesis física.

7.1.9. AO222D0BWP7T

Tabla de verdad						
Entradas						Salida
A1	A2	B1	B2	C1	C2	Z
1	1	x	x	x	x	1
x	x	1	1	x	x	1
x	x	x	x	1	1	1
0	x	x	0	x	0	0
x	0	x	0	x	0	0
0	x	0	x	x	0	0
x	0	0	x	x	0	0
0	x	x	0	0	x	0
x	0	x	0	0	x	0
0	x	0	x	0	x	0
x	0	0	x	0	x	0

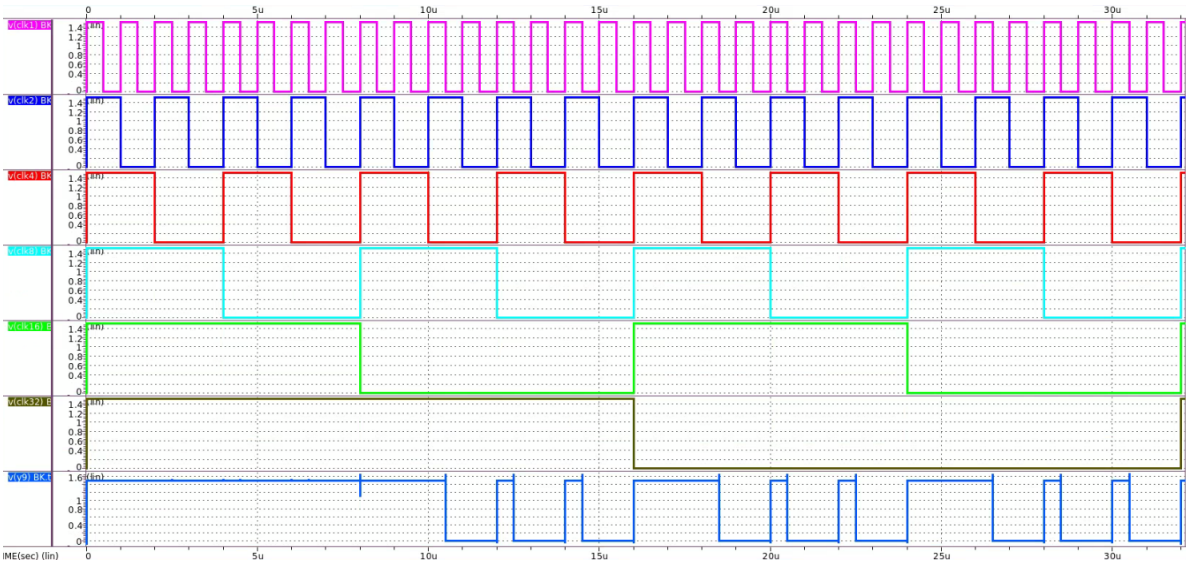


Figura 12: Resultados de la simulación del circuito AO222D0BWP7T realizado en HSPICE.

7.1.10. OA211D0BWP7T

Tabla de verdad				
Entradas				Salida
A1	A2	B	C	Z
0	0	x	x	0
x	x	0	x	0
x	x	x	0	0
1	x	1	1	1
x	1	1	1	1

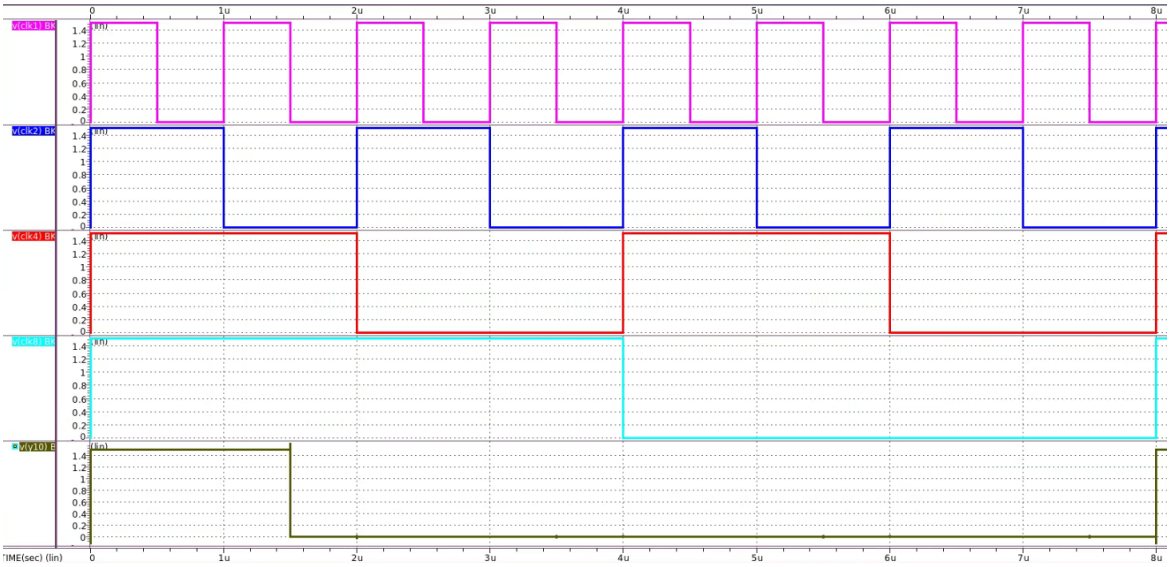


Figura 13: Resultados de la simulación del circuito OA211D0BWP7T realizado en HSPICE.

7.1.11. NR2D1BWP7T

Tabla de verdad		
Entradas		Salida
A1	A2	ZN
0	0	1
1	0	0
0	1	0
1	1	1

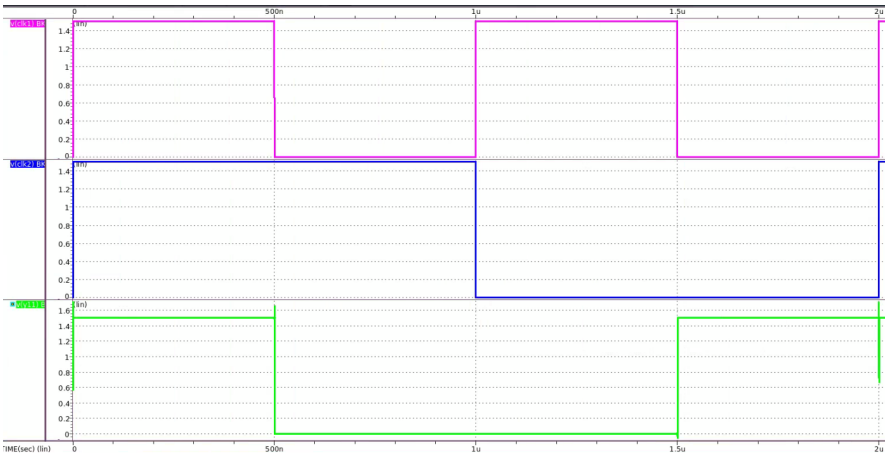


Figura 14: Resultados de la simulación del circuito NR2D1BWP7T realizado en HSPICE.

7.1.12. MOAI22D0BWP7T

Tabla de verdad				
Entradas				Salida
A1	A2	B1	B2	ZN
0	0	x	x	1
x	x	1	1	1
1	x	x	0	0
x	1	x	0	0
1	x	0	x	0
x	1	0	x	0

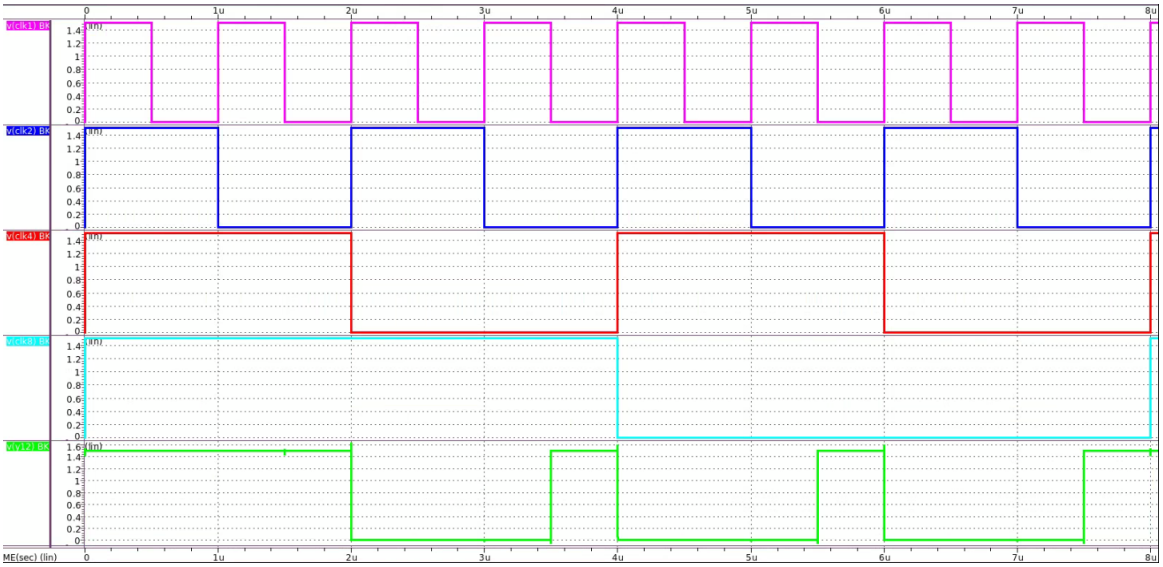


Figura 15: Resultados de la simulación del circuito MOAI22D0BWP7T realizado en HSPICE.

7.1.13. IINR4D0BWP7T

Tabla de verdad				
Entradas				Salida
A1	A2	B1	B2	ZN
1	1	0	0	1
0	x	x	x	0
x	0	x	x	0
x	x	1	x	0
x	x	x	1	0

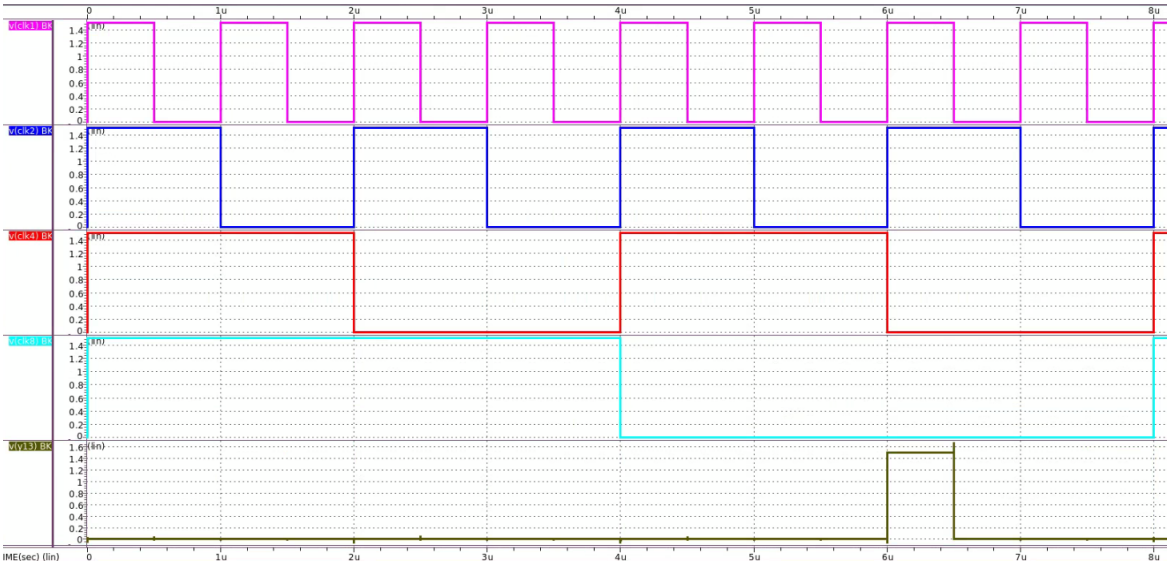


Figura 16: Resultados de la simulación del circuito IINR4D0BWP7T realizado en HSPICE.

7.1.14. IND2D1BWP7T

Tabla de verdad		
Entradas		Salida
A1	B1	ZN
0	1	0
1	x	1
x	0	1

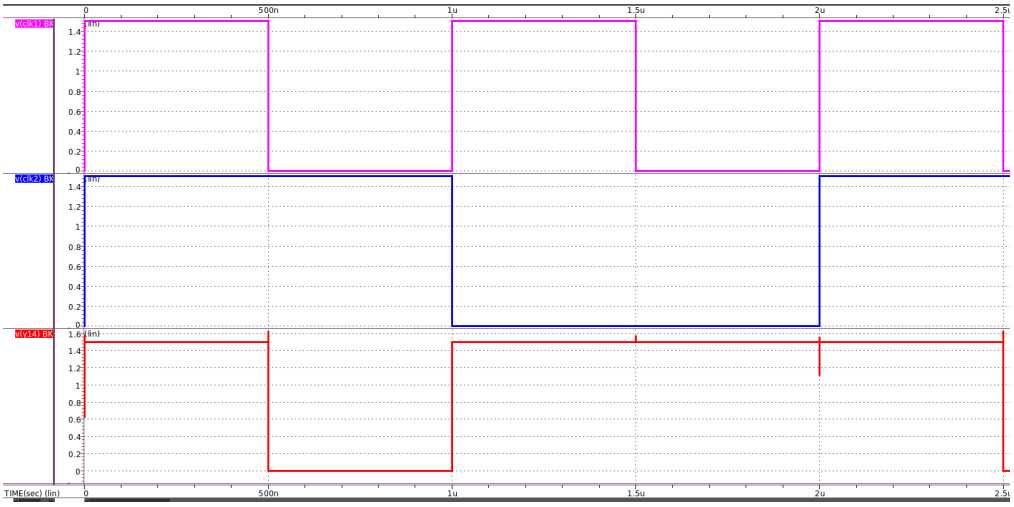


Figura 17: Resultados de la simulación del circuito IND2D1BWP7T realizado en HSPICE.

7.1.15. AN3D1BWP7T

Tabla de verdad			
Entradas			Salida
A1	A2	A3	ZN
1	1	1	1
0	x	x	0
x	0	x	0
x	x	0	0

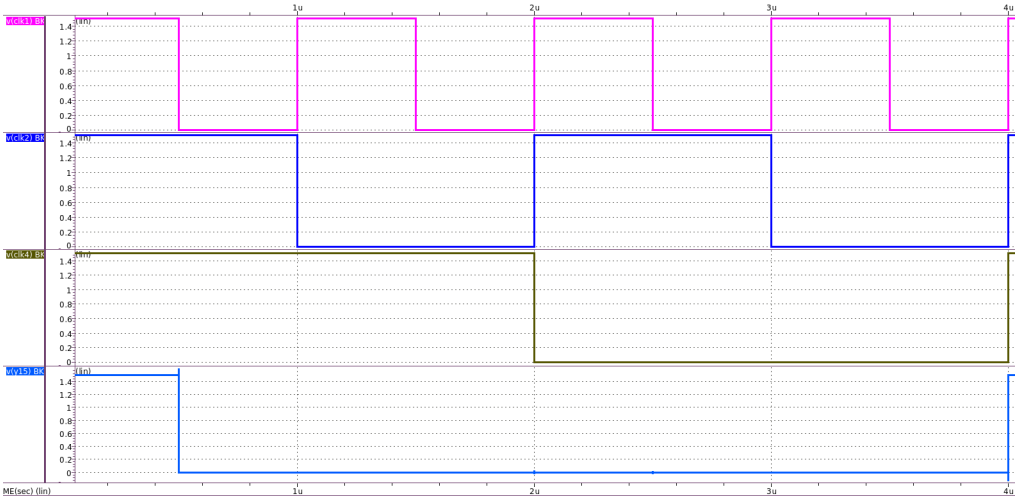


Figura 18: Resultados de la simulación del circuito AN3D1BWP7T realizado en HSPICE.

7.1.16. OR2D1BWP7T

Tabla de verdad		
Entradas		Salida
A1	A2	ZN
0	0	0
1	x	1
x	1	1

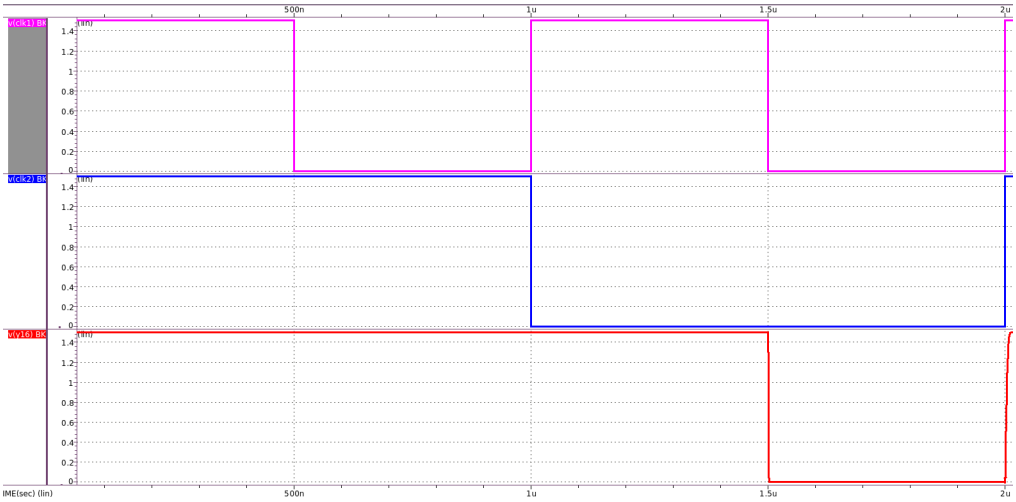


Figura 19: Resultados de la simulación del circuito OR2D1BWP7T realizado en HSPICE.

7.1.17. HA1D0BWP7T

Tabla de verdad		
Entradas		Salida
A1	B1	ZN
0	1	0
1	x	1
x	0	1

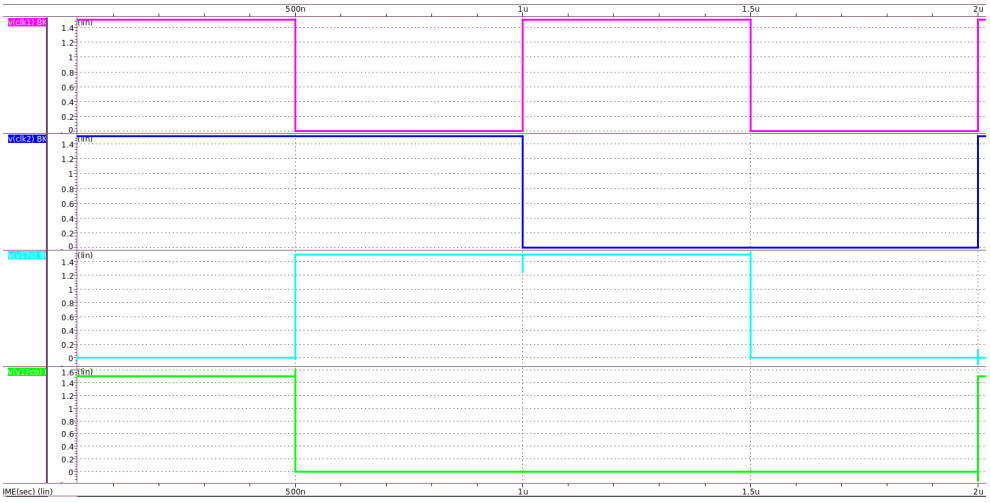


Figura 20: Resultados de la simulación del circuito HA1D0BWP7T realizado en HSPICE.

7.1.18. CKXOR2D0BWP7T

Tabla de verdad		
Entradas		Salida
A1	B1	ZN
0	1	0
1	x	1
x	0	1

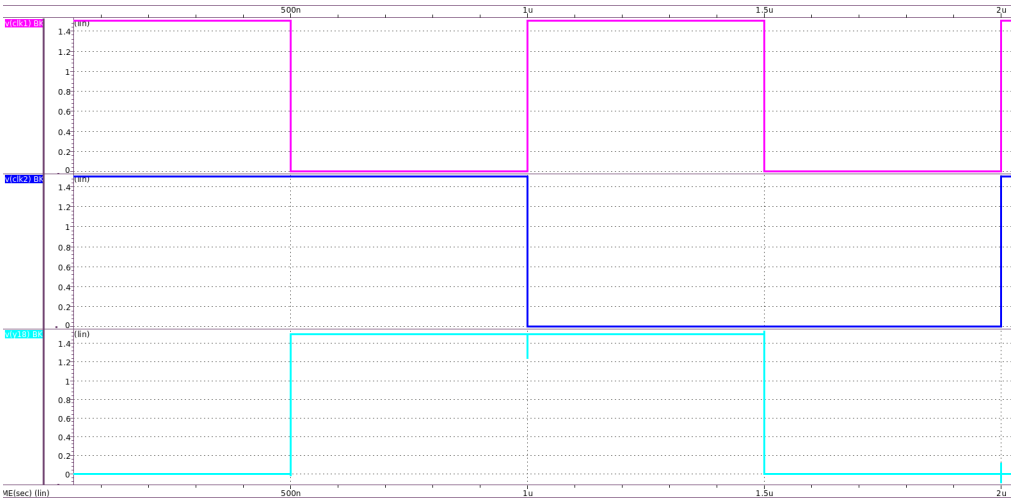


Figura 21: Resultados de la simulación del circuito CKXOR2D0BWP7T realizado en HSPICE.

7.1.19. OAI211D0BWP7T

Tabla de verdad		
Entradas		Salida
A1	B1	ZN
0	1	0
1	x	1
x	0	1

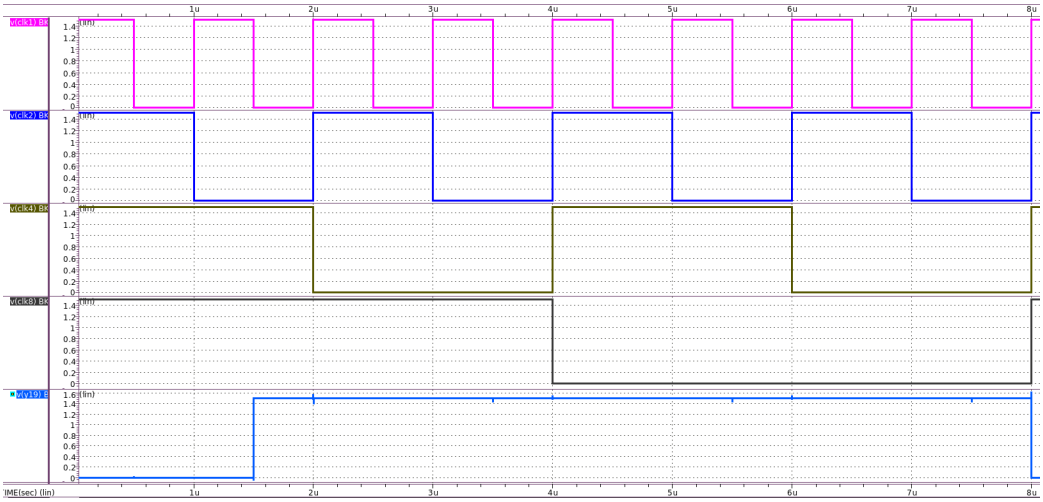


Figura 22: Resultados de la simulación del circuito OAI211D0BWP7T realizado en HSPICE.

7.1.20. OA31D0BWP7T

Tabla de verdad		
Entradas		Salida
A1	B1	ZN
0	1	0
1	x	1
x	0	1

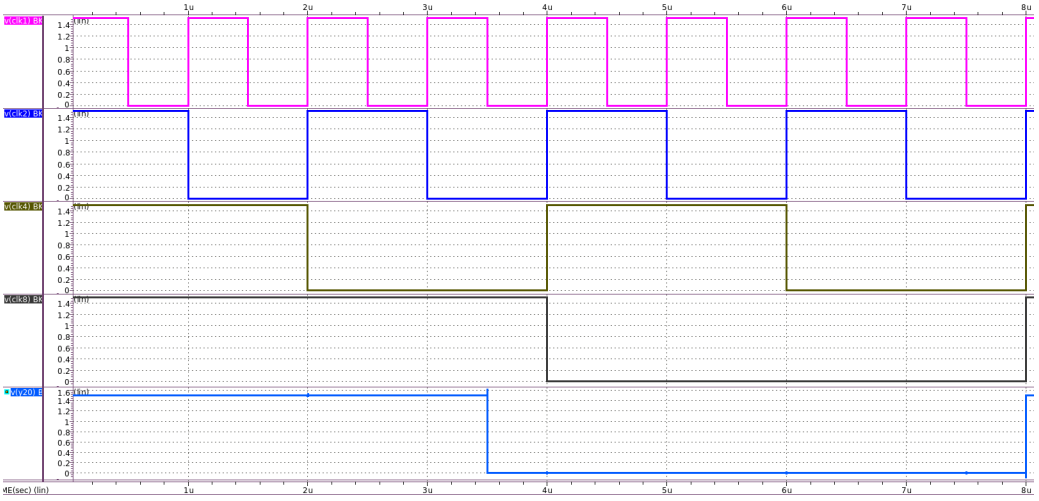


Figura 23: Resultados de la simulación del circuito OA31D0BWP7T realizado en HSPICE.

7.1.21. AN4D0BWP7T

Tabla de verdad		
Entradas		Salida
A1	B1	ZN
0	1	0
1	x	1
x	0	1

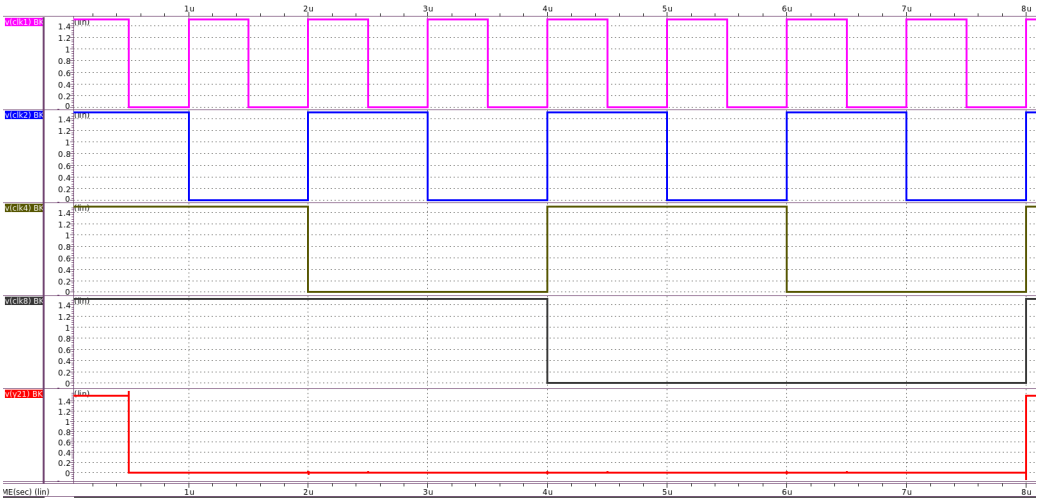


Figura 24: Resultados de la simulación del circuito AN4D0BWP7T realizado en HSPICE.

7.1.22. AO22D0BWP7T

Tabla de verdad		
Entradas		Salida
A1	B1	ZN
0	1	0
1	x	1
x	0	1

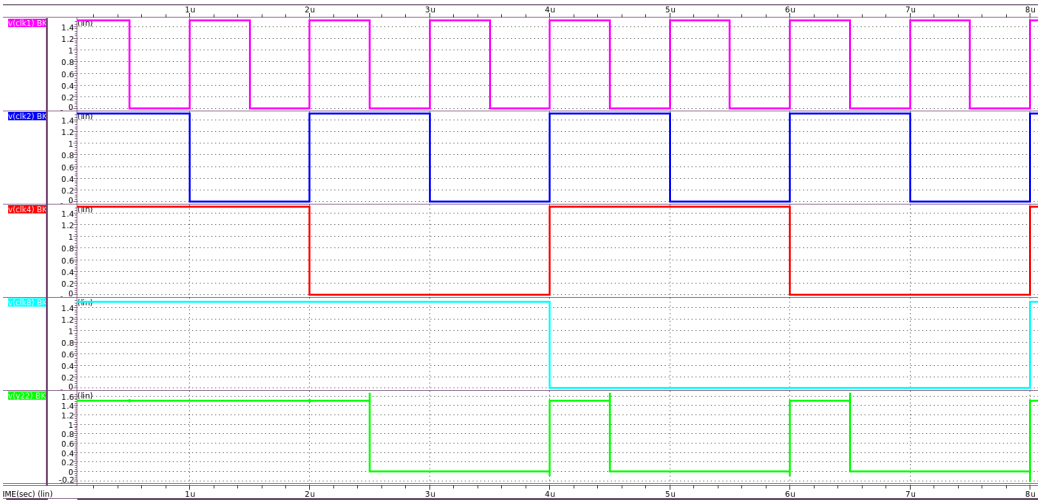


Figura 25: Resultados de la simulación del circuito AO22D0BWP7T realizado en HSPICE.

CAPÍTULO 8

Pruebas en PrimeSim

8.1. Manual de inicio rápido

8.1.1. Análisis DC

Para realizar este análisis se diseñó un deck en HSPICE para estudiar como el voltaje de entrada de una compuerta NOT afecta el voltaje de salida de esta misma. El Deck utilizado es el siguiente:

```
*CIRCUITO INVERSOR

*Importacion de los modelos de MOSFET
* El LEVEL=1 representa a un transistor hecho a partir de los modelos de
* Schichman-Hodges
.MODEL PCH PMOS LEVEL=1
.MODEL NCH NMOS LEVEL=1

*Construccion del NOT CMOS
M1 VCC IN OUT VCC PCH L = 1U W = 20U
M2 OUT IN 0 0 NCH L = 1U W = 20U
VCC VCC 0 5
VIN IN 0 0 0 *Nodo al que se le hace el barrido DC
CLOAD OUT 0 750f

*Analisis DC
* Se realiza un barrido DC de 0V a 5V con un incremento
* de 0.1V en cada iteracion
```

```
.DC VIN 0 5 0.1
.PRINT DC  V(IN) V(OUT)

.OPTION POST

.END
```

Una vez realizada la simulación en *PrimeSim* HSPICE, se procedió a cargar los resultados en *WaveView*, en donde se obtuvo la siguiente gráfica:

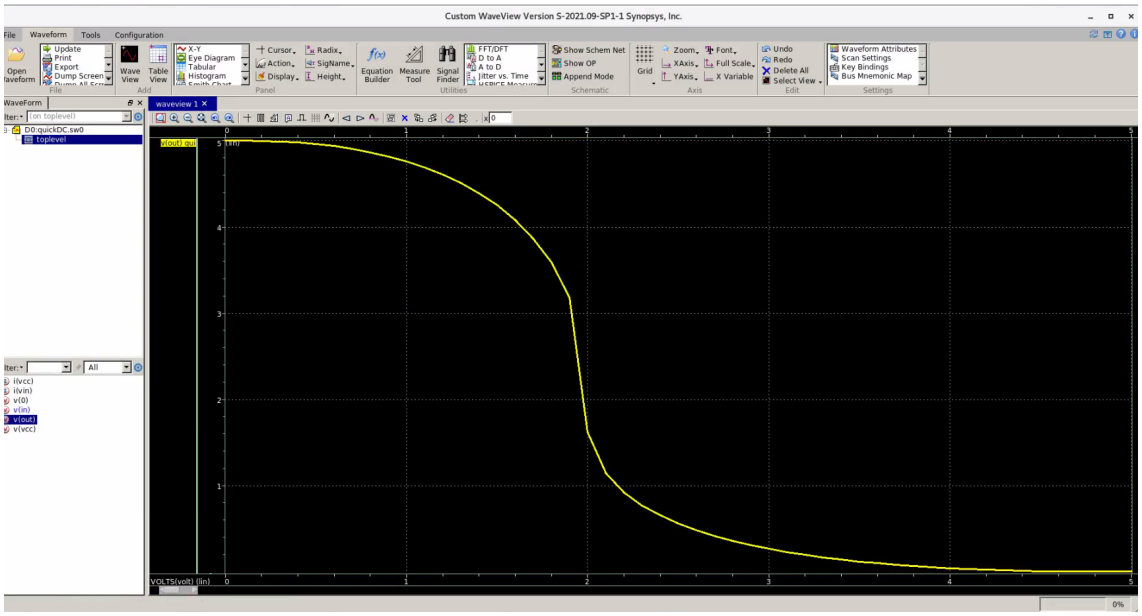


Figura 26: Vista de la herramienta *WaveView* con los filtros de las mediciones realizadas en la simulación.

Mediante la gráfica mostrada en la Figura 42 se pueden determinar márgenes de ruido para determinar, en el peor caso, si una señal de entrada será válida para el correcto funcionamiento del circuito. En otras palabras, mediante el análisis de este tipo de gráficas, se puede determinar las características de transferencia: de un circuito con el objetivo de determinar la zona prohibida: del circuito.

8.1.2. Análisis AC

Análisis AC sin variación

Para realizar el análisis AC se diseñó un circuito RC que corresponde a un filtro pasivo, al cual se le realizará un barrido de frecuencia. El Deck utilizado es el siguiente:

```
*Circuito RC
```

```
*Construccion del circuito
V1 1 0 10 AC 1
R1 1 2 1K
R2 2 0 1K
C1 2 0 1P

*Analisis AC
.OPTION LIST NODE POST
.OP
.AC DEC 10 1K 1MEG
.PRINT AC V(1) V(2) I(R2) I(C1)

.END
```

Una vez realizada la simulación en *PrimeSim* HSPICE, se procedió a cargar los resultados en *WaveView*, en donde se obtuvo la siguiente gráfica:

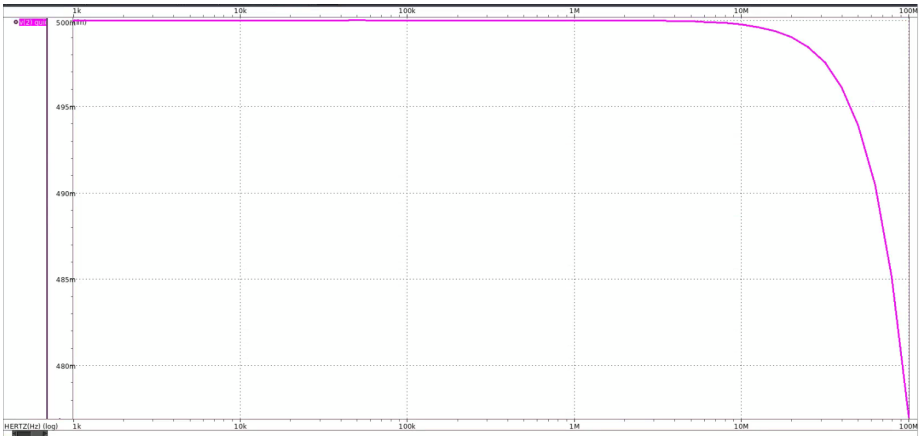


Figura 27: Resultados de la simulación.

Gracias al análisis se puede determinar que el circuito funciona como un filtro pasa-bajas cuya frecuencia de corte se encuentra aproximadamente en 16.8 MHz. Este análisis resulta muy valioso para realizar diseños de circuito RC que funcionarán con fuentes de voltaje AC, ya que permite determinar las frecuencias a la que debe estar la señal de voltaje para que el circuito funcione correctamente.

Análisis AC usando Monte Carlo

Para realizar el análisis AC se diseñó un circuito RC que corresponde a un filtro, al cual se le realizará un barrido de frecuencia y se utilizará un análisis de Monte Carlo para determinar como cambia el voltaje en el nodo de salida al momento de realizar una serie de variaciones en el valor de la resistencia R2. El deck utilizado es el siguiente:

```
*Circuito RC
```

```
*Parámetro que será variado
.PARAM res_val=AUNIF(10000,9000)
*Construccion del circuito
V1 1 0 50 AC 50
R1 1 2 1K
R2 2 0 res_val
C1 2 0 1P

*Analisis AC VARIACION MONTECARLO
.OPTION LIST NODE POST
.OP
.AC DEC 10 1K 100MEG SWEEP MONTE=50 * Se configuran 50 iteraciones
.PRINT AC V(1) V(2) I(R2) I(C1)

.END
```

Una vez realizada la simulación en *PrimeSim* HSPICE, se procedió a cargar los resultados en *WaveView*, en donde se obtuvo la siguiente gráfica:

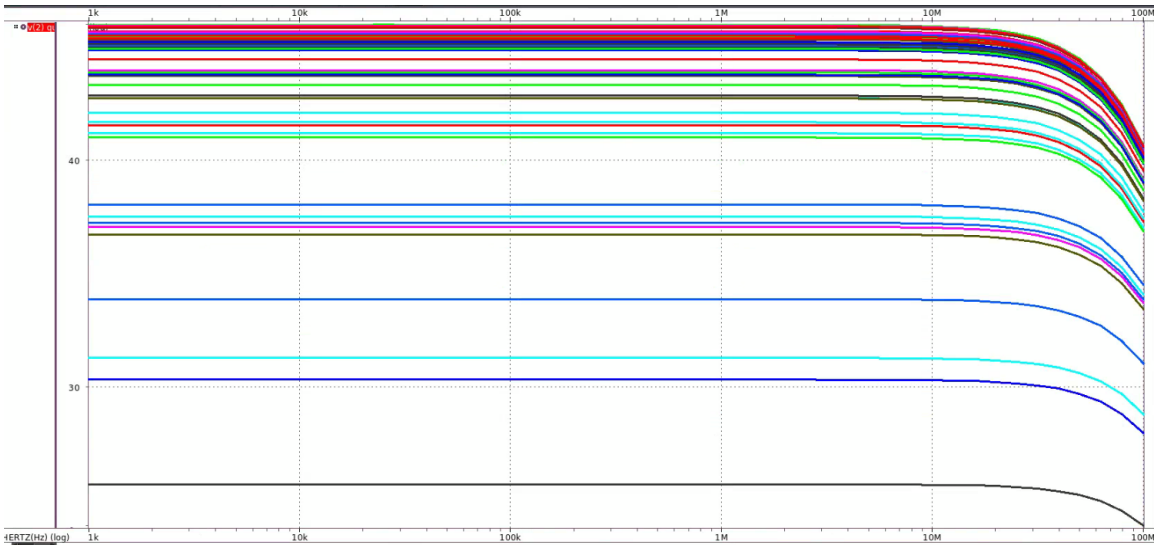


Figura 28: Resultados de la simulación.

Al variar el valor de la resistencia que se encuentra en paralelo al capacitor, se está variando también la distribución del divisor de voltaje. Es por ello que en la gráfica de la Figura 44 se muestra como el voltaje de operación en el nodo 2 varía en cada una de las 50 iteraciones realizadas en el análisis de Monte Carlo. Dado que R2 forma parte del divisor de voltaje, mientras más grande sea este valor, mayor será el voltaje presente en el nodo 2.

8.1.3. Análisis transitorio

Para realizar este análisis se diseñó una compuerta NOT a la que se trató de estudiar su comportamiento en el tiempo según la temperatura a la que esta se encuentra. Para ello se utilizó un barrido de temperatura iniciando en -55°C con un incremento de 10°C por cada iteración hasta que se llegará al punto final que era 75°C. El deck utilizado es el siguiente:

```
*Circuito NOT

*Importacion de los modelos de MOSFET
.MODEL PCH PMOS LEVEL=1
.MODEL NCH NMOS LEVEL=1

*Construccion del circuito
M1 VCC IN OUT VCC PCH L = 1U W = 20U
M2 OUT IN 0 0 NCH L = 1U W = 20U
VCC VCC 0 5
VIN IN 0 0 PULSE .2 4.8 2N 1N 1N 5N 20N
CLOAD OUT 0 .75P

*Analisis transiente
.OPTION LIST NODE POST
.TRAN 200P 20N SWEEP TEMP -55 75 10
.PRINT TRAN V(IN) V(OUT)

.END
```

Una vez realizada la simulación en *PrimeSim* HSPICE, se procedió a cargar los resultados en *WaveView*, en donde se obtuvo la siguiente gráfica:

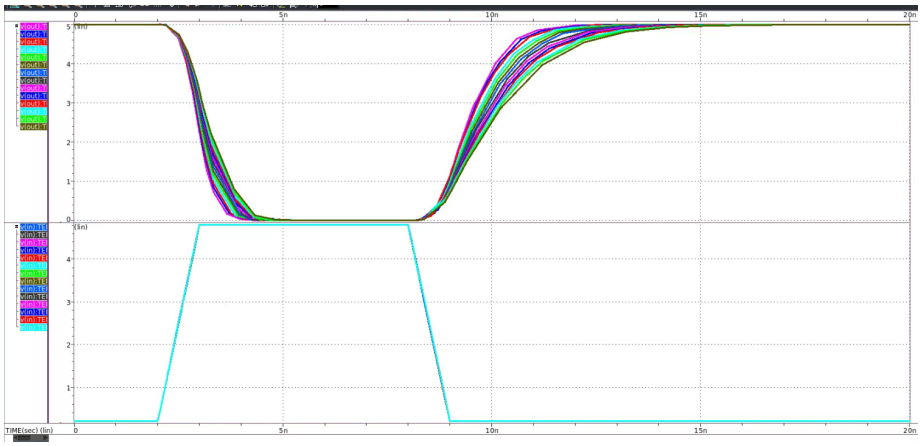


Figura 29: Resultados de la simulación.

Gracias a la gráfica mostrada en la Figura 45, se puede identificar que la temperatura a la que esté expuesta la compuerta NOT, modificará su respuesta transitoria. Por medio de

la gráfica se logra apreciar que las compuertas que estén a mayores temperaturas, tardarán más tiempo en realizar el cambio de estado.

8.1.4. Análisis de frecuencia

Análisis de *Ring Oscillator*

Para este análisis se utilizó un *Ring Oscillator* compuesto por 7 compuertas *NOT*. Para obtener la frecuencia a la que oscila la señal producida por el circuito, se utilizó un análisis de oscilación que utiliza el método de disparo de Newton para obtener la frecuencia de oscilación de la señal. Esto se logra mediante la instrucción .SNOSC en HSPICE.

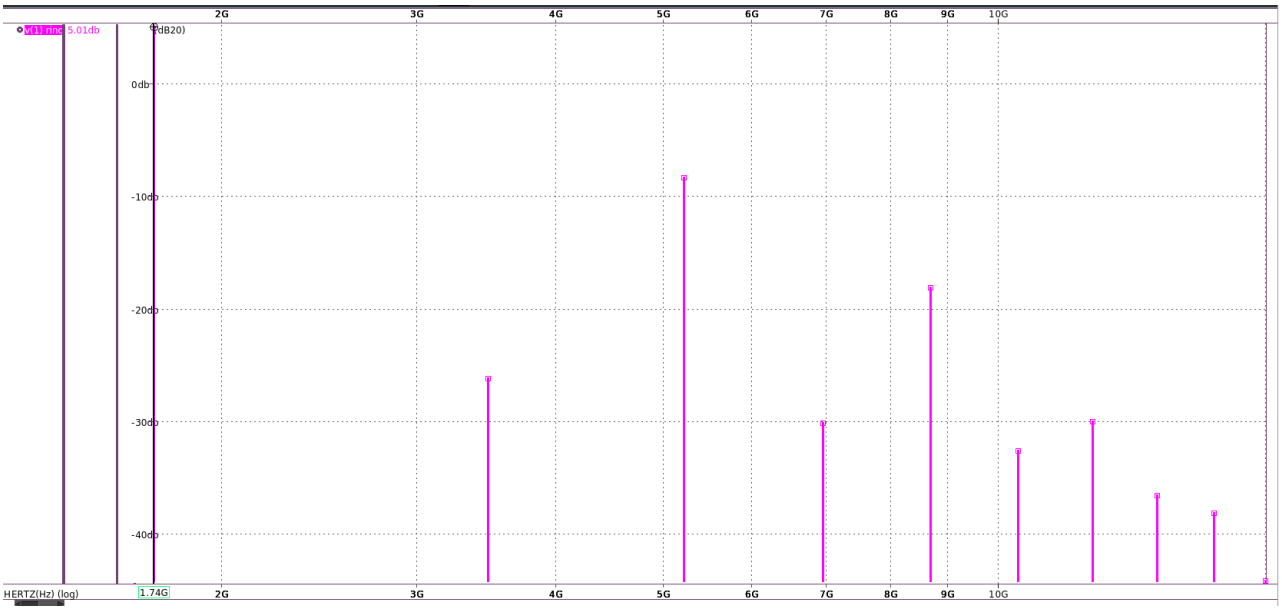


Figura 30: Resultados de la simulación.

Análisis de Ring Oscillator con LPE

Originalmente, se tenía planeado utilizar el análisis de oscilación mediante el método de disparo de Newton. No obstante, antes de realizar este análisis fue necesaria la realización de un análisis transitorio cuyo resultado se muestra en la Figura 31 con el objetivo de obtener un valor de frecuencia cercano al valor real de la frecuencia. La razón de esto es porque el método de disparo de Newton utiliza un punto inicial para comenzar a realizar la aproximación; si en dado caso, este punto inicial no es lo suficientemente cercano al valor real, el método numérico no va a converger. Para hacer más exacta la medición, se decidió utilizar la instrucción .MEASURE. De esta manera se obtuvo que la frecuencia de oscilación era de 22.2122 GHz, como se muestra en la Figura 32

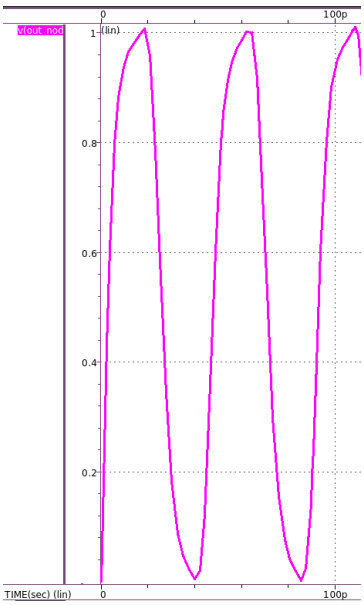


Figura 31: Resultados de la simulación transitoria del circuito.

```
******
*comentario inicial

****** transient analysis tnom=  25.000 temp=  25.000 ******
period=  45.0201354200p  targ=  47.9962132490p   trig=   2.9760778289p
frequency=  22.2122832522g
```

Figura 32: Medición realizada mediante HSPICE sobre la frecuencia del circuito.

Con el valor de frecuencia obtenido mediante el análisis transitorio, se definió que la frecuencia para el inicio del método de disparo de Newton iniciara en 22.21 GHz. Utilizando esta frecuencia como punto de inicio se logró que el método convergiera en el valor de 25 GHz como se muestra en la Figura 33

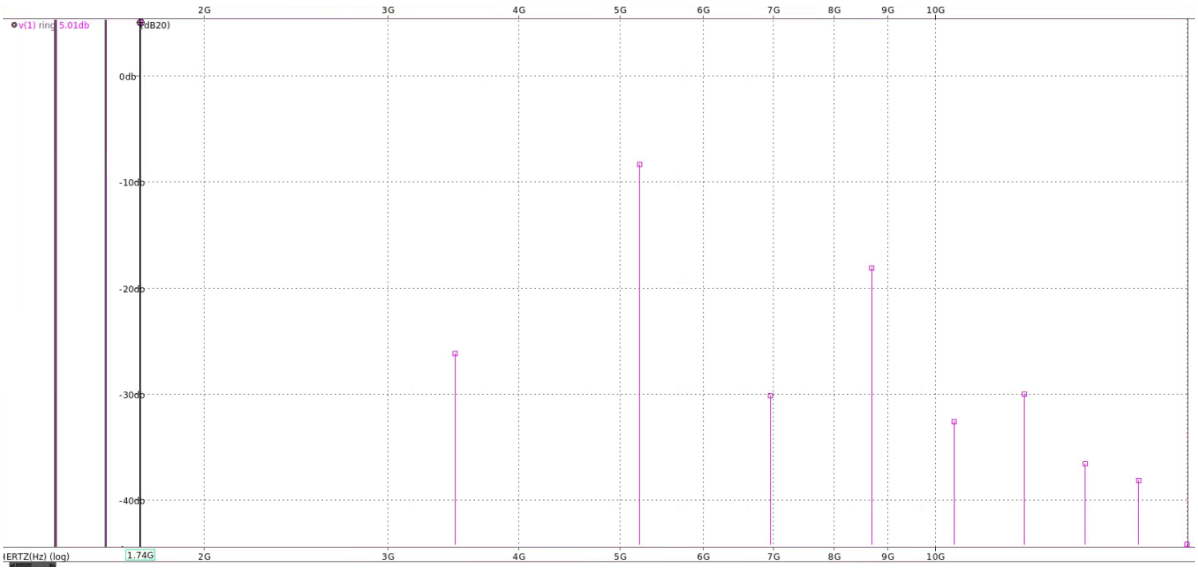


Figura 33: Resultados de la simulación.

8.2. Utilización de *WaveView* para lectura de ASCII

Uno de los principales retos para la simulación del circuito de El Gran Jaguar consiste en la lectura del bus de datos de 8 bits que corresponde a la impresión del texto y verificar que la secuencia de bits sea la correcta. Debido a que el texto que será mostrado por el circuito consta de varios caracteres ASCII, resulta muy complejo y laborioso traducir a mano la secuencia de bits a caracteres ASCII de forma manual. Por tanto, para atacar este reto, surgieron diversas ideas, de las cuales, la más viable debido a su simpleza para ser implementada, fue explorar la herramienta de *WaveView* para identificar alguna funcionalidad para convertir las señales analógicas generadas por los pines de salida del chip, en señales digitales para luego ser agrupadas en un bus de datos de tal forma que la señal resultante en el bus pudiera ser convertida a su equivalente en ASCII.

Para comprobar esta funcionalidad se creó un Deck en HSPICE que estuviera formado por 8 señales de reloj, cuya frecuencia fuera un el resultado de elevar el número 2 a una potencia. Esto con el objetivo de crear un contador de 8 bits cuyo rango fuera de 0 a 255 12.3, ya que corresponde al rango de valores ASCII. A este se le aplicó un análisis transitorio para que simule la señal de salida de cada uno de los pines del contador, el cual se muestra en la Figura 34.

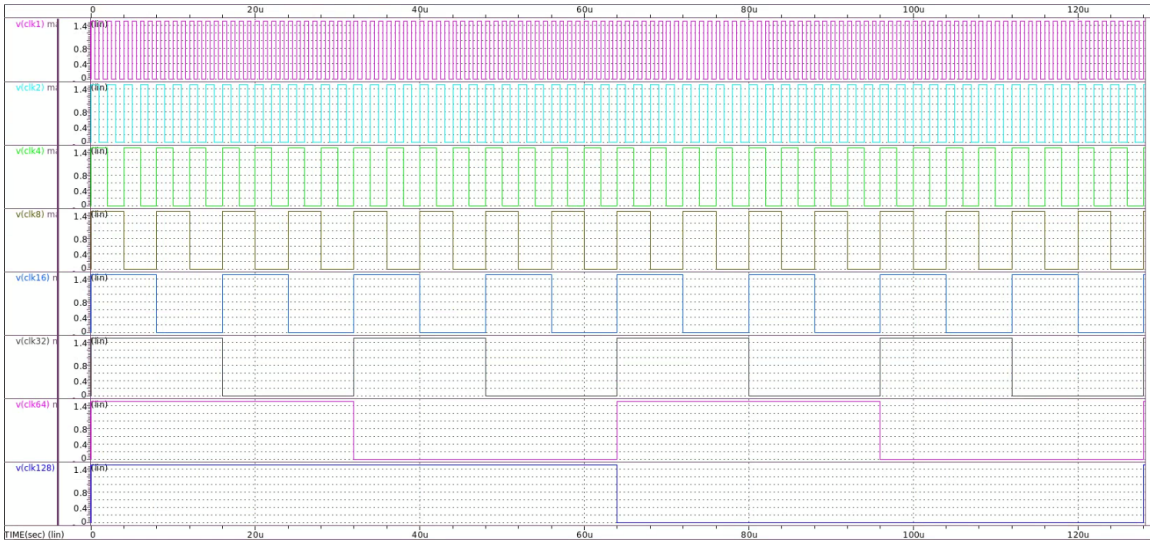


Figura 34: Señal analógica de salida del contador de 8 bits

Una vez conseguido esto, se procedió a convertir la señal analógica que sale de los pines del chip en una señal digital mediante la función *A to D* presente en *WaveView* obteniendo el siguiente resultado mostrado en la Figura 35

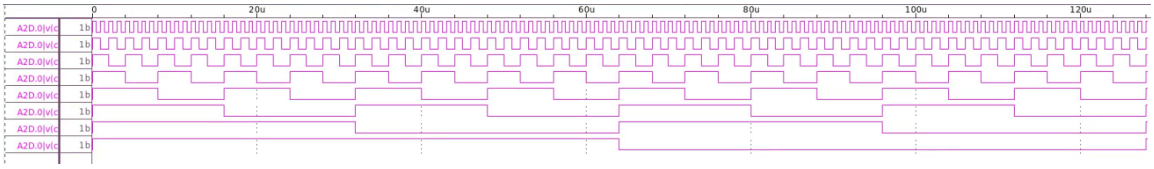


Figura 35: Conversión analógica a digital mediante WaveView.

Una vez obtenidas las señales digitales, se agruparon mediante un bus y se definió que la señal de salida del bus fuera mostrado en formato ASCII. Obteniendo la secuencia de letras del abecedario junto con otros caracteres ASCII, Figura 36 y 37. Esto demuestra que mediante la herramienta de *WaveView* se puede hacer la lectura de los bits de salida del chip El Gran Jaguar y convertirlos a formato ASCII para verificar que el texto que esté siendo impreso por el circuito integrado sea el correcto.

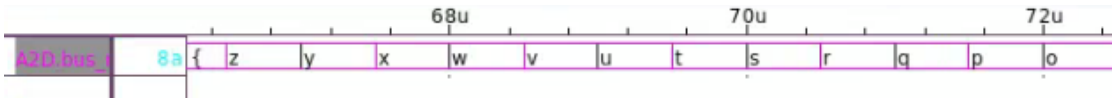


Figura 36: Representación de un bus de señales digitales en formato ASCII.

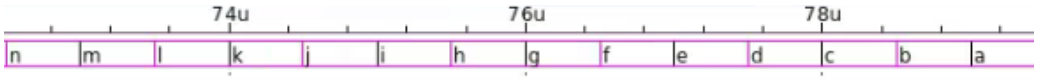


Figura 37: Representación de un bus de señales digitales en formato ASCII.

8.3. Utilización de *WaveView* para análisis de multiplicador de 32-bits

Originalmente, se tenía planeado emplear *PrimeSim* HSPICE para simular el correcto funcionamiento del circuito de El Gran Jaguar. Sin embargo, el equipo encargado de extracción de parásitos se encontró con una serie de obstáculos que le impidieron generar el deck que debe ser simulado. Ante esta situación se optó por realizar un deck de un multiplicador de 32-bits, dado que la forma para simular el funcionamiento del multiplicador es muy similar a como debe ser simulado el circuito de El Gran Jaguar, puesto que ambos tienen pines de salidas que forman un bus de datos y utilizan una señal de reloj para cambiar los valores de dicho bus.

La simulación de este circuito multiplicador permitió profundizar en la funcionalidad de *WaveView* para el análisis de datos generados a partir de un análisis transitorio. El multiplicador de 32-bits es uno de tipo secuencial que fue construido a partir de un sumador de 32-bit y un *Shift Register* de 64-bit. Para armar este circuito, se tomaron como base los decks realizados en el proyecto final del curso de Nanoelectrónica en el año 2020 [21].

Dado que este circuito se trata de un multiplicador secuencial de 32-bit, es necesaria la ejecución de 32 pulsos de reloj para que se muestre el resultado. Para lograr esto fue necesaria la definición de dos características importantes. La primera fue la señal de reloj; y la segunda consistió en el tiempo de simulación de cara a poder simular los 32 pulsos de reloj necesarios para obtener el resultado de la multiplicación.

Para la primera característica se definió un reloj con una frecuencia de 100 MHz, esto se logró mediante la siguiente línea en el deck de HSPICE.

```
V2 CLOCK  0 PULSE (0 Vdd1 10n 100p 100p 5n 10n)
```

Para la segunda característica expuesta, se definió un tiempo de simulación de 325 nanosegundos con un período de muestreo de 100 picosegundos. Para ello se utilizó la siguiente línea en el deck de HSPICE.

```
.TRAN 100p 325n
```

Los decks completos utilizados para la simulación de este circuito se pueden encontrar en la sección de anexos. Una vez realizada la simulación, se procedió a abrir el archivo de los resultados del análisis transitorio mediante *WaveView* con el objetivo de digitalizar las señales de salida, unificarlas en un mismo bus de datos para luego convertirlas a un tipo de dato decimal y comprobar que el resultado de la multiplicación fuera el correcto. Una vez hecho esto, se obtienen los gráficos de la Figura 38

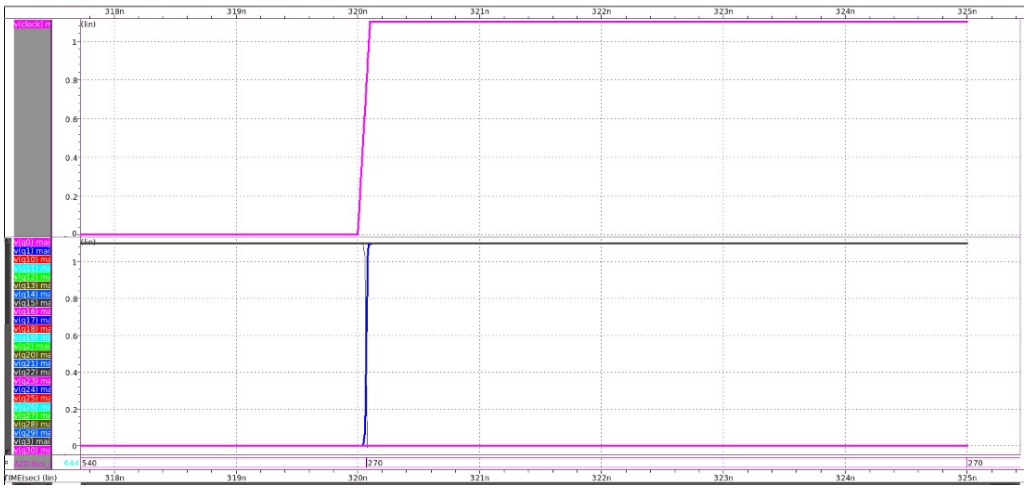


Figura 38: Representación de un bus de señales digitales en formato decimal de un multiplicador de 32 bits.

Figura 39: Zoom del resultado del bus de datos.

Como se observa en la Figura 39, el valor decimal mostrado al cabo de 32 pulsos de reloj, y por consiguiente, el resultado de la multiplicación es de 270.

CAPÍTULO 9

Conclusiones

- *WaveView* es una herramienta que permite traducir señales analógicas a señales digitales para que estas puedan ser procesadas como tales.
- Para determinar la frecuencia de oscilación de un *Ring Oscillator*, se puede utilizar la instrucción .SNOSC. Sin embargo, esta necesita un valor inicial lo suficientemente cercano al resultado real para poder converger.
- Resulta conveniente realizar un análisis transitorio para determinar valores iniciales aproximados, en caso se quiera utilizar un análisis que emplee métodos numéricos.
- Dado que el proceso de fabricación de un nanochip puede tomar varios meses y recursos, es importante realizar simulaciones de este para comprobar que lo diseñado funciona correctamente.
- Como se observa en la sección de antecedentes, se logró cumplir con el objetivo de replicar los trabajos de las generaciones anteriores, con lo cual se pudo tomar como punto de partida las lecciones aprendidas en las iteraciones pasadas de la línea de investigación.
- Se realizó un manual de inicio rápido para que futuros estudiantes que formen parte de la línea de investigación tengan acceso a una serie de análisis que pueden emplear para simular el funcionamiento de futuros diseños.
- Debido a que no fue posible generar el archivo de extracción de parásitos para el circuito de El Gran Jaguar, no se pudieron realizar simulaciones en HSPICE para comprobar que su funcionamiento sea el adecuado y que el diseño no presentara errores. Ante este escenario se optó por simular un multiplicador de 32-bits debido a las similitudes que comparten entre sí.

CAPÍTULO 10

Recomendaciones

- Para futuras investigaciones dentro de la línea de investigación, se recomienda utilizar las aplicaciones de NanoTime, PrimePower, TestMax para realizar análisis de tiempos, consumo de potencia y análisis de entradas por medio de un entorno gráfico y dedicado para este tipo de análisis.
- Dado que *WaveView* permite analizar de forma gráfica los resultados de una simulación mediante HSPICE, se recomienda que en futuros trabajos se profundice más en la utilización de esta herramienta.
- Para la elaboración de un futuro flujo de diseño se recomienda la inclusión de la herramienta de *PrimeSim* PrimeWave para poder realizar simulaciones de diseños mediante un entorno gráfico.

CAPÍTULO 11

Bibliografía

[1] J. De los Santos, "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.," Universidad del Valle de Guatemala, 2014, págs. 15-28.

[2] S. H. Rubio Vasquez, "Definición del Flujo de Diseño para Fabricación de un Chip con Tecnologia VLSI CMOS," Universidad del Valle de Guatemala, 2019, págs. 2-29.

[3] L. A. Nájera Vásquez, "Implementación de circuitos sintetizados a nivel netlist a partir de un diseño en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado.," Universidad del Valle de Guatemala, 2019, págs. 1-52.

[4] M. G. Flores Espino, "Corrección de anillo de entradas/salidas y pruebas de antenna y ERC para la definición del flujo de diseño del primer chip con tecnología nanométrica desarrollado en Guatemala," Universidad del Valle de Guatemala, 2020, págs. 1-72.

[5] J. R. Girón Rubio, "Etapa de verificación física de Diseño en Silicio vs. Esquemático (LVS) en el flujo de diseño para un chip a nanoescala," Universidad del Valle de Guatemala, 2020, págs. 1-74.

[6] J. A. Ayala Escobar, "Diseño de un Circuito Integrado con Tecnología de 180 nm Usando Librerías de Diseño de TSMC: Ejecución de la Síntesis Física, Verificaciones de Antena y Corrección de Errores Obtenidos.," Universidad del Valle de Guatemala, 2021, págs. 24-62.

[7] A. Altuna Hernández, "Diseño de un Circuito Integrado con Tecnología de 180nm usando Librerías de Diseño de TSMC: Ejecución de la Síntesis Física, Verificación de Reglas de Diseño y Corrección de Errores Obtenidos.," Universidad del Valle de Guatemala, 2021, págs. 8-97.

[8] J. E. Shin Jo, "Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: ejecución de la síntesis física, validación de reglas eléctricas y corrección de errores obtenidos," Universidad del Valle de Guatemala, 2021, págs. 2-76.

[9] J. A. Ruiz Orozco, "Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: ejecución de la fase de verificación física Layout vs Schematic (LVS)," Universidad del Valle de Guatemala, 2021, págs. 14-66.

[10] *PrimeSim Pro Datasheet*, CS658874826-DG | PrimeSim Pro DS, Synopsys, Inc, mar. de 2021.

[11] *PrimeSim SPICE Datasheet*, CS663400928-PrimeSim-spice-ds, Synopsys, Inc, abr. de 2021.

[12] *PrimeSim XA Datasheet*, CS648062439-DG | PrimeSim XA DS, Synopsys, Inc, mar. de 2021.

[13] *PrimeSim Custom Fault Datasheet*, CS647523138-PrimeSim Custom Fault DS, Synopsys, Inc, mar. de 2021.

[14] *PrimeSim HSPICE Datasheet*, CS648232768-DG | PrimeSim HSPICE ds, Synopsys, Inc, mar. de 2021.

[15] *PrimeSim Continuum Datasheet*, CS685001915-PrimeSim Continuum ds, Synopsys, Inc, mayo de 2021.

[16] *PrimeSim Reliability Analysis Datasheet*, CS665883462-PrimeSim Reliability Analysis DS, Synopsys, Inc, abr. de 2021.

[17] *PrimeWave Design Environment Datasheet*, CS667099719 PrimeWave DS, Synopsys, Inc, abr. de 2021.

[18] *Custom Compiler: Best-in-Class Technology for Advanced-node Custom Design*, CS695973833 Custom Compiler ds, Synopsys, Inc, mayo de 2021.

[19] *PrimeSim™ HSPICE S-2021.09-SP2 User Documentation*, ver. S-2021.09-SP2, Synopsys, Inc, sep. de 2021.

[20] N. Integration y M. (Group. "180nm BSIM3 model card for bulk CMOS: V0.0." (), dirección: http://ptm.asu.edu/modelcard/180nm_bulk.txt. (accedido: 18.04.2022).

[21] C. E. et al. "Multiplicador de 24 bits en HSPICE2008." (), dirección: https://github.com/iri16009/multiplicador-nano-2020/blob/master/README.md. (accedido: 01.11.2022).

[22] N. Integration y M. (Group. "32nm PTM HP model: V2.1." (), dirección: http://ptm.asu.edu/modelcard/HP/32nm_HP.pm. (accedido: 01.11.2022).

CAPÍTULO 12

Anexos

12.1. Código de Decks realizados para simular *Blackboxes*

A continuación se muestra el código utilizado para crear los decks simulados en la sección de Decks realizados 7.1

```
*COMENTARIO INICIAL

*** Subcircuit for NOT Gate ***
.SUBCKT NotGate in1 out1 vsource size = 1
  M1 out1 in1 0 0 nmos W='size*UNIT_W' L=UNIT_L
  M2 out1 in1 vsource vsource pmos W='2*size*UNIT_W' L=UNIT_L
.ENDS

*** Subcircuit for XOR Gate ***
* D G S B
.SUBCKT XorGate in1 in2 out1 vsource size = 1
  *Nego las entradas
  X5 in1 in1_neg vsource NotGate size = 1
  X6 in2 in2_neg vsource NotGate size = 1

  *TOPOLOG A CMOS
  *PUN
  M1 n1 in1_neg vsource vsource pmos W='2*size*UNIT_W' L=UNIT_L
  M2 out1 in2 n1 n1 pmos W='2*size*UNIT_W' L=UNIT_L
  M3 n2 in1 vsource vsource pmos W='2*size*UNIT_W' L=UNIT_L
  M4 out1 in2_neg n2 n2 pmos W='2*size*UNIT_W' L=UNIT_L
  *PDN
  M5 out1 in1 n3 n3 nmos W='size*UNIT_W' L=UNIT_L
  M6 n3 in2 0 0 nmos W='size*UNIT_W' L=UNIT_L
  M7 out1 in1_neg n4 n4 nmos W='size*UNIT_W' L=UNIT_L
  M8 n4 in2_neg 0 0 nmos W='size*UNIT_W' L=UNIT_L
.ENDS

*** Subcircuit for NAND Gate ***
.SUBCKT NandGate in1 in2 out1 vsource size = 1
  M1 out1 in1 vsource vsource pmos W='2*size*UNIT_W' L=UNIT_L
```

```
  M2 out1 in2 vsource vsource pmos W='2*size*UNIT_W' L=UNIT_L
  M3 out1 in1 m m nmos W='size*UNIT_W' L=UNIT_L
  M4 m in2 0 0 nmos W='size*UNIT_W' L=UNIT_L
.ENDS

*** Subcircuit for NOR Gate ***
* D G S B
.SUBCKT NorGate in1 in2 out_nor vsource size = 1
  M7 out_nor in1 x vsource pmos W='2*size*UNIT_W' L=UNIT_L
  M8 x in2 vsource vsource pmos W='2*size*UNIT_W' L=UNIT_L
  M9 out_nor in1 0 0 nmos W='size*UNIT_W' L=UNIT_L
  M10 out_nor in2 0 0 nmos W='size*UNIT_W' L=UNIT_L
.ENDS

*** Subcircuit for AND Gate ***
.SUBCKT AndGate in1 in2 out1 vsource size = 1
    X1 in1 in2 sal1 vsource NandGate size = 1
    X2 sal1 sal1 out1 vsource NandGate size = 1
.ENDS

*** Subcircuit for OR Gate ***
.SUBCKT OrGate in1 in2 out1 vsource size = 1
  X1 in1 in2 out_nor vsource NorGate size = 1
  X2 out_nor out_nor out1 vsource NorGate size = 1
.ENDS

**************************************************
* PRIMER SUBCIRCUITO - Listo
*
.SUBCKT IAO21D0BWP7T A1 A2 B ZN vsource
    * Se arma la primera AND
    X1 A1 A1_neg vsource NotGate size = 1
    X2 A2 A2_neg vsource NotGate size = 1
    X3 A1_neg A2_neg out1 vsource AndGate size = 1
    * Se arma el NOR
    X4 out1 B ZN vsource NorGate size = 1
.ENDS
**************************************************

**************************************************
*SEGUNDO SUBCIRCUITO - Listo
*
.SUBCKT AOI211D1BWP7T A1 A2 B C ZN vsource
    * Se arma la AND
    X1 A1 A2 out1 vsource AndGate size = 1
    * Se arma el NOR de 3 entradas a partir de 1 OR de 2 entradas conectadas
     a un OR
    X2 B C out_nor vsource OrGate size = 1
  X3 out1 out_nor ZN vsource NorGate size = 1
.ENDS
**************************************************

**************************************************
*TERCER SUBCIRCUITO - Listo
* 2 INPUT AND
.SUBCKT AN2XD1BWP7T A1 A2 Z vsource
  X1 A1 A2 Z vsource AndGate size = 1
.ENDS
**************************************************
```

```
90
91 ****************************************************
92 *CUARTO SUBCIRCUITO
93 * D FLIP-FLOP WITH ASYNC CLEAR - Listo
94 .SUBCKT DFCND0BWP7T CDN CP D Q QN vsource
95   X1 S2 S S1 vsource NandGate size = 1
96   X2 S1 CP A1 vsource AndGate size = 1
97   X3 A1 CDN S vsource NandGate size = 1
98   X4 S CP A2 vsource AndGate size = 1
99   X5 A2 S2 R vsource NandGate size = 1
100  X6 R D A3 vsource AndGate size = 1
101  X7 A3 CDN S2 vsource NandGate size = 1
102  X8 S QN Q vsource NandGate size = 1
103  X9 Q R A4 vsource AndGate size = 1
104  X10 A4 CDN QN vsource NandGate size = 1
105
106 .ENDS
107 ****************************************************
108
109 ****************************************************
110 *QUINTO SUBCIRCUITO - Listo
111 *
112 .SUBCKT AOI31D0BWP7T A1 A2 A3 B ZN vsource
113   * Se crea el 3-input AND por medio de 2 2-input AND
114   X1 A1 A2 and1 vsource AndGate size = 1
115   X2 and1 A3 and2 vsource AndGate size = 1
116   * Se realiza el NOR
117   X3 and2 B ZN vsource NorGate size = 1
118 .ENDS
119 ****************************************************
120
121 ****************************************************
122 *SEXTO SUBCIRCUITO - Listo
123 * 2-1-1 AO
124 .SUBCKT AO211D0BWP7T A1 A2 B C Z vsource
125   *Se crea el AND
126   X1 A1 A2 and1 vsource AndGate size = 1
127   *Se crea el 3-input or con 2 2-input or
128   X2 B C or1 vsource OrGate size = 1
129   X3 and1 or1 Z vsource OrGate size = 1
130 .ENDS
131 ****************************************************
132
133 ****************************************************
134 *SEPTIMO SUBCIRCUITO - Listo
135 * 2-2 AOI
136 .SUBCKT AOI22D0BWP7T A1 A2 B1 B2 ZN vsource
137   X1 A1 A2 out1 vsource AndGate size = 1
138   X2 B1 B2 out2 vsource AndGate size = 1
139   X3 out1 out2 ZN vsource NorGate size = 1
140 .ENDS
141 ****************************************************
142
143 ****************************************************
144 *OCTAVO SUBCIRCUITO
145 * ECO THE HIGH
146 .SUBCKT TIEHBWP7T Z
147
148
```

```
.ENDS
************************************************

************************************************
*NOVENO SUBCIRCUITO - Listo
* 2-2-2 AO
.SUBCKT AO222D0BWP7T A1 A2 B1 B2 C1 C2 Z vsource
  *Se realizan los ands
  X1 A1 A2 and1 vsource AndGate size = 1
  X2 B1 B2 and2 vsource AndGate size = 1
  X3 C1 C2 and3 vsource AndGate size = 1
  *Se realiza el 3-input or usando 2 2-input or
  X4 and1 and2 or1 vsource OrGate size = 1
  X5 and3 or1 Z vsource OrGate size = 1
.ENDS
************************************************

************************************************
*DECIMO SUBCIRCUITO - Listo
* 2-1-1 OA
.SUBCKT OA211D0BWP7T A1 A2 B C Z vsource
  X1 A1 A2 out_or vsource OrGate size = 1
  *Se crea el AND de 3 entradas como 2-input AND en cascada
  X2 B C out_and1 vsource AndGate size = 1
  X3 out_and1 out_or Z vsource AndGate size = 1
.ENDS
************************************************

************************************************
*ONCEAVO SUBCIRCUITO - Listo
* 2 INPUT EXCLUSIVE NOR
.SUBCKT NR2D1BWP7T A1 A2 ZN vsource
  X1 A1 A2 nor1 vsource NorGate size = 1
  X2 A1 A2 and1 vsource AndGate size = 1
  X3 nor1 and1 ZN vsource OrGate size = 1
.ENDS
************************************************

************************************************
*DOCEAVO SUBCIRCUITO - Listo
* 2-INPUT OR, 2-INPUT NAND INTO 2-INPUT NAND
.SUBCKT MOAI22D0BWP7T A1 A2 B1 B2 ZN vsource
  X1 A1 A2 out_or vsource OrGate size = 1
  X2 B1 B2 out_nand vsource NandGate size = 1
  X3 out_nand out_or ZN vsource NandGate size = 1
.ENDS
************************************************

************************************************
*TRECEAVO SUBCIRCUITO - Listo
* 4-INPUT NOR WITH 2 INVERTED INPUTS
.SUBCKT IINR4D0BWP7T A1 A2 B1 B2 ZN vsource
  *Se niegan las entradas respectivas
  X1 A1 A1_neg vsource NotGate size = 1
  X2 A2 A2_neg vsource NotGate size = 1
  *Se hace el 4-input NOR usando 2 2-input OR conectado a un 2-input NOR
  X3 A1_neg A2_neg out_nor1 vsource OrGate size = 1
  X4 B1 B2 out_nor2 vsource OrGate size = 1
  X5 out_nor1 out_nor2 ZN vsource NorGate size = 1
```

```
.ENDS
*****************************************************

*****************************************************
*14 SUBCIRCUITO
* 2-INPUT NAND WITH 1 INVERTED INPUT
.SUBCKT IND2D1BWP7T A1 B1 ZN vsource
  X1 A1 A1_neg vsource NotGate size = 1
  X2 B1 A1_neg ZN vsource NandGate size = 1
.ENDS
*****************************************************

*****************************************************
*15 SUBCIRCUITO
* 3-INPUT AND
.SUBCKT AN3D1BWP7T A1 A2 A3 Z vsource
  *Se crea el AND de 3 pines poniendo 2 ANDs en cascada
  X1 A1 A2 out_and1 vsource AndGate size = 1
  X2 out_and1 A3 Z vsource AndGate size = 1
.ENDS
*****************************************************

*****************************************************
*16 SUBCIRCUITO
* 2-INPUT OR
.SUBCKT OR2D1BWP7T A1 A2 Z vsource
  * CAPACITANCIA PARASITAS
  C1 A1 0 0.003607p
  C2 A2 0 0.004215P
  C3 Z 0 0.2977p
  * CONSTRUCCION DE CIRCUITO
  X1 A1 A2 Z vsource OrGate size = 1
.ENDS
*****************************************************

*****************************************************
*17 SUBCIRCUITO
* 1-BIT HALF ADDER
.SUBCKT HA1D0BWP7T A B S CO vsource
  X1 A B CO vsource AndGate size = 1
  X2 A B S vsource XorGate size = 1
.ENDS
*****************************************************

*****************************************************
*18 SUBCIRCUITO
* 2-INPUT EXCLUSIVE OR
.SUBCKT CKXOR2D0BWP7T A1 A2 ZN vsource
  X1 A1 A2 ZN vsource XorGate size = 1
.ENDS
*****************************************************

*****************************************************
*19 SUBCIRCUITO
* 2-1-1 OAI
.SUBCKT OAI211D0BWP7T A1 A2 B C ZN vsource
  * Parte del OR de 2 entradas
  X1 A1 A2 out_or vsource OrGate size = 1
  * Parte del NAND de 3 entradas
```

```
  X2 B C out_and1 vsource AndGate size = 1
  X3 out_and1 out_or ZN vsource NandGate size = 1
.ENDS
*****************************************************

*****************************************************
*20 SUBCIRCUITO
* 3-1 OA
.SUBCKT OA31D0BWP7T A1 A2 A3 B Z vsource
  * AND de 3 entradas
  X1 A1 A2 out_or1 vsource OrGate size = 1
  X2 A3 out_or1 out_or2 vsource OrGate size = 1
  * Parte del OR
  X3 out_or2 B Z vsource AndGate size = 1
.ENDS
*****************************************************

*****************************************************
*21 SUBCIRCUITO
* 4-INPUT AND
.SUBCKT AN4D0BWP7T A1 A2 A3 A4 Z vsource
  *Se hace el AND de 4 entradas a partir de 3 ANDs de 2 entradas
  X1 A1 A2 out_and1 vsource AndGate size = 1
  X2 A3 A4 out_and2 vsource AndGate size = 1
  X3 out_and1 out_and2 Z vsource AndGate size = 1
.ENDS
*****************************************************

*****************************************************
*22 SUBCIRCUITO
* 2-2 AO WITH SIMPLE GATES
.SUBCKT AO22D0BWP7T A1 A2 B1 B2 Z vsource
  X1 A1 A2 out_and1 vsource AndGate size = 1
  X2 B1 B2 out_and2 vsource AndGate size = 1
  X3 out_and1 out_and2 Z vsource OrGate size = 1
.ENDS
*****************************************************
```

Listing 12.1: decks

Código en HSPICE para realizar el análisis transitorio de los decks construidos. 12.1

```
*Comentario inicial

.include 'C:\Users\Lab\Desktop\Israel\Diseno\decks_nano\Params.sp'
.include 'C:\Users\Lab\Desktop\Israel\Diseno\decks_nano\lib_180nm.sp'
.include 'C:\Users\Lab\Desktop\Israel\Diseno\decks_nano\Mis_decks.sp'
.include 'C:\Users\Lab\Desktop\Israel\Diseno\decks_nano\Params_clocks.sp'

* Fuente de alimentaci n
v1 vdd 0 1.5

*Se al de reloj que ser  inyectada a cada entrada del circuito.
V_A32 clk32 gnd pulse (0 vdd1 td tr tf pwA32 per32 )
V_A16 clk16 gnd pulse (0 vdd1 td tr tf pwA16 per16 )
V_A8 clk8 gnd pulse (0 vdd1 td tr tf pwA8 per8 )
V_A4 clk4 gnd pulse (0 vdd1 td tr tf pwA4 per4 )
V_A2 clk2 gnd pulse (0 vdd1 td tr tf pwA2 per2 )
V_A1 clk1 gnd pulse (0 vdd1 td tr tf pwA1 per1 )
```

```
18
19 * Instancia de los decks realizados
20 X1 clk1 clk2 clk4 Y1 vdd IAO21D0BWP7T size = 1
21 X2 clk1 clk2 clk4 clk8 Y2 vdd AOI211D1BWP7T size = 1
22 X3 clk1 clk2 Y3 vdd AN2XD1BWP7T size = 1
23 X4 clk1 clk2 clk4 Q QN vdd DFCND0BWP7T size = 1
24 X5 clk1 clk2 clk4 clk8 Y5 vdd AOI31D0BWP7T size = 1
25 X6 clk1 clk2 clk4 clk8 Y6 vdd AO211D0BWP7T size = 1
26 X7 clk1 clk2 clk4 clk8 Y7 vdd AOI22D0BWP7T size = 1
27 *X8 clk1 clk2 Y8 vdd TIEHBWP7T size = 1
28 X9 clk1 clk2 clk4 clk8 clk16 clk32 Y9 vdd AO222D0BWP7T size = 1
29 X10 clk1 clk2 clk4 clk8 Y10 vdd OA211D0BWP7T size = 1
30 X11 clk1 clk2 Y11 vdd NR2D1BWP7T size = 1
31 X12 clk1 clk2 clk4 clk8 Y12 vdd MOAI22D0BWP7T size = 1
32 X13 clk1 clk2 clk4 clk8 Y13 vdd IINR4D0BWP7T size = 1
33 X14 clk1 clk2 Y14 vdd IND2D1BWP7T size = 1
34 X15 clk1 clk2 clk4 Y15 vdd AN3D1BWP7T size = 1
35 X16 clk1 clk2 Y16 vdd OR2D1BWP7T size = 1
36 X17 clk1 clk2 Y17S Y17CO vdd HA1D0BWP7T size = 1
37 X18 clk1 clk2 Y18 vdd CKXOR2D0BWP7T size = 1
38 X19 clk1 clk2 clk4 clk8 Y19 vdd OAI211D0BWP7T size = 1
39 X20 clk1 clk2 clk4 clk8 Y20 vdd OA31D0BWP7T size = 1
40 X21 clk1 clk2 clk4 clk8 Y21 vdd AN4D0BWP7T size = 1
41 X22 clk1 clk2 clk4 clk8 Y22 vdd AO22D0BWP7T size = 1
42 Cc1 Y9 0 1f
43
44 *Configuraci n del an lisis transitorio
45 .TRAN 1e-9 32.1u
46 .Option post
47 .Plot v(clk1)
48
49 .END
```

Listing 12.2: decks

El archivo "Params_clocks.spçontiene el valor de las variables utilizadas para configurar las señales de reloj que ingresan a las entradas de los decks a analizar.

```
1 .param tr=0.05n    ** rise time
2 .param tf=0.05n    ** fall time
3 .param td= 0n     ** initial delay
4 .param per32=32u    ** period
5 .param per16=16u    ** period
6 .param per8=8u    ** period
7 .param per4=4u    ** period
8 .param per2=2u    ** period
9 .param per1=1u    ** period
10 .param pwA32=' (per32-tr-tf)/2'  ** high pulse width
11 .param pwA16=' (per16-tr-tf)/2'  ** high pulse width
12 .param pwA8=' (per8-tr-tf)/2'  ** high pulse width
13 .param pwA4=' (per4-tr-tf)/2'  ** high pulse width
14 .param pwA2=' (per2-tr-tf)/2'  ** high pulse width
15 .param pwA1=' (per1-tr-tf)/2'  ** high pulse width
```

Listing 12.3: decks

12.2. Deck para análisis de *Ring Oscillator*

Deck de simulación en PrimeSim HSPICE para el análisis de frecuencia de la señal de reloj de salida del Ring Oscillator mediante análisis de Fourier.

```
*Comentario inicial
.title ringosc

* Subcircuito de NOT
.subckt inv in out vdd
  mn1 out in 0 0 nmos l=0.25u w=2u
  mp1 out in vdd vdd pmos l=0.25u w=6u
.ends

* Fuente de alimentaci n
vdd vdd 0 3
* Construcci n del ring Oscillator con 7 nots
x1 1 2 vdd inv
x2 2 3 vdd inv
x3 3 4 vdd inv
x4 4 5 vdd inv
x5 5 6 vdd inv
x6 6 7 vdd inv
x7 7 1 vdd inv
c1 1 0 0.022p

* Librer a de CMOS utilizada
.lib 'tsmc018.m' tt

* Condiciones iniciales de la simulaci n
.ic v(1)=3


.options post
.option measdgt=10

* An lisis de oscilacion mediante el m todo de Newton
.snosc tones=1.7e9 nharms=10 oscnode=1 trinit=10n
* An lisis de ruido de fase
.phasenoise v(7) dec 10 100 10meg listfreq=(all) listcount=5 listsources=on
.probe phasenoise phnoise v(7)
.print phasenoise phnoise v(7)

.probe sn v(1)
.probe snfd v(1) v(7)

* Medici n
.measure snfd freqsnfd1 find 'HERTZ[1]' at=0.01p

.end
```

Listing 12.4: decks

12.3. Deck para contador de 8 bits

Deck de simulación en PrimeSim HSPICE para la simulación de un contador de 8 bits mediante señales de reloj, el cual fue utilizado para verificar si la herramienta de *WaveView* contaba con alguna función para convertir las señales del contador a caracteres ASCII.

```
*Deck con las senales del contador de 8 bits

v1 vdd 0 1.5
V_A128 clk128 gnd pulse (0 vdd1 td tr tf pwA128 per128 )
V_A64 clk64 gnd pulse (0 vdd1 td tr tf pwA64 per64 )
V_A32 clk32 gnd pulse (0 vdd1 td tr tf pwA32 per32 )
V_A16 clk16 gnd pulse (0 vdd1 td tr tf pwA16 per16 )
V_A8 clk8 gnd pulse (0 vdd1 td tr tf pwA8 per8 )
V_A4 clk4 gnd pulse (0 vdd1 td tr tf pwA4 per4 )
V_A2 clk2 gnd pulse (0 vdd1 td tr tf pwA2 per2 )
V_A1 clk1 gnd pulse (0 vdd1 td tr tf pwA1 per1 )

* Analisis a realizar para verificar el funcionamiento

.TRAN 1e-8 128.1u
.OPTION LIST NODE POST
.Plot v(clk1) v(clk2) v(clk4) v(clk8) v(clk16) v(clk32) v(clk64) v(clk128)
.PRINT TRAN V(clk1) V(clk2) V(clk4) V(clk8) V(clk16) V(clk32) V(clk64) V(
    clk128)

.END
```

Listing 12.5: decks

12.4. Deck para el análisis de multiplicador de 32-bit

Decks de simulación en HSPICE para realizar un análisis transitorio de un multiplicador de 32-bit. Dada la complejidad de la realización de un circuito de este tipo, se realizaron distintos archivos los cuales fueron invocados desde un mismo deck llamado main.sp.

12.4.1. main.sp

En este archivo se invocaron todos los decks que son necesarios para la simulación del multiplicador. Ya que este es el archivo central, este es el que se debe de invocar al momento de realizar la simulación de HSPICE. Además, en este deck se hace la instancia del sumador de 32-bit y el *Shift Register* de 64-bit. El código es el siguiente:

```
****************************** INCLUDE
    ************************************
.include 'sub.sp'
.include 'sources.sp'
.include 'params.sp'
.include '32nm.sp'
.include 'Analysis.sp'
.include 'subct_adder12bits.sp'
```

```
8
9 ***************************** ADDER 32 BITS
      ***********************************
10 *     |     RESULTADO DEL SHIFT ENTRA A SUMADOR
11 Xi5  Q63 Q62 Q61 Q60 Q59 Q58 Q57 Q56 Q55 Q54 Q53 Q52 Q51 Q50 Q49 Q48 Q47 Q46
       Q45
12 +Q44 Q43 Q42 Q41 Q40 Q39 Q38 Q37 Q36 Q35 Q34 Q33 Q32 0 0 0 0 0 0 0 0 0 0 0
       0 0 0
13 +0 0 0 0 0 0 0 0 0 0 0 0 0 Vdd 0 0 Vdd          0 carryout s31 s30 s29 s28
      s27 s26
14 +s25 s24 s23 s22 s21 s20 s19 s18 s17 s16 s15 s14 s13 s12 s11 s10 s09 s08 s07
       s06 s05
15 +s04 s03 s02 s01 s00 vdd sum32 size = 1
16 ********************************************************************************

17
18 ****************************** SHIFT REGISTER CARGA PARALELA
      ********************
19 Xi6 Q0 CLOCK carryout s31 s30 s29 s28 s27 s26 s25 s24 s23 s22 s21 s20 s19
      s18
20 +s17 s16 s15 s14 s13 s12 s11 s10 s09 s08 s07 s06 s05 s04 s03 s02 s01 s00 0 0
       0
21 +0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Q63 Q62 Q61 Q60 Q59
       Q58
22 +Q57 Q56 Q55 Q54 Q53 Q52 Q51 Q50 Q49 Q48 Q47 Q46 Q45 Q44 Q43 Q42 Q41 Q40 Q39
       Q38
23 +Q37 Q36 Q35 Q34 Q33 Q32 Q31 Q30 Q29 Q28 Q27 Q26 Q25 Q24 Q23 Q22 Q21 Q20 Q19
       Q18
24 +Q17 Q16 Q15 Q14 Q13 Q12 Q11 Q10 Q9 Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 vdd SRP size
      =1
25 ********************************************************************************

26
27
28 *MULTIPLICADOR SE CARGA EN Q63 A Q0
29 .IC V(Q63) = 0
30 .IC V(Q62) = 0
31 .IC V(Q61) = 0
32 .IC V(Q60) = 0
33 .IC V(Q59) = 0
34 .IC V(Q58) = 0
35 .IC V(Q57) = 0
36 .IC V(Q56) = 0
37 .IC V(Q55) = 0
38 .IC V(Q54) = 0
39 .IC V(Q53) = 0
40 .IC V(Q52) = 0
41 .IC V(Q51) = 0
42 .IC V(Q50) = 0
43 .IC V(Q49) = 0
44 .IC V(Q48) = 0
45 .IC V(Q47) = 0
46 .IC V(Q46) = 0
47 .IC V(Q45) = 0
48 .IC V(Q44) = 0
49 .IC V(Q43) = 0
50 .IC V(Q42) = 0
51 .IC V(Q41) = 0
52 .IC V(Q40) = 0
```

```
.IC V(Q39) = 0
.IC V(Q38) = 0
.IC V(Q37) = 0
.IC V(Q36) = 0
.IC V(Q35) = 0
.IC V(Q34) = 0
.IC V(Q33) = 0
.IC V(Q32) = 0

.IC V(Q31) = 0
.IC V(Q30) = 0
.IC V(Q29) = 0
.IC V(Q28) = 0
.IC V(Q27) = 0
.IC V(Q26) = 0
.IC V(Q25) = 0
.IC V(Q24) = 0
.IC V(Q23) = 0
.IC V(Q22) = 0
.IC V(Q21) = 0
.IC V(Q20) = 0
.IC V(Q19) = 0
.IC V(Q18) = 0
.IC V(Q17) = 0
.IC V(Q16) = 0
.IC V(Q15) = 0
.IC V(Q14) = 0
.IC V(Q13) = 0
.IC V(Q12) = 0
.IC V(Q11) = 0
.IC V(Q10) = 0
.IC V(Q9) = 0
.IC V(Q8) = 0
.IC V(Q7) = 0
.IC V(Q6) = 0
.IC V(Q5) = 0
.IC V(Q4) = Vdd1
.IC V(Q3) = Vdd1
.IC V(Q2) = Vdd1
.IC V(Q1) = Vdd1
.IC V(Q0) = 0

.option post

.END
```

Listing 12.6: decks

12.4.2. sources.sp

En este deck se encuentran las señales que se utilizarán para alimentar el circuito; es decir la señal de alimentación de 1.1 V y la señal de reloj de 100 MHz. Para lograr esto se construyó el siguiente deck:

```
******* VOLTAGE SOURCES **********************

```

```
3 V1 Vdd 0 Vdd1
4
5 ********************** CLOCK SIGNAL
      ***************************************
6 V2 CLOCK  0 PULSE (0 Vdd1 10n 100p 100p 5n 10n)
```

Listing 12.7: decks

12.4.3. params.sp

En este deck se encuentran la definición de los parámetros de voltaje y dimensiones del transistor a utilizar. Las instrucciones son las siguientes:

```
1 *DEFINICION DE PARAMETROS
2
3 .PARAM Vdd1=1.1
4 .PARAM Tech = 32e-9
5 .PARAM UNIT_W = '2*Tech'
6 .PARAM UNIT_L = Tech
```

Listing 12.8: decks

12.4.4. 32nm.sp

En este deck se encuentra la definición de la tecnología de MOSFET de 32nm la cual se encuentra disponible en internet gracias a la Universidad Estatal de Arizona [22]

12.4.5. Analysis.sp

En este deck se encuentra el análisis transitorio a ejecutar. Para este caso se define un análisis con una duración de 325 nanosegundos y con un período de muestreo de 100 picosegundos. Para ello se utilizan las siguientes instrucciones:

```
1 ****** Secci n del an lisis ******
2 .TRAN 100p 325n
3
4 ****************************
```

Listing 12.9: decks

12.4.6. subct_adder32bits.sp

En este deck se encuentran las compuertas lógicas necesarias para armar el sumador de 32 bits con el que se construirá el multiplicador; entre las cuales se encuentran compuertas XOR, OR, AND, NAND, NOR y NOT. Las instrucciones para conseguir esto son las siguientes:

```
1 *COMPUERTA NAND2:
2 *ENTRADA: in1, in2
3 *SALIDA:  out 1
```

```
4
5 .SUBCKT Nand2 in1 in2 out1 vsource size = 1
6   M1 out1 in1 vsource vsource PMOS W='size*UNIT_W' L=UNIT_L
7   M2 out1 in2 vsource vsource PMOS W='size*UNIT_W' L=UNIT_L
8
9   M3 out1 in1 J 0 NMOS W='size*UNIT_W' L=UNIT_L
10  M4 J in2 0 0 NMOS W='size*UNIT_W' L=UNIT_L
11 .ENDS
12
13 ********************************************************************
14 *COMPUERTA NOR2:
15 *ENTRADA: in1, in2
16 *SALIDA:  out1
17
18 .SUBCKT Nor2 in1 in2 out1 vsource size = 1
19   M1 out1 in2 p1 vsource PMOS W='size*UNIT_W' L=UNIT_L
20   M2 p1 in1 vsource vsource PMOS W='size*UNIT_W' L=UNIT_L
21
22   M3 out1 in1 0 0 NMOS W='size*UNIT_W' L=UNIT_L
23   M4 out1 in2 0 0 NMOS W='size*UNIT_W' L=UNIT_L
24 .ENDS
25
26 ********************************************************************
27 *COMPUERTA XOR2:
28 *ENTRADA: in1, in2
29 *SALIDA:  out1
30
31 .SUBCKT Xor2 in1 in2 out1 vsource size = 1
32   Xe1 in1 in1n vsource NotGate size = 1
33   Xe2 in2 in2n vsource NotGate size = 1
34
35   M1 p1 in1 vsource vsource PMOS W='size*UNIT_W' L=UNIT_L
36   M2 p2 in1n vsource vsource PMOS W='size*UNIT_W' L=UNIT_L
37   M3 p1 in2n out1 vsource PMOS W='size*UNIT_W' L=UNIT_L
38   M4 p2 in2 out1 vsource PMOS W='size*UNIT_W' L=UNIT_L
39
40   M5 n1 in1n out1 0 NMOS W='size*UNIT_W' L=UNIT_L
41   M6 n1 in2 out1 0 NMOS W='size*UNIT_W' L=UNIT_L
42   M7 n1 in1 0 0 NMOS W='size*UNIT_W' L=UNIT_L
43   M8 n1 in2n 0 0 NMOS W='size*UNIT_W' L=UNIT_L
44
45 .ENDS
46 ********************************************************************
47
48
49 ********************************************************************
50 *Sumadores
51 ********************************************************************
52
53 ********************************************************************
54 *SUMADOR 1 BIT:
55 *ENTRADAS: in1, in2
56 *ENTRADA CARRY: cin1
57 *SALIDAS: out1
58 *SALIDA CARRY: cout1
59 .SUBCKT Sum1 in1 in2 cin1 out1 cout1 vsource size = 1
60   *xor
61   Xf3 in1 in2 z1 vsource Xor2 size = 1
62   Xf4 z1 cin1 out1 vsource Xor2 size = 1
```

```
63
64   *nand
65   Xf5 z1 cin1 z3 vsource Nand2 size = 1
66   Xf6 in1 in2 z2 vsource Nand2 size = 1
67
68   *not
69   Xf7 z3 z5 vsource NotGate size = 1
70   Xf8 z2 z4 vsource NotGate size = 1
71   Xf10 z6 cout1 vsource NotGate size = 1
72
73   *nor
74   Xf9 z5 z4 z6 vsource Nor2 size = 1
75
76 .ENDS
77
78 ********************************************************************
79 *SUMADOR DE 4 BITS:
80 *ENTRADAS:
81 * valor 1: a3 a2 a1 a0
82 * valor 2: b3 b2 b1 b0
83 *   (Tanto a3 como b3 son los bits m s significativos de cada valor.)
84 * carry 0: k0 (En la gran mayor a de aplicaciones se va a ingresar con
       valor 0.)
85 *SALIDAS:
86 * resultado: s3 s2 s1 s0 (El bit m s significativo es s3.)
87 * carry out: k4
88
89 .SUBCKT Sum4 a3 a2 a1 a0 b3 b2 b1 b0 k0 k4 s3 s2 s1 s0 vsource size = 1
90   Xg11 a0 b0 k0 s0 k1 vsource Sum1 size = 1
91   Xg12 a1 b1 k1 s1 k2 vsource Sum1 size = 1
92   Xg13 a2 b2 k2 s2 k3 vsource Sum1 size = 1
93   Xg14 a3 b3 k3 s3 k4 vsource Sum1 size = 1
94 .ENDS
95
96 ********************************************************************
97 *SUMADOR DE 12 BITS:
98 *ENTRADAS:
99 * valor 1: a11 hasta a00
100 * valor 2: b11 hasta b00
101 * carry 0: k00 (Se recomienda dejar el valor a tierra.)
102 *SALIDAS:
103 * resultado: s11 hasta s00 (s11 es el bit m s significativo del resultado.)
104 * carry out: k12
105
106 .SUBCKT Sum12 a11 a10 a09 a08 a07 a06 a05 a04 a03 a02 a01 a00 b11 b10 b09
       b08 b07 b06
107 +b05 b04 b03 b02 b01 b00 k00 k12 s11 s10 s09 s08 s07 s06 s05 s04 s03 s02 s01
        s00 vsource
108 +size = 1
109   Xh15 a03 a02 a01 a00 b03 b02 b01 b00 k00 k04 s03 s02 s01 s00 vsource Sum4
       size = 1
110   Xh16 a07 a06 a05 a04 b07 b06 b05 b04 k04 k08 s07 s06 s05 s04 vsource Sum4
       size = 1
111   Xh17 a11 a10 a09 a08 b11 b10 b09 b08 k08 k12 s11 s10 s09 s08 vsource Sum4
       size = 1
112 .ENDS
113
114 ********************************************************************
115
```

```
116 ********************************************************************
117 *SUMADOR DE 32 BITS:
118 *ENTRADAS:
119 * valor 1: a31 hasta a00
120 * valor 2: b31 hasta b00
121 * carry 0: k00 (Se recomienda dejar el valor a tierra.)
122 *SALIDAS:
123 * resultado: s11 hasta s00 (s11 es el bit m s significativo del resultado.)
124 * carry out: k32
125
126 .SUBCKT Sum32 a31 a30 a29 a28 a27 a26 a25 a24 a23 a22 a21 a20 a19 a18 a17
      a16 a15
127 +a14 a13 a12 a11 a10 a09 a08 a07 a06 a05 a04 a03 a02 a01 a00 b31 b30 b29 b28
       b27
128 +b26 b25 b24 b23 b22 b21 b20 b19 b18 b17 b16 b15 b14 b13 b12 b11 b10 b09 b08
       b07
129 +b06 b05 b04 b03 b02 b01 b00 k00 k32 s31 s30 s29 s28 s27 s26 s25 s24 s23 s22
       s21
130 +s20 s19 s18 s17 s16 s15 s14 s13 s12 s11 s10 s09 s08 s07 s06 s05 s04 s03 s02
       s01
131 +s00 vsource size = 1
132   * 2 Sumadores de 12 bits
133
134   Xh18 a11 a10 a09 a08 a07 a06 a05 a04 a03 a02 a01 a00 b11 b10 b09 b08 b07
       b06 b05
135   b04 b03 b02 b01 b00 k00 k12 s11 s10 s09 s08 s07 s06 s05 s04 s03 s02 s01
       s00
136   vsource Sum12 size=1
137   Xh19 a23 a22 a21 a20 a19 a18 a17 a16 a15 a14 a13 a12 b23 b22 b21 b20 b19
       b18 b17
138   b16 b15 b14 b13 b12 k12 k24 s23 s22 s21 s20 s19 s18 s17 s16 s15 s14 s13
       s12 vsource
139   Sum12 size=1
140
141   * 2 Sumadores de 4 bits
142
143   Xh20 a27 a26 a25 a24 b27 b26 b25 b24 k24 k28 s27 s26 s25 s24 vsource Sum4
       size = 1
144   Xh21 a31 a30 a29 a28 b31 b30 b29 b28 k28 k32 s31 s30 s29 s28 vsource Sum4
       size = 1
145
146 .ENDS
147
148 ********************************************************************
```

Listing 12.10: decks

12.4.7. subct$_a$*dder*32*bits.sp*

En este deck se encuentran las compuertas lógicas necesarias para armar el *Shif Register* de 64 bits con el que se construirá el multiplicador. Las compuertas incluídas son: NOT, latch tipo D y un Flip Flop tipo D. Las instrucciones para conseguir esto son las siguientes:

```
1 ********************* NOT GATE **************
2 .SUBCKT NotGate in1 out1 vsource size=1
3   M1 out1 in1 0 0 NMOS W='size*UNIT_W' L=UNIT_L
```

```
  M2 out1 in1 vsource vsource PMOS W='2*size*UNIT_W' L=UNIT_L
.ENDS


******************** NEGATIVE LEVEL SENSITIVE D latch ***********
.SUBCKT pDLATCH D CLK Qn2 vsource size=1
  Xa1 CLK CLKn vsource NotGate size = 1
  Xa2 Q2 Qn2 vsource NotGate size = 1
  Xa3 Qn2 Qnn2 vsource NotGate size = 1

  M1 Q2 CLK D 0 NMOS W='size*UNIT_W' L=UNIT_L
  M2 Q2 CLKn D vsource PMOS W='2*size*UNIT_W' L=UNIT_L

  M3 Qnn2 CLKn Q2 0 NMOS W='size*UNIT_W' L=UNIT_L
  M4 Qnn2 CLK Q2 vsource PMOS W='2*size*UNIT_W' L=UNIT_L

.ENDS

******************** POSITIVE LEVEL SENSITIVE D latch ***********

.SUBCKT nDLATCH D CLK Qn1 vsource size=1

  Xb1 CLK CLKn vsource NotGate size = 1
  Xb2 Q1 Qn1 vsource NotGate size = 1
  Xb3 Qn1 Qnn1 vsource NotGate size = 1

  M1 Q1 CLKn D 0 NMOS W='size*UNIT_W' L=UNIT_L
  M2 Q1 CLK D vsource PMOS W='2*size*UNIT_W' L=UNIT_L

  M3 Qnn1 CLKn Q1 0 NMOS W='size*UNIT_W' L=UNIT_L
  M4 Qnn1 CLK Q1 vsource PMOS W='2*size*UNIT_W' L=UNIT_L

.ENDS

***************** CMOS POSITIVE EDGE TRIGGERED D FLIP-FLOP ***********
* Combination of two level-sensitive latches, one negative-sensitive and one
* positive-sensitive in series, "DFFout" is the OUTPUT(Q).
*****************************************************************

.SUBCKT DFF D CLK DFFout vsource size=1

  Xc8 D CLK outNdlatch vsource nDLATCH size=1
  Xc9 outNdlatch CLK DFFout vsource pDLATCH size=1
  Cracing2 outNdlatch 0 100f

.ENDS

*****************************************
*                MULTIPLEXER                *
*****************************************
* Descripci n:
* S = 0 pasa D0
* S = 1 pasa D1

.SUBCKT MUX D0 D1 S Y1 vsource size=1

  Xd1 S Sneg vsource NotGate size = 1
  M1 n1 D0 vsource vsource PMOS W='2*size*UNIT_W' L=UNIT_L
  M2 n1 S Y vsource PMOS W='2*size*UNIT_W' L=UNIT_L
```

```
  M3 Y Sneg n2 0 NMOS W='size*UNIT_W' L=UNIT_L
  M4 n2 D0 0 0 NMOS W='size*UNIT_W' L=UNIT_L

  M5 n3 D1 vsource vsource PMOS W='2*size*UNIT_W' L=UNIT_L
  M6 n3 Sneg Y vsource PMOS W='2*size*UNIT_W' L=UNIT_L
  M7 Y S n4 0 NMOS W='size*UNIT_W' L=UNIT_L
  M8 n4 D1 0 0 NMOS W='size*UNIT_W' L=UNIT_L

  Xd2 Y Y1 vsource NotGate size = 1
  Cracing1 Y1 0 1f

.ENDS

************************************************
*               SHIFT REGISTER                 *
************************************************
* Descripci n:
* SCTRL selecciona si pasa D o hace SHIFT
* si es 0 hace SHIFT, si es 1 pasa D


.SUBCKT SRP SCTRL CLK D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15 D16
     D17 D18
+D19 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D30 D31 D32 D33 D34 D35 D36 D37
     D38 D39
+D40 D41 D42 D43 D44 D45 D46 D47 D48 D49 D50 D51 D52 D53 D54 D55 D56 D57 D58
     D59 D60
+D61 D62 D63 D64 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15 Q16 Q17
    Q18 Q19
+Q20 Q21 Q22 Q23 Q24 Q25 Q26 Q27 Q28 Q29 Q30 Q31 Q32 Q33 Q34 Q35 Q36 Q37 Q38
     Q39 Q40
+Q41 Q42 Q43 Q44 Q45 Q46 Q47 Q48 Q49 Q50 Q51 Q52 Q53 Q54 Q55 Q56 Q57 Q58 Q59
     Q60 Q61
+Q62 Q63 Q64 vsource size=1

  X1 0 D1 SCTRL Y1 vsource MUX size=1
  X2 Y1 CLK Q1 vsource DFF size=1
  X3 Q1 D2 SCTRL Y2 vsource MUX size=1
  X4 Y2 CLK Q2 vsource DFF size=1
  X5 Q2 D3 SCTRL Y3 vsource MUX size=1
  X6 Y3 CLK Q3 vsource DFF size=1
  X7 Q3 D4 SCTRL Y4 vsource MUX size=1
  X8 Y4 CLK Q4 vsource DFF size=1

  X9 Q4 D5 SCTRL Y5 vsource MUX size=1
  X10 Y5 CLK Q5 vsource DFF size=1
  X11 Q5 D6 SCTRL Y6 vsource MUX size=1
  X12 Y6 CLK Q6 vsource DFF size=1
  X13 Q6 D7 SCTRL Y7 vsource MUX size=1
  X14 Y7 CLK Q7 vsource DFF size=1
  X15 Q7 D8 SCTRL Y8 vsource MUX size=1
  X16 Y8 CLK Q8 vsource DFF size=1

  X17 Q8 D9 SCTRL Y9 vsource MUX size=1
  X18 Y9 CLK Q9 vsource DFF size=1
  X19 Q9 D10 SCTRL Y10 vsource MUX size=1
  X20 Y10 CLK Q10 vsource DFF size=1
  X21 Q10 D11 SCTRL Y11 vsource MUX size=1
  X22 Y11 CLK Q11 vsource DFF size=1
```

```
  X23 Q11 D12 SCTRL Y12 vsource MUX size=1
  X24 Y12 CLK Q12 vsource DFF size=1

  X25 Q12 D13 SCTRL Y13 vsource MUX size=1
  X26 Y13 CLK Q13 vsource DFF size=1
  X27 Q13 D14 SCTRL Y14 vsource MUX size=1
  X28 Y14 CLK Q14 vsource DFF size=1
  X29 Q14 D15 SCTRL Y15 vsource MUX size=1
  X30 Y15 CLK Q15 vsource DFF size=1
  X31 Q15 D16 SCTRL Y16 vsource MUX size=1
  X32 Y16 CLK Q16 vsource DFF size=1

  X33 Q16 D17 SCTRL Y17 vsource MUX size=1
  X34 Y17 CLK Q17 vsource DFF size=1
  X35 Q17 D18 SCTRL Y18 vsource MUX size=1
  X36 Y18 CLK Q18 vsource DFF size=1
  X37 Q18 D19 SCTRL Y19 vsource MUX size=1
  X38 Y19 CLK Q19 vsource DFF size=1
  X39 Q19 D20 SCTRL Y20 vsource MUX size=1
  X40 Y20 CLK Q20 vsource DFF size=1

  X41 Q20 D21 SCTRL Y21 vsource MUX size=1
  X42 Y21 CLK Q21 vsource DFF size=1
  X43 Q21 D22 SCTRL Y22 vsource MUX size=1
  X44 Y22 CLK Q22 vsource DFF size=1
  X45 Q22 D23 SCTRL Y23 vsource MUX size=1
  X46 Y23 CLK Q23 vsource DFF size=1
  X47 Q23 D24 SCTRL Y24 vsource MUX size=1
  X48 Y24 CLK Q24 vsource DFF size=1

*28 bits
  X49 Q24 D25 SCTRL Y25 vsource MUX size=1
  X50 Y25 CLK Q25 vsource DFF size=1
  X51 Q25 D26 SCTRL Y26 vsource MUX size=1
  X52 Y26 CLK Q26 vsource DFF size=1
  X53 Q26 D27 SCTRL Y27 vsource MUX size=1
  X54 Y27 CLK Q27 vsource DFF size=1
  X55 Q27 D28 SCTRL Y28 vsource MUX size=1
  X56 Y28 CLK Q28 vsource DFF size=1

*32 bits
  X57 Q28 D29 SCTRL Y29 vsource MUX size=1
  X58 Y29 CLK Q29 vsource DFF size=1
  X59 Q29 D30 SCTRL Y30 vsource MUX size=1
  X60 Y30 CLK Q30 vsource DFF size=1
  X61 Q30 D31 SCTRL Y31 vsource MUX size=1
  X62 Y31 CLK Q31 vsource DFF size=1
  X63 Q31 D32 SCTRL Y32 vsource MUX size=1
  X64 Y32 CLK Q32 vsource DFF size=1

*36 bits
  X65 Q32 D33 SCTRL Y33 vsource MUX size=1
  X66 Y33 CLK Q33 vsource DFF size=1
  X67 Q33 D34 0 Y34 vsource MUX size=1
  X68 Y34 CLK Q34 vsource DFF size=1
  X69 Q34 D35 0 Y35 vsource MUX size=1
  X70 Y35 CLK Q35 vsource DFF size=1
  X71 Q35 D36 0 Y36 vsource MUX size=1
  X72 Y36 CLK Q36 vsource DFF size=1
```

```
175
176 *40 bits
177   X73 Q36 D37 0 Y37 vsource MUX size=1
178   X74 Y37 CLK Q37 vsource DFF size=1
179   X75 Q37 D38 0 Y38 vsource MUX size=1
180   X76 Y38 CLK Q38 vsource DFF size=1
181   X77 Q38 D39 0 Y39 vsource MUX size=1
182   X78 Y39 CLK Q39 vsource DFF size=1
183   X79 Q39 D40 0 Y40 vsource MUX size=1
184   X80 Y40 CLK Q40 vsource DFF size=1
185
186 *44 bits
187   X81 Q40 D41 0 Y41 vsource MUX size=1
188   X82 Y41 CLK Q41 vsource DFF size=1
189   X83 Q41 D42 0 Y42 vsource MUX size=1
190   X84 Y42 CLK Q42 vsource DFF size=1
191   X85 Q42 D43 0 Y43 vsource MUX size=1
192   X86 Y43 CLK Q43 vsource DFF size=1
193   X87 Q43 D44 0 Y44 vsource MUX size=1
194   X88 Y44 CLK Q44 vsource DFF size=1
195
196 *48 bits
197   X89 Q44 D45 0 Y45 vsource MUX size=1
198   X90 Y45 CLK Q45 vsource DFF size=1
199   X91 Q45 D46 0 Y46 vsource MUX size=1
200   X92 Y46 CLK Q46 vsource DFF size=1
201   X93 Q46 D47 0 Y47 vsource MUX size=1
202   X94 Y47 CLK Q47 vsource DFF size=1
203   X95 Q47 D48 0 Y48 vsource MUX size=1
204   X96 Y48 CLK Q48 vsource DFF size=1
205
206 *52 bits
207   X97 Q48 D49 0 Y49 vsource MUX size=1
208   X98 Y49 CLK Q49 vsource DFF size=1
209   X99 Q49 D50 0 Y50 vsource MUX size=1
210   X100 Y50 CLK Q50 vsource DFF size=1
211   X101 Q50 D51 0 Y51 vsource MUX size=1
212   X102 Y51 CLK Q51 vsource DFF size=1
213   X103 Q51 D52 0 Y52 vsource MUX size=1
214   X104 Y52 CLK Q52 vsource DFF size=1
215
216 *56 bits
217   X105 Q52 D53 0 Y53 vsource MUX size=1
218   X106 Y53 CLK Q53 vsource DFF size=1
219   X107 Q53 D54 0 Y54 vsource MUX size=1
220   X108 Y54 CLK Q54 vsource DFF size=1
221   X109 Q54 D55 0 Y55 vsource MUX size=1
222   X110 Y55 CLK Q55 vsource DFF size=1
223   X111 Q55 D56 0 Y56 vsource MUX size=1
224   X112 Y56 CLK Q56 vsource DFF size=1
225
226 *60 bits
227   X113 Q56 D57 0 Y57 vsource MUX size=1
228   X114 Y57 CLK Q57 vsource DFF size=1
229   X115 Q57 D58 0 Y58 vsource MUX size=1
230   X116 Y58 CLK Q58 vsource DFF size=1
231   X117 Q58 D59 0 Y59 vsource MUX size=1
232   X118 Y59 CLK Q59 vsource DFF size=1
233   X119 Q59 D60 0 Y60 vsource MUX size=1
```

```
  X120 Y60 CLK Q60 vsource DFF size=1

*64 bits
  X121 Q60 D61 0 Y61 vsource MUX size=1
  X122 Y61 CLK Q61 vsource DFF size=1
  X123 Q61 D62 0 Y62 vsource MUX size=1
  X124 Y62 CLK Q62 vsource DFF size=1
  X125 Q62 D63 0 Y63 vsource MUX size=1
  X126 Y63 CLK Q63 vsource DFF size=1
  X127 Q63 D64 0 Y64 vsource MUX size=1
  X128 Y64 CLK Q64 vsource DFF size=1

.ENDS
```

Listing 12.11: decks

12.5. Manual de inicio rápido

En esta sección del anexo se encuentra una copia de lo contenido en el manual de inicio rápido que fue creado a raíz de uno de los objetivos del presente trabajo.

12.5.1. Introducción

El presente documento tiene como objetivo servir de guía para las futuras promociones del departamento de Ingeniería electrónica, mecatrónica y biomédica que trabajen en la línea de investigación de nanoelectrónica. En este se incluyen una serie de análisis que se pueden realizar en *PrimeSim* HSPICE con el objetivo que se pueda analizar el correcto funcionamiento de un circuito diseñado.

La etapa de simulación es de suma importancia dentro del flujo de diseño de un chip, ya que permite verificar que el diseño final tenga el comportamiento esperado antes de que este sea enviado a fabricar. Por medio de la detección temprana de errores que afecten la funcionalidad del circuito o su rendimiento.

12.5.2. Ejecución de un archivo demo en HSPICE

Sección de ayuda

Se puede acceder a la documentación de *PrimeSim* HSPICE en formato PDF usando el siguiente comando:

```
% hspice -doc
```

También se puede acceder a la documentación en línea de la plataforma por medio de un buscador usando formato HTML por medio del comando:

```
% hspice -help
```

Archivos de demostración

Al momento de instalar *PrimeSim HSPICE*, se pueden localizar archivos de demostración en la siguiente dirección:

```
$installdir/demo/hspice/
```

Donde *$installdir* corresponde al directorio donde se instaló *PrimeSim HSPICE*. Dentro de esta carpeta se encuentran los directorios mostrados en el cuadro 15. La forma de ejecutar estos ejemplos depende del sistema operativo que se esté utilizando. Si se está en el ambiente de Windows, se debe utilizar *PrimeSim HSPICE User Interface* (HSPUI). Mientras que, si se está utilizando en Linux, se debe ejecutar el siguiente comando desde la terminal:

```
%> hspice -i archivo_demo_seleccionado -o archivo_salida
```

Directorio	Descripción
/spiceADE_integ	Tutorial de integración de PrimeSim HSPICE con Cadence® Virtuoso® Analog Design Environment
/apps	Aplicaciones generales
/back_annotation	Utilización de las BA_options
/behave	Componentes de comportamiento analógico
/bench	Pruebas de rendimiento estándar
/bisect	Optimización de bisección
/bjt	Componentes bipolares
/cchar	Características de celdas prototipo
/ciropt	Optimización a nivel de circuito
/devopt	Optimización a nivel de dispositivos
/encryption	Encriptación tradiciones, 8-bits y 3DES
/fft	Análisis de Fourier
/filters	Filtros
/ibis	Ejemplos de IBIS
/lstb	Análisis de estabilidad de retroalimentación
/mag	Transformadores y componentes de núcleo magnético
/mos	Componentes MOS
/rf_examples	Ejemplos de circuitos RF
/si	Aplicaciones de integridad de señal
/sources	Fuentes de alimentación dependientes e independientes
/sparam	Aplicaciones de parámetro-S
/tline	Filtros y líneas de transmisión
/twline	Líneas de transmisión de elementos W y solucionadores de campo.
/variability	Bloque de variación, Monte Carlo, y ejemplos de discordancia AC/DC
/veriloga	Ejemplos de Verilog-A

Cuadro 15: Árbol de directorios de archivos de demostración proporcionados por PrimeSim.

Invocar simulación de HSPICE

Al momento de invocar una simulación de *HSPICE*, *PrimeSim HSPICE* ejecuta los siguientes pasos:

1. Se realiza la invocación de los archivos de entrada y salida. Por ejemplo, al ejecutar el comando:

   ```
   %> hspice -i demo.sp -o salida.lis
   ```

   se invoca la herramienta en el archivo demo.sp y se direccionan las salidas de la simulación al archivo salida.lis

2. La herramienta de *PrimeSim HSPICE* determina la mejor arquitectura de ejecucion y determina la versión de la herramienta.

3. La herramienta de *PrimeSim HSPICE* lee las varaibles de ambiente relacionadas con la licencia del producto y verifica la validez de estas. De no ser autorizada la licencia, se cancela la ejecución de la simulación.

4. *PrimeSim HSPICE* lee el correspondiente archivo *hspice.ini*

5. Se abre el *netlist* de entrada (siguiendo el ejemplo inicial seria demo.sp), si este archivo no existe se genera error. Seguido de esto, se abre el arhivo de salida (salida.lis).

6. Se abren los archivos que fueron incluidos en el *netlist* de entrada mediante la instrucción .INCLUDE o .LIB

7. Se realiza la lectura de las condiciones iniciales definidas con las instrucciones .IC y .NODESET. Además, se encuentra un punto de operación cuya inforación se puede guardar usando el comando .SAVE

8. Se ejecuta el barrido especificado de diseño y produce un conjunto de archivos de salida.

9. Mediante la utilización del comando .ALTER, se pueden alterar las condiciones de la simulación y repetir análisis.

10. Mediante la combinación de las teclas Ctrl + Z se puede suspender la ejecución de un análisis usando *PrimeSim HSPICE*.

11. Una vez finalizada la simulación, *PrimeSim HSPICE* cierra todos los archivos abiertos.

12.5.3. Análisis DC

El análisis DC consiste en realizar un barrido de voltaje en un nodo determinado de un circuito; y observar como varía el comportamiento de voltaje, corriente o potencia en otro nodo de este circuito. Para este caso, se muestra un análisis DC de un inversor (NOT) CMOS con el objetivo de encontrar la región prohibida del circuito.

Inversor (NOT) CMOS

1. Para realizar este ejemplo, se debe usar la carpeta Quickstart brindada por el encargado de laboratorio.

2. Dentro de la carpeta de Quickstart, se debe ingresar a la carpeta ./Not_DC

3. En esta carpeta se encontrará el archivo quickDC.sp el cual contiene el archivo en HSPICE donde se realiza un análisis DC de una NOT CMOS.

4. Al abrir el archivo mediante un editor de texto, se mostrará el siguiente código:

```
*CIRCUITO INVERSOR

*Importacion de los modelos de MOSFET
* El LEVEL=1 representa a un transistor hecho a partir de los modelos de
* Schichman-Hodges
.MODEL PCH PMOS LEVEL=1
.MODEL NCH NMOS LEVEL=1

*Construccion del NOT CMOS
M1 VCC IN OUT VCC PCH L = 1U W = 20U
M2 OUT IN 0 0 NCH L = 1U W = 20U
VCC VCC 0 5
VIN IN 0 0 0 *Nodo al que se le hace el barrido DC
CLOAD OUT 0 750f

*Analisis DC
* Se realiza un barrido DC de 0V a 5V con un incremento
* de 0.1V en cada iteracion
.DC VIN 0 5 0.1
.PRINT DC  V(IN) V(OUT)

.OPTION POST

.END
```

5. Para ejecutar el programa se debe abrir la terminal de linux en el directorio donde está el archivo quickDC.sp y ejecutar la siguiente línea de comando:

```
hspice -i quickDC.sp -o quickDC.lis
```

Donde quickDC.sp corresponde al archivo en HSPICE que será simulado mediante PrimeSim HSPICE, y el archivo quickDC.lis corresponde al nombre del archivo que será creado por la herramienta para documentar la simulación.

6. Una vez ejecutado el comando, se generarán los siguientes archivos:
   a) **quickDC.lis:** Este archivo corresponde a los mensajes y mediciones realizados por PrimeSim HSPICE durante la ejecución de la simulación.
   b) **quickDC.st0:** Corresponde al estado de las salidas de la simulación.
   c) **quickDC.sw0:** Corresponde a los resultados del análisis DC realizado.

Análisis de programa

La parte del código donde se realiza el análisis DC del inversor, consta de dos instrucciones. Una para hacer el barrido de voltaje DC y la segunda para imprimir los valores de voltaje en los nodos que se desean estudiar.

Para realizar el análisis de barrido DC, se debe usar la instrucción .DC, la cual posee la siguiente sintaxis:

```
.DC Nodo_a_variar_voltaje Voltaje_inicio Voltaje_final Variación_voltaje
```

Por lo que, en el programa al emplear la instrucción:

```
.DC VIN 0 5 0.1
```

Se está realizando un barrido de voltaje en el nodo VIN, que corresponde al voltaje de entrada del circuito, comenzando en 0V y terminando en 5V, con un incremento de voltaje de 0.1V por cada iteración, obteniendo un total de 50 iteraciones.

Seguido de esto, es necesaria la utilización de una instrucción para mostrar los voltajes obtenidos durante las iteraciones en los nodos que se están estudiando. Para ello, se utiliza la instrucción .PRINT. Esta instrucción posee la siguiente sintaxis:

```
.PRINT tipo_análisis V(nodo_1) V(nodo_2) ...
```

Por tanto, al utilizar la instrucción:

```
.PRINT DC  V(IN) V(OUT)
```

Se mostrará el voltaje obtenido gracias al análisis DC en los nodos IN y OUT.

Finálmente, se debe utilizar la siguiente instrucción para guardar los resultados de la simulación y poderlos visualizar mediante una interfaz gráfica, como lo podría ser WaveView.

```
.OPTION POST
```

Análisis gráfico

Para analizar este ejemplo se utilizará la herramienta WaveView. Para hacer uso de ella, se deben seguir los siguientes pasos:

1. Invocar la terminal desde el directorio donde se generaron los archivos de la simulación en PrimeSim.

2. Se debe ejecutar la siguiente instrucción en el terminal:

```
wv -i quickDC.sw0
```

   Al ejecutar este comando se invocará la herramienta de WaveView y mostrará la siguiente ventana. Para cargar los datos de la simulación es necesario hacer click en la región dentro del círculo rojo 40.

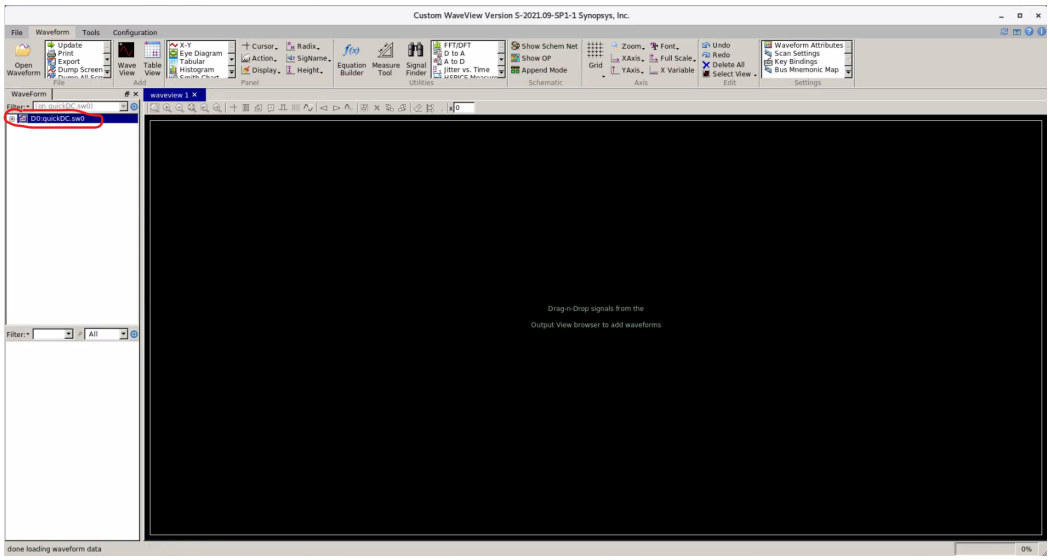


Figura 40: Vista inicial de la herramienta WaveView.

   Esto desplegará una nueva pestaña llamada *toplevel* la cual contiene los voltajes y corrientes medidos en la simulación. Se debe hacer click sobre esta pestaña para que en el recuadro inferior se muestren las opciones para cargar los datos gráficos de la medición 41.

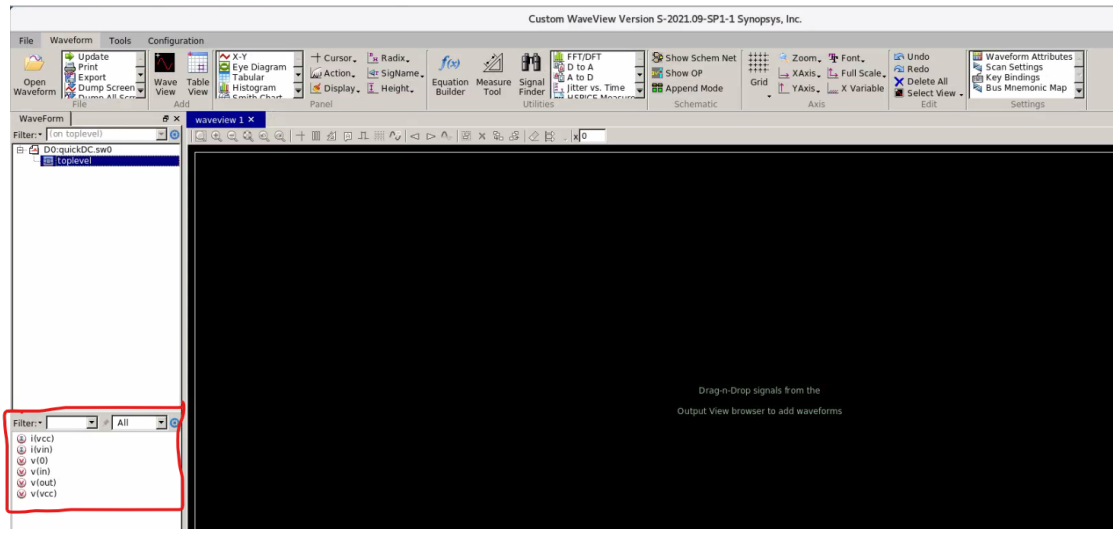


Figura 41: Vista de la herramienta WaveView con los filtros de las mediciones realizadas en la simulación.

Para cargar las gráficas que se desean observar, se debe hacer click sobre el elemento que se desea observar. Para este caso, se mostrará el voltaje en el nodo de salida (out) 42.

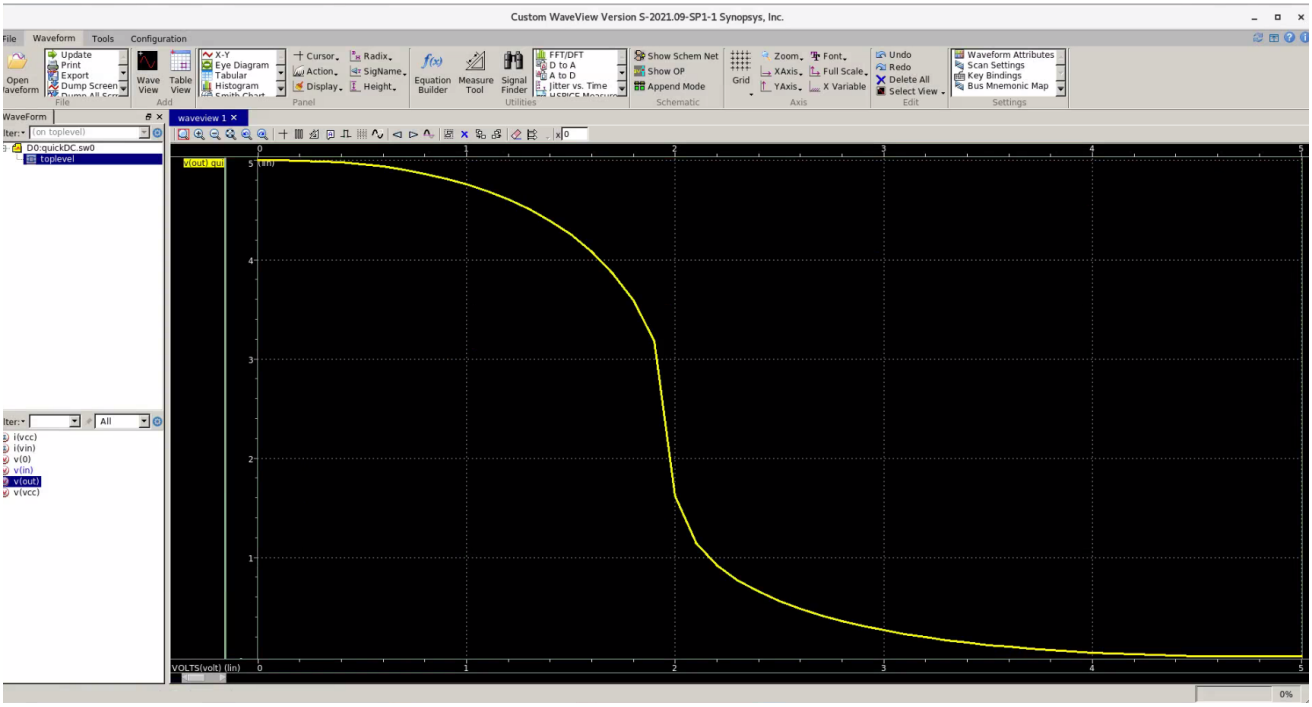


Figura 42: Vista de la herramienta WaveView con los filtros de las mediciones realizadas en la simulación.

Con esto se cargarán los datos del análisis DC almacenados en el archivo quickDC.sw0 en WaveView, el cual permitirá observar los resultados del análisis en un entorno gráfico. Lo que el gráfico de la Figura 42 está mostrando es la zona prohibida de la compuerta

NOT. Gracias a esto, se puede mapear que valores de voltaje son válidos en la entrada de la compuerta, para que esta genere una correcta señal de salida.

12.5.4. Análisis AC

El análisis AC permite realizar un barrido de frecuencia y observar el comportamiento de un circuito con base en esta variación. Para ejemplificar este análisis, se realiza la construcción de un circuito RC al cual se le realizará un barrido de frecuencia de 10 puntos por década desde 1KHz hasta 1MHz.

Circuito RC

1. Para realizar este ejemplo, se debe usar la carpeta Quickstart brindada por el encargado de laboratorio.
2. Dentro de la carpeta de Quickstart, se debe ingresar a la carpeta ./RC_AC
3. En esta carpeta se encontrará el archivo quickAC.sp, el cual contiene el código en HSPICE donde se realiza el análisis AC de un circuito RC.
4. Al abrir el archivo mediante un editor de texto, se mostrará el siguiente código:

```
*Circuito RC

*Construccion del circuito
V1 1 0 10 AC 1
R1 1 2 1K
R2 2 0 1K
C1 2 0 1P

*Analisis AC
.OPTION LIST NODE POST
.OP
.AC DEC 10 1K 1MEG
.PRINT AC V(1) V(2) I(R2) I(C1)

.END
```

5. Para ejecutar el programa se debe abrir la terminal de linux en el directorio donde está el archivo quickAC.sp y ejecutar la siguiente línea de comando:

```
hspice -i quickAC.sp -o quickAC.lis
```

   Donde quickAC.sp corresponde al archivo en HSPICE que será simulado mediante PrimeSim HSPICE, y el archivo quickAC.lis corresponde al nombre del archivo que será creado por la herramienta para documentar la simulación.

6. Una vez ejecutado el comando, se generarán los siguientes archivos:

   a) **quickAC.lis:** Este archivo corresponde a los mensajes y mediciones realizados por PrimeSim HSPICE durante la ejecución de la simulación.

   b) **quickAC.st0:** Corresponde al estado de las salidas de la simulación.

   c) **quickAC.ac0:** Corresponde a los resultados del análisis AC realizado.

   d) **quickAC.ic0:** Corresponde a los valores iniciales de voltaje de operación en los nodos especificados.

Análisis gráfico

Para analizar este ejemplo se utilizará la herramienta WaveView. Para hacer uso de ella, se deben seguir los siguientes pasos:

1. Invocar la terminal desde el directorio donde se generaron los archivos de la simulación en PrimeSim.

2. Se debe ejecutar la siguiente instrucción en el terminal:

```
wv -i quickAC.ac0
```

3. Una vez se esté dentro la interfaz gráfica de WaveView, se deben ver los datos que son de interés. Para este ejemplo, es el voltaje en la resistencia R2 43.

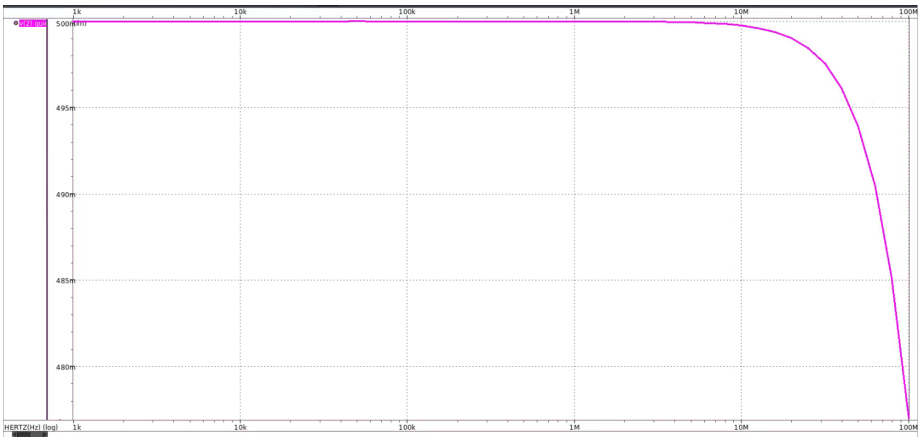


Figura 43: Resultados de la simulación.

Para este ejemplo, se diseñó un circuito RC y gracias a la simulación se pudo determinar que funciona como filtro pasa bajas. Gracias a esto, se puede determinar la frecuencia máxima de la señal en el nodo de entrada con la que el circuito funciona correctamente.

Análisis de programa

Para este ejemplo, la parte donde se realiza el análisis AC, consta de 4 líneas. La primera línea consiste en la configuración de los resultados obtenidos en la simulación. Para ello, se utiliza la instrucción

```
.OPTION LIST NODE POST
```

Esta instrucción tiene tres funciones. La primera consiste en Mostrar una tabla de nodos de referencia cruzada, la segunda consiste en imprimir el listado de elementos, nodos, conexiones realizadas en el circuito; y la tercera consiste en guardar los resultados de la simulación para que puedan ser analizados en un entorno gráfico.

La segunda instrucción utilizada es la siguiente:

```
.OP
```

La finalidad de esta es calcular el punto de operación DC del circuito.

La tercera instrucción consiste en realizar el análisis AC. Para ello se utiliza la instrucción .AC, la cual posee la siguiente sintaxis:

```
.AC tipo salto freq_ini freq_fin
```

Por lo que, al utilizar la instrucción:

```
.AC DEC 10 1K 1MEG
```

Se está configurando un análisis AC para realizar un barrido de frecuencia de 10 puntos por década desde 1KHz hasta 1MHz.

Ya obtenidos los resultados de la simulación AC, resulta necesaria la utilización de la instrucción .PRINT para mostrar los valores de interes. Para ello se utiliza la siguiente instrucción:

```
.PRINT AC V(1) V(2) I(R2) I(C1)
```

Esta instrucción permite mostrar los voltajes obtenidos mediante el análisis AC en el nodo 1 y 2 y la corriente que fluye por R2 y C1.

Circuito RC usando Monte Carlo

1. Para realizar este ejemplo, se debe usar la carpeta Quickstart brindada por el encargado de laboratorio.

2. Dentro de la carpeta de Quickstart, se debe ingresar a la carpeta ./AC_MONTE

3. En esta carpeta se encontrará el archivo quickAC.sp, el cual contiene el código en HSPICE donde se realiza el análisis AC de un circuito RC.

4. Al abrir el archivo mediante un editor de texto, se mostrará el siguiente código:

```
*Circuito RC

*Parámetro que será variado
.PARAM res_val=AUNIF(10000,9000)
*Construccion del circuito
V1 1 0 50 AC 50
R1 1 2 1K
R2 2 0 res_val
C1 2 0 1P

*Analisis AC VARIACION MONTECARLO
.OPTION LIST NODE POST
.OP
.AC DEC 10 1K 100MEG SWEEP MONTE=50 * Se configuran 50 iteraciones
.PRINT AC V(1) V(2) I(R2) I(C1)

.END
```

5. Para ejecutar el programa se debe abrir la terminal de linux en el directorio donde está el archivo quickAC.sp y ejecutar la siguiente línea de comando:

```
hspice -i quickAC.sp -o quickAC.lis
```

   Donde quickAC.sp corresponde al archivo en HSPICE que será simulado mediante PrimeSim HSPICE, y el archivo quickAC.lis corresponde al nombre del archivo que será creado por la herramienta para documentar la simulación.

6. Una vez ejecutado el comando, se generarán los siguientes archivos:

   a) **quickAC.lis:** Este archivo corresponde a los mensajes y mediciones realizados por PrimeSim HSPICE durante la ejecución de la simulación.

   b) **quickAC.st0:** Corresponde al estado de las salidas de la simulación.

   c) **quickAC.ac0:** Corresponde a los resultados del análisis AC realizado. En este archivo se encontrarán la simulación de Monte Carlo mediante el análisis AC.

   d) **quickAC.ic0:** Corresponde a los valores iniciales de voltaje de operación en los nodos especificados.

Análisis gráfico

Para analizar este ejemplo se utilizará la herramienta WaveView. Para hacer uso de ella, se deben seguir los siguientes pasos:

1. Invocar la terminal desde el directorio donde se generaron los archivos de la simulación en PrimeSim.
2. Se debe ejecutar la siguiente instrucción en el terminal:

```
wv -i quickAC.ac0
```

3. Una vez se esté dentro la interfaz gráfica de WaveView, se deben ver los datos que son de interés. Para este ejemplo, es el voltaje en la resistencia R2 como se muestra en la Figura 44.

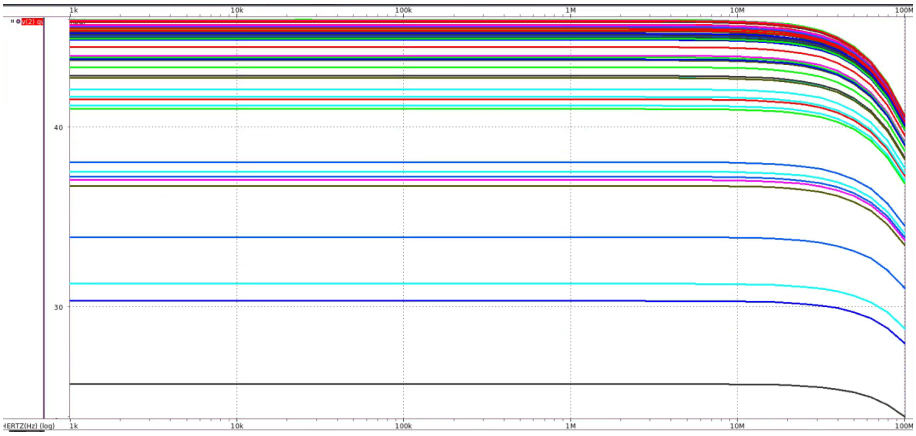


Figura 44: Resultados de la simulación.

Para este ejemplo, se diseñó un circuito RC y gracias a la simulación se pudo determinar que funciona como filtro pasa bajas. Mediante el análisis de Monte Carlo, se pudo observar como se comportaría el circuito al realizar variaciones en el valor de la resistencia R2. Al variar el valor de dicha resistencia, la cual se encuentra en paralelo al capacitor, se está variando también la distribución del divisor de voltaje. Es por ello que en la gráfica de la Figura 44 se muestra como el voltaje de operación en el nodo 2 varía en cada una de las 50 iteraciones realizadas en el análisis de Monte Carlo. Dado que R2 forma parte del divisor de voltaje, mientras más grande sea este valor, mayor será el voltaje presente en el nodo 2.

Análisis de programa

Este deck inicia una la declaración de un parámetro bajo el nombre de res_val". A este parámetro se le aplicará variación de Monte Carlo para que en cada iteración el parámetro tenga un valor distinto. La forma en como se asginará valor en cada iteración es por medio de una distribución uniforme con variación absoluta cuyo valor central se encuentra en 10000 y su magnitud de variación equivale a 9000. Esto se logra mediante la instrucción:

```
.PARAM res_val=AUNIF(10000,9000)
```

Seguido de esto, se realiza la construcción del circuito RC que será analizado mediante *PrimeSim* HSPICE. Para lograr la variación del valor de la resistencia, es necesario asignar el parámetro res_val.ªl valor de resistividad de la resistencia que se desea variar en cada iteración. Ya que para este ejemplo se pretende variar la resistencia que conecta el nodo 2 con tierra, se usa la siguiente instrucción:

```
R2 2 0 res_val
```

Para este ejemplo, la parte donde se realiza el análisis AC, consta de 4 líneas. La primera línea consiste en la configuración de los resultados obtenidos en la simulación. Para ello, se utiliza la instrucción

```
.OPTION LIST NODE POST
```

Esta instrucción tiene tres funciones. La primera consiste en Mostrar una tabla de nodos de referencia cruzada, la segunda consiste en imprimir el listado de elementos, nodos, conexiones realizadas en el circuito; y la tercera consiste en guardar los resultados de la simulación para que puedan ser analizados en un entorno gráfico.

La segunda instrucción utilizada es la siguiente:

```
.OP
```

La finalidad de esta es calcular el punto de operación DC del circuito.

La tercera instrucción consiste en realizar el análisis AC usando Monte Carlo. Para ello se utiliza la instrucción .AC, la cual posee la siguiente sintaxis:

```
.AC tipo salto freq_ini freq_fin SWEEP MONTE=num_iter
```

Por lo que, al utilizar la instrucción:

```
.AC DEC 10 1K 100MEG SWEEP MONTE=50
```

Se está configurando un análisis AC para realizar un barrido de frecuencia de 10 puntos por década desde 1KHz hasta 1MHz. Adicionalmente, se establece que se realizarán 50 iteraciones del análisis de Monte Carlo, tomando en cuenta las variaciones a los parámetros definidos anteriormente.

Ya obtenidos los resultados de la simulación AC, resulta necesaria la utilización de la instrucción .PRINT para mostrar los valores de interes. Para ello se utiliza la siguiente instrucción:

```
.PRINT AC V(1) V(2) I(R2) I(C1)
```

Esta instrucción permite mostrar los voltajes obtenidos mediante el análisis AC en el nodo 1 y 2 y la corriente que fluye por R2 y C1.

12.5.5. Análisis transitorio

El análisis transitorio permite estudiar el comportamiento de un circuito con el paso del tiempo. Para ejemplificar este análisis, se estudiará como cambia la señal de salida de un circuito NOT, cuya entrada esta oscilando.

NOT CMOS

1. Para realizar este ejemplo, se debe usar la carpeta Quickstart brindada por el encargado de laboratorio.

2. Dentro de la carpeta de Quickstart, se debe ingresar a la carpeta ./Not_tran

3. En esta carpeta se encontrará el archivo quickINV.sp, el cual contiene el código en HSPICE donde se realiza el análisis transitorio de una NOT.

4. Al abrir el archivo mediante un editor de texto, se mostrará el siguiente código:

```
*Circuito NOT

*Importacion de los modelos de MOSFET
.MODEL PCH PMOS LEVEL=1
.MODEL NCH NMOS LEVEL=1

*Construccion del circuito
M1 VCC IN OUT VCC PCH L = 1U W = 20U
M2 OUT IN 0 0 NCH L = 1U W = 20U
VCC VCC 0 5
VIN IN 0 0 PULSE .2 4.8 2N 1N 1N 5N 20N
CLOAD OUT 0 .75P

*Analisis transiente
.OPTION LIST NODE POST
.TRAN 200P 20N SWEEP TEMP -55 75 10
.PRINT TRAN V(IN) V(OUT)

.END
```

5. Para ejecutar el programa se debe abrir la terminal de linux en el directorio donde está el archivo quickDC.sp y ejecutar la siguiente línea de comando:

```
hspice -i quickINV.sp -o quickINV.lis
```

   Donde quickINV.sp corresponde al archivo en HSPICE que será simulado mediante PrimeSim HSPICE, y el archivo quickinv.lis corresponde al nombre del archivo que será creado por la herramienta para documentar la simulación.

6. Una vez ejecutado el comando, se generarán los siguientes archivos:

   a) **quickINV.lis:** Este archivo corresponde a los mensajes y mediciones realizados por PrimeSim HSPICE durante la ejecución de la simulación.

   b) **quickINV.st0:** Corresponde al estado de las salidas de la simulación.

   c) **quickINV.tr0:** Corresponde a los resultados del análisis transitorio realizado.

   d) **quickINV.ic0:** Corresponde a los valores iniciales de voltaje de operación en los nodos especificados.

Análisis gráfico

Para analizar este ejemplo se utilizará la herramienta WaveView. Para hacer uso de ella, se deben seguir los siguientes pasos:

1. Invocar la terminal desde el directorio donde se generaron los archivos de la simulación en PrimeSim.

2. Se debe ejecutar la siguiente instrucción en el terminal:

```
wv -i quickINV.tr0
```

3. Una vez se esté dentro la interfaz gráfica de WaveView, se deben ver los datos que son de interés. Para este ejemplo, son el voltaje de entrada y salida 45.

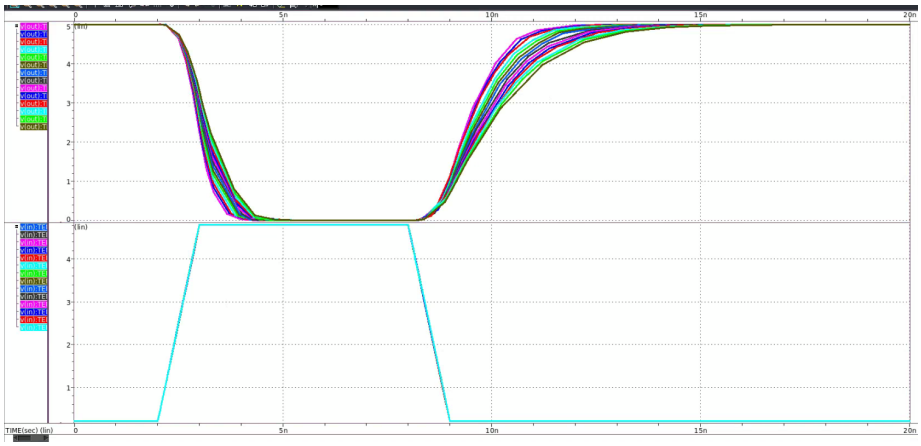


Figura 45: Resultados de la simulación.

Para este ejemplo, dado que se utilizó un SWEEP para hacer un barrido de temperatura en conjunto con el análisis transitorio, al momento de incluir el voltaje en el nodo de entrada y salida, se incluyen los datos obtenidos de cada simulación de temperatura diferente; razón por la cual, se ven varias gráficas.

Análisis de programa

Para este ejemplo, la parte donde se realiza el análisis transitorio, consta de 3 líneas. La primera línea consiste en la configuración de los resultados obtenidos en la simulación. Para ello, se utiliza la instrucción:

```
.OPTION LIST NODE POST
```

Esta instrucción tiene tres funciones. La primera consiste en Mostrar una tabla de nodos de referencia cruzada, la segunda consiste en imprimir el listado de elementos, nodos, conexiones realizadas en el circuito; y la tercera consiste en guardar los resultados de la simulación para que puedan ser analizados en un entorno gráfico.

La segunda instrucción consiste en realizar el análisis transitorio. Para lograr esto se utiliza la instrucción .TRAN, la cual posee la siguiente sintaxis:

```
.TRAN t_salto t_final
```

Esta instrucción tiene una serie de variaciones para hacer análisis más complejos. Uno de ellos es analizar la respuesta transitoria de un circuito dependiendo la temperatura de este. Para lograr esto, se debe usar la instrucción .TRAN con la siguiente sintaxis:

```
.TRAN t_salto t_final SWEEP TEMP temp_inicio temp_final variacion_temp
```

Para este ejemplo se utiliza esta instrucción de la siguiente manera:

```
.TRAN 200P 20N SWEEP TEMP -55 75 10
```

Esto significa que se realizará un análisis transitorio iniciando en de 0ns a 20ns con un período de muestreo de 200ps, y se realizará un barrido de temperatura empezando en -55°C y terminando en 75°C, con incrementos de 10°C en cada iteración.

Una vez conseguidos los datos de la simulación, se utiliza la instrucción .PRINT para mostrar los datos que se están estudiando. Para ello se utiliza la siguiente instrucción:

```
.PRINT TRAN V(IN) V(OUT)
```

Esta instrucción permite mostrar los voltajes obtenidos mediante el análisis transitorio en el nodo IN y OUT.

12.5.6. Análisis de *Ring oscillator*

PrimeSim HSPICE cuenta con un análisis de *Ring Oscillator* incluido dentro de su conjunto de instrucciones. Este análisis se obtiene al invocar la instrucción .SNOSC, la cual, utiliza el método numérico de disparo de Newton para obtener una aproximación de la frecuencia de oscilación del circuito. Cabe recalcar, que para utilizar este análisis es necesario definir un punto de inicio lo suficientemente cercano al valor real de oscilación para que el método converja.

Análisis de *Ring Oscillator* de 7 NOT

1. Para realizar este ejemplo, se debe usar la carpeta Quickstart brindada por el encargado de laboratorio.
2. Dentro de la carpeta de Quickstart, se debe ingresar a la carpeta ./Ringosc_Newt
3. En esta carpeta se encontrará el archivo ringoscSN.sp, el cual contiene el código en HSPICE donde se realiza el análisis de *Ring Oscillator* a un oscilador compuesto por 7 compuertas NOT.
4. Al abrir el archivo mediante un editor de texto, se mostrará el siguiente código:

```
*Comentario inicial
.title ringosc

*Subcircuito de la compuerta NOT
.subckt inv in out vdd
 mn1 out in 0 0 nmos l=0.25u w=2u
 mp1 out in vdd vdd pmos l=0.25u w=6u
.ends

* CONSTRUCCION DEL RING OSCILLATOR DE 7 NOT
vdd vdd 0 3
x1 1 2 vdd inv
x2 2 3 vdd inv
x3 3 4 vdd inv
x4 4 5 vdd inv
x5 5 6 vdd inv
x6 6 7 vdd inv
x7 7 1 vdd inv
* CAPACITANCIA PARASITA
c1 1 0 0.022p

* INCLUSION DE LIBRERIA DE 180 nm
.lib 'tsmc018.m' tt

* CONDICION DE INICIO
.ic v(1)=3
```

```
.options post
.option measdgt=10

* ANALISIS DE RING OSCILLATOR
.snosc tones=1.7e9 nharms=10 oscnode=1 trinit=10n
.phasenoise v(7) dec 10 100 10meg listfreq=(all) listcount=5 listsources=on
.probe phasenoise phnoise v(7)
.print phasenoise phnoise v(7)

.probe sn v(1)
.probe snfd v(1) v(7)

* MEDICION DE LA FRECUENCIA
.measure snfd freqsnfd1 find 'HERTZ[1]' at=0.01p

.end
```

5. Para ejecutar el programa se debe abrir la terminal de linux en el directorio donde está el archivo quickDC.sp y ejecutar la siguiente línea de comando:

```
hspice -i ringoscSN.sp -o ringoscSN.lis
```

   Donde ringoscSN.sp corresponde al archivo en HSPICE que será simulado mediante PrimeSim HSPICE, y el archivo ringoscSN.lis corresponde al nombre del archivo que será creado por la herramienta para documentar la simulación.

6. Una vez ejecutado el comando, se generarán los siguientes archivos:

   a) **ringoscSN.lis:** Este archivo corresponde a los mensajes y mediciones realizados por PrimeSim HSPICE durante la ejecución de la simulación.

   b) **ringoscSN.sn0:** Corresponde al archivo de simulación en el dominio del tiempo.

   c) **ringoscSN.snf0:** Corresponde al archivo de simulación en el dominio de la frecuencia.

   d) **ringoscSN.snpn:** Corresponde al archivo de simulación del análisis de ruido de fase.

Glosario

características de transferencia: Describen el voltaje de salida de un circuito en función de su voltaje de entrada. 54

core: Parte central del *layout* de un circuito. Corresponde al lugar donde se colocan las celdas de este. 3

FPGA: Dispositivo electrónico programable básado en bloques de lógica cuya interconexión y funcionalidad pueden ser configuradas en poco tiempo mediante el uso de un lenguaje descriptor de *hardware*. 3

HDL: Hace referencia a lenguaje descriptor de *hardware*. Permite la descripción de la estructura y el comportamiento de circuitos electrónicos. 3

layout: Representación del circuito en términos de figuras geométricas planas. 2

nanochip: Circuito integrado muy pequeño cuya escala de medición se encuentra en el orden de los nanómetros. 1

netlist: Archivo que contiene los componentes y conexiones de un circuito a nivel de nodos. 3

Splashtop: Software que permite acceso remoto a computadoras. 5

zona prohibida: Rango de voltaje de entrada en la que el circuito genera un voltaje de salida incorrecto. 54