

UNIVERSIDAD DEL VALLE DE GUATEMALA  
Facultad de Ingeniería



**Diseño de un circuito integrado con tecnología 180 nm usando librerías de diseño de TSMC: Automatización de la etapa de síntesis lógica y creación de archivos Verilog para pruebas físicas en un FPGA *Genesys Xilinx Virtex-5 LX50T* y automatización de la verificación *extracción de parásitos*.**

Trabajo de graduación presentado por Allison Estuardo Aguilar Chocooj 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 180 nm usando librerías de diseño de TSMC: Automatización de la etapa de síntesis lógica y creación de archivos Verilog para pruebas físicas en un FPGA *Genesys Xilinx Virtex-5 LX50T* y automatización de la verificación *extracción de parásitos*.**

Trabajo de graduación presentado por Allison Estuardo Aguilar Chocooj para optar al grado académico de Licenciado en Ingeniería Electrónica

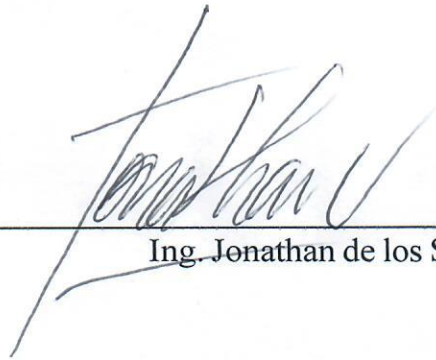
Guatemala,

2023



Vo.Bo.:

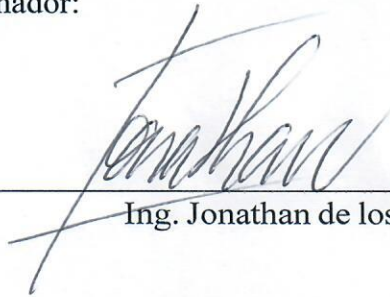
(f)



Ing. Jonathan de los Santos

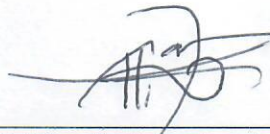
Tribunal Examinador:

(f)



Ing. Jonathan de los Santos

(f)



MSc. Carlos Esquit

(f)



Ing. Luis Nájera

Fecha de aprobación: Guatemala, 20 de Junio de 2023.





El método de enseñanza se orienta a la investigación y el desarrollo de tecnologías que buscan solucionar problemas de la población. La ingeniería electrónica que se imparte en la Universidad del Valle de Guatemala abarca la electrónica digital como la electrónica analógica, sin embargo, marcan la diferencia en la enseñanza y aplicación de cada una de estas dos ramas de la electrónica. Este proyecto no es solo la unión de estas dos disciplinas, abarca el diseño y fabricación de circuitos integrados en una disciplina más compleja que se integra en la industria. En el diseño de un circuito integrado a escala nano métrica es de vital importancia que un ingeniero electrónico maneje y domine las herramientas de software para la correcta creación en cada etapa de un circuito integrado. La empresa IMEC es colaboradora del diseño y fabricación del primer chip en Guatemala, con lo que nos proporcionó todos los recursos necesarios, tanto software como las restricciones y librerías de TSMC para poder llevar a cabo el diseño.

Este trabajo, que se ha realizado a lo largo de varios años, consta de dos etapas de flujo de diseño para circuitos integrados, estas son: Síntesis lógica y síntesis física. Para ambas etapas se cuenta con documentación de los pasos a seguir, así para una mejor comprensión en el uso de las herramientas de Synopsys.

Para la síntesis lógica se escaló de forma que se inició con un circuito de muy baja complejidad y se escaló en complejidad para luego poder sintetizar estos circuitos. Se comienza con una compuerta lógica NOT, luego una compuerta lógica XOR, luego un Sumador Completo, luego una unidad aritmética lógica y un contador secuencial de 4 bits.

Para la síntesis física se comenzó utilizando la herramienta de IC Compiler I y se hizo un gran avance al actualizar a IC Compiler II, con esta mejora de proceso se presentaron desafíos, ya que al actualizar no se tiene conocimiento de correcto manejo de esta herramienta con las actualizaciones que tiene, pero se contó con el apoyo de documentación y trabajos anteriores y el apoyo importante del IMEC para terminar este diseño. A esta etapa de síntesis física le sigue etapas de verificaciones que tienen que cumplir con los requerimientos de dichas pruebas para poder cumplir con las reglas de manufactura que solicita TSMC, con esto se le hace las verificaciones de LVS, LPE, ERC, Antenna y Rule Check. Al pasar cada una de estas comprobaciones, el circuito propuesto es apto para ser fabricado.

Debido a la complejidad, el trabajo en equipo entre los demás miembros de esta investigación es fundamental y juega un papel importante. No solo en la delegación de tareas y el logro de objetivos individuales, sino también para lograr el objetivo general, que en colaboración con la UVG es la creación del primer circuito a nano escala de la región.

<b>Prefacio</b>	VI
<b>Lista de figuras</b>	X
<b>Lista de cuadros</b>	XII
<b>Resumen</b>	XIV
<b>Abstract</b>	XVI
<b>1. Introducción</b>	<b>1</b>
<b>2. Antecedentes</b>	<b>3</b>
2.1 Instalación de aplicaciones de Synopsys . . . . .	5
2.2 Generación de archivo Verilog . . . . .	5
2.3 Síntesis lógica . . . . .	5
2.4 Síntesis Cell . . . . .	5
2.5 Síntesis Cell IO . . . . .	6
2.6 Síntesis física . . . . .	7
2.7 Comprobaciones físicas . . . . .	7
<b>3. Justificación</b>	<b>13</b>
<b>4. Objetivos</b>	<b>15</b>
4.1 Objetivo general . . . . .	15
4.2 Objetivos específicos . . . . .	15
<b>5. Alcance</b>	<b>17</b>
<b>6. Marco teórico</b>	<b>19</b>
6.1 Design Vision . . . . .	19
6.2 VCS . . . . .	19
6.3 Scripting . . . . .	20
6.4 StarRC . . . . .	22

6.4.1 Características de StarRC	22
6.4.2 Flujo de diseño de StarRC	23
6.5 Síntesis lógica	23
6.5.1 Flujo del diseño	23
<b>7. Script de Python para lectura universal</b>	<b>25</b>
7.1 Código explicado	26
<b>8. Planificación de la automatización</b>	<b>33</b>
8.1 Script de BASH	33
8.1.2 Resultados del script	33
8.2 Script para Design Compiler	36
8.2.1 Inclusión de librerías	36
8.2.2 Revisión de diseño	38
8.2.3 Compilación	39
8.2.4 Reportes	39
8.2.5 Outputs	39
<b>9. Modificaciones importantes de archivos</b>	<b>41</b>
9.1 Modificación del texto	41
9.2 Modificación del contador	45
9.3 Modificación de la tabla de verdad	46
<b>10. Integración de la automatización con la etapa de síntesis física y extracción de parásitos</b>	<b>49</b>
10.1 Montar servidor compartido	49
10.2 Script modificado para uso dentro del servidor	51
Integración de extracción de parásitos	52
<b>11. Automatización de la verificación de extracción de parásitos</b>	<b>55</b>
11.1 Script para la automatización de la verificación	55
<b>12. Conclusiones</b>	<b>61</b>
<b>13. Recomendaciones</b>	<b>63</b>
<b>14. Bibliografía</b>	<b>65</b>
<b>15. Anexos</b>	<b>67</b>
15.1 Códigos útiles y de resultados	67

---

## Lista de figuras

---

1. Comportamiento del nanoChip . . . . .	5
2. Esquemático del nanoChip . . . . .	6
3. Core del nanoChip . . . . .	6
4. Core del nanoChip . . . . .	7
5. DRC mostrando NOT CLEAN . . . . .	8
6. DRC mostrando errores en la densidad de los 6 metales . . . . .	9
7. Verificación de Antenna . . . . .	10
8. Verificación de extracción de parásitos . . . . .	11
9. Flujo de diseño de StarRC . . . . .	23
10. Diagrama del flujo de diseño . . . . .	24
11. Carpeta ubicada en el escritorio . . . . .	34
12. Archivos dentro de Sintesis Cell . . . . .	34
13. Archivos de salida dentro de la carpeta de salidas . . . . .	34
14. Archivos dentro de Sintesis Cell IO . . . . .	35
15. Archivos dentro de Union-cell-IO . . . . .	35
16. Archivos de salida con pines de entrada y salida dentro de la carpeta de salida . . . . .	36
17. Estructura de la lógica combinacional del texto anterior . . . . .	42
18. Estructura de la lógica combinacional del texto nuevo que es utiliza . . . . .	45
19. Carpetas dentro del servidor compartido . . . . .	50
20. Carpetas contenidas dentro de la sección designada . . . . .	50
21. Automatización de síntesis lógica . . . . .	51
22. Automatización de extracción de parásitos . . . . .	52
23. Dentro de la carpeta de parásitos dentro del servidor compartido . . . . .	52
24. Dentro de la carpeta de parásitos dentro de libs . . . . .	53
25. Dentro de la carpeta de parásitos dentro de logs . . . . .	53
26. Dentro de la carpeta de parásitos dentro de salidas . . . . .	53
27. Carpeta de extracción de parásitos en el escritorio . . . . .	55
28. Carpeta de extracción de parásitos en el escritorio dentro de las librerías . . . . .	57
29. Archivo de salida de la extracción de parásitos . . . . .	58

30. Archivo de reporte de la extracción de parásitos . . . . . 60

1. Código para realizar una simulación con VCS	19
2. Línea para creación de archivo con extensión <i>.vcd</i>	20
3. Importación de librerías útiles en Python	26
4. Lectura de un archivo	26
5. Límite de lectura	26
6. Listas utilizadas	27
7. Búsqueda dentro del rango establecido	27
8. Búsqueda del número de bits que conforma la señal	27
9. Búsqueda del número de bits que conforma la señal parte de salidas	28
10. Búsqueda especial para encontrar caracteres	28
11. Filtro para la palabra que se buscó anteriormente en el cuadro anterior	28
12. Filtro del nombre módulo y almacenaje en una lista diferente	29
13. Se crea el archivo final que se utilizará	29
14. Se escriben las variables dentro de la descripción del módulo nuevo	29
15. Escritura dentro del archivo todas las entradas, salidas y los wires	30
16. Línea para descripción de instancias	30
17. Filtro para signos de puntuación	30
18. Escritura de los pines de entrada y salida con librería de TSMC	31
19. Escritura de pines de alimentación	32
20. Código de inclusión de librerías utilizando la ruta de ubicación	36
21. Restricciones de celdas de la biblioteca	37
22. Restricción de un bloque de la celda	37
23. Lectura del archivo descrito en HDL	37
24. Comandos de revisión de diseño	38
25. Comando de compilación	39
26. Comandos de reportes	39
27. Escritura de archivos de salida	39
28. Textos originales	41
29. Segunda parte de los textos originales	42
30. Primer texto modificado	42

31. Segunda parte del primer texto modificado	43
32. Continuación del primer texto modificado	43
33. Segundo texto modificado	44
34. Continuación del segundo texto modificado	44
35. Contador original	45
36. Contador modificado	46
37. Tabla de verdad sin modificación	46
38. Tabla de verdad con la modificación para la compuerta XOR	46
39. Selector del texto original	47
40. Selector del texto modificado	47
41. Comando para montar un servidor compartido	49
42. Comandos extras para el servidor compartido	51
43. Script para ejecutar la verificación de extracción de parásitos	56
44. Comandos e inclusión de librerías para completar la verificación	59



El proyecto se basa en la automatización de la etapa de síntesis lógica para llevar a cabo una etapa automática, rápida y efectiva del primer chip, diseñado y elaborado en Guatemala por estudiantes de la Universidad del Valle de Guatemala. Esta es la primera etapa para la realización del chip, es muy importante, ya que al ser la primera etapa los resultados deben de salir correctos para que no interrumpa el proceso de diseño. El objetivo por cumplir es verificar que los resultados obtenidos sean los adecuados después de realizar la automatización de dicho proceso y lograr obtener los archivos de Verilog que se utilizaran para realizar las pruebas en un FPGA *Genesys Xilinx Virtex-5 LX50T* para corroborar que el proceso de diseño del chip es exitoso.

La síntesis lógica es un código que describe un circuito en código Verilog para otorgar compuertas lógicas de hardware para una tecnología dada, la que se utilizaran en este diseño del primer chip será con librerías de TSMC. Este proceso de la etapa de síntesis lógica consta de dos pasos:

- Traducción implica comprender la descripción VHDL de las compuertas.
- Optimización que implica seleccionar la combinación más eficiente de las librerías de las bibliotecas de esa tecnología que se utiliza y las celdas para lograr relacionar las funciones y que este funcione de la manera más adecuada y que cumpla con su funcionalidad descrita.

En esta etapa se pueden agregar aspectos importantes para que el diseño de este funcione adecuadamente, como lo son las restricciones de diseño que se ven delimitadas por la tecnología que se utiliza, colocación de componentes de manera eficiente para optimizar de mejor manera el rendimiento del chip que se está desarrollando. Como resultados del proceso de esta etapa se pueden obtener 3 formatos de archivos, que son: *'.sdc'*, este agrupa las netlist que el usuario configura durante el proceso de síntesis, el *'.ddc'* almacena toda la información de los netlist que se generaron al igual que las restricciones que se le asignan al circuito y el último *'.v'* es el archivo que contiene ya todas las netlist con las configuraciones

y restricciones que se hicieron previamente donde también se interconectan y se utiliza en las etapas siguientes para el seguimiento del desarrollo del chip.

Design Vision es una herramienta de software utilizada para llevar a cabo con éxito la etapa de síntesis lógica de un circuito que se describe a través de archivos HDL, este pasa por un proceso de síntesis donde se logra sintetizar para poder obtener archivos netlist a estructura de compuerta. Este puede utilizar librerías comunes y personalizadas como las que proporciona la empresa de TSMC.

VCS es una herramienta donde se realizan las verificaciones del diseño en el que se está trabajando, esta misma cuenta con un simulador y sistema para resolver limitaciones que pueda presentar el diseño. DVE, por otra parte, es un complemento de VCS, este permite simular y ver el comportamiento del diseño que se trabajó, con esto lograr ver que los resultados a la salida son los adecuados, gracias a la interfaz gráfica amigable es fácil de comprender y utilizar.

StarRC es una herramienta fundamental en la industria de EDA para la extracción de parásitos. Esta herramienta proporciona soluciones de extracción de silicio precisa y de alto rendimiento para SoC, digital personalizado y analógico / señal mixta. Esta implementación ofrece una facilidad de uso y productividad para poder acelerar el cierre de diseño y verificación de firma.

---

## Abstract

---

The project is based on the automation of the logic synthesis stage to carry out a fast and effective automated stage of the first chip, designed and produced in Guatemala by students from the Universidad del Valle de Guatemala. This is the first stage for the realization of the chip, it is very important since being the first stage the results must be correct so that it does not interrupt the design process. The objective to be met is to verify that the results obtained are adequate after automating said process and obtaining the files of Verilog that are used to perform tests on a *Genesys Xilinx Virtex-5 LX50T* FPGA to verify that the chip design process is successful.

Logic synthesis is a code that describes a circuit in Verilog code to provide hardware logic gates for a given technology, which will be used in this first chip design with releases from TSMC. This process of the logical synthesis stage consists of two steps:

- Translation Involves understanding the VHDL description of the gates.
- Optimization that implies selecting the most efficient combination of the libraries of that technology that is used and the cells to correlate the functions and that it works in the most appropriate way and that it complies with its described functionality..

At this stage, important aspects can be added so that the design works properly, such as the design restrictions that are delimited by the technology used, efficient component placement to better optimize the performance of the chip that is used. it is developing. As a result of the process of this stage, 3 file formats can be obtained, which are: *'sdc'*, this groups the netlist that the user configures during the synthesis process, the *'ddc'* stores all the information of the netlists that were generated as well as the restrictions that are assigned to the circuit and the last *'v'* is the file that already contains all the netlists with the

configurations and restrictions that were previously made where they are also interconnected and used in the following stages to follow the development of the chip.

Design Vision is a software tool used to successfully carry out the logical synthesis stage of a circuit that is described through HDL files, this goes through a synthesis process where it is possible to synthesize in order to obtain netlist files to structure of gate. It can use common and custom libraries such as those provided by the TSMC company.

VCS is a tool where the verifications of the design that is being worked on are carried out, it has a simulator and a system to resolve limitations that the design may present. DVE, on the other hand, is a VCS complement, it allows you to simulate and see the behavior of the design that was worked on, with this you can see that the output results are adequate, thanks to the friendly graphical interface it is easy to understand and use.

StarRC is a fundamental tool in the EDA industry for parasite extraction. This tool provides high-performance, precise silicon removal solutions for SoCs, custom digital, and analog/mixed-signal. This implementation offers ease of use and productivity in order to speed up design close and signature verification.

# CAPÍTULO 1

---

## Introducción

---

Con la etapa de diseño de síntesis lógica y un flujo de diseño adecuado se hace la creación de un circuito en lenguaje HDL con el que es necesario contar antes de poder iniciar con la síntesis lógica, el cual toma uso de librerías de TSMC. Es necesario que este circuito final no presente ningún tipo de error para lograr avanzar a la siguiente etapa del flujo de diseño. Para el flujo se utilizó la herramienta de Design Vision, donde se colocan los comandos necesarios uno a uno, con lo que se demora en completar la síntesis lógica. Se emplea la terminal para facilitar de forma eficiente y correctamente en el menor tiempo posible, asegurando una síntesis lógica completa y adecuada y así entregar en un servidor compartido los archivos necesarios para uso en la siguiente etapa de flujo de diseño.

El presente trabajo esta organizado y dividido en 5 capítulos:

El capítulo 7 contiene una explicación del proceso que se quiere para poder analizar y realizar una lectura universal de un código descrito en HDL, para el cual se hizo uso de Python para realizar un script con el cual se hace una lectura y se reescribe en un archivo nuevo el módulo, las entradas y salidas del circuito que describe el funcionamiento. En este mismo capítulo se hace una explicación detallada de todo el código utilizado.

El capítulo 8 habla específicamente de las planificación de la automatización dividida en dos puntos importantes, el primero un script en bash para navegar dentro de directorios y tener una buena organización de carpetas ya que se eliminan los archivos y carpetas para evitar duplicidades; en el segundo script se tiene un archivo con todos los comandos necesarios que usa *design vision* para poder completar el proceso de síntesis lógica.

El capítulo 9 contiene todas las modificaciones importantes que se realizaron, estas me-

jas se hicieron al texto ya que se agrego mas al mismo y se hizo una cambio ya que el segundo texto es el primer texto solo que en idioma inglés. Al hacer este cambio se tuvo que hacer más grande el contador para que pueda pasar por todos los caracteres que se agregaron. El último cambio que se realizó fue la modificación de la tabla de verdad en la forma de seleccionar el texto, esto con el objetivo de poder generar una compuerta XOR para poder tener dos selectores.

El capítulo 10 se presenta toda la integración explicada a detalle de como se hizo un servidor compartido para entregar de una forma mas eficiente los archivos que se son necesarios para la siguiente etapa que es síntesis física.

El capítulo 11 se documenta todo el proceso para la automatización de la verificación de extracción de parásitos, en dicho capítulo se tiene la explicación de los archivos a utilizar así como los archivos que se generan y por último el orden de navegar entre directorios para tener una organización de carpetas, eliminar archivos de pruebas anteriores con el propósito de no tener archivos duplicados.

El primer paso que se dio fue en 2014 cuando se logró tener un acuerdo con la empresa de Synopsys y de esta forma poder tener acceso al software para poder desarrollar tecnologías nuevas para Guatemala. Esta adquisición fue imperativa para que estudiantes se adentraran en esta rama de la electrónica que es la nanoelectrónica y que en ese mismo año se presente el primer estudiante en presentar un trabajo sobre VLSI por parte de UVG.

Synopsys Inc. es una empresa líder en desarrollo de software que se utiliza en el diseño de semiconductores, especializado en el campo de automatización de diseño electrónico. Estos productos ayudan a desarrollar y probar diseños de circuitos integrados antes de ser fabricados. Esto permite alcanzar un estándar óptimo en costos, tamaño y potencia. Ya con estas pruebas realizadas, Synopsys vende los productos finales a los fabricantes de semiconductores, informática, comunicaciones, electrónica comercial y agencias aeroespaciales.

En el año de 1970, el software de automatización del diseño electrónico se convirtió en un factor clave en los avances dramáticos de la industria electrónica. Cada vez más los circuitos integrados se hacían más complejos, junto a la escasez de ingenieros de circuitos integrados, crearon la necesidad de tener un software que pudiera reducir el tiempo de comercializar y costos de diseño, como también la confiabilidad de diseños de alta velocidad y de alta densidad.

Por la década de 1970 trajo la primera generación de EDA (Electronic Design Automation) y son:

1. Diseño asistido por computadora (CAD)
2. Ingeniería asistida por computadora (CAE)

### 3. Automatización en el diseño electrónico (EDA)

La CAE logró un proceso en la automatización del diseño de los circuitos integrados más complejos, pero los ingenieros seguían invirtiendo mucho tiempo innecesario en conectar todos los nodos que se utilizan en los chips de silicio, con lo que la CAE no pudo seguir los rápidos cambios de la industria electrónica y la complejidad de los circuitos lo que llevo a la invención de un método mejor y es donde surgió la tecnología de EDA junto con Synopsys.

Aart de Geus, desarrolló una idea para una nueva tecnología de software llamada síntesis. Con esta síntesis los ingenieros podrían describir la funcionalidad de un circuito en lenguaje informático, en lugar de describirlos en términos de puertas individuales. Esta herramienta crearía automáticamente la síntesis lógica, esto les ahorra tiempo en diseño y libera a los ingenieros para soluciones de diseño para mayor eficiencia. La síntesis genera diseños de circuitos a partir de lenguajes que describen hardware como VHDL y Verilog, lo que permite esto es la nueva tecnología de EDA en la automatización del diseño del lenguaje de hardware.

1

Desde el 2019 se tuvo la visión de poder estudiar, diseñar y mandar a fabricar el primer procesador con una tecnología de 180 nm a la empresa de TSMC que es la empresa a nivel mundial que fabrica los procesadores, con lo que la universidad al igual que el trabajo arduo de estudiantes hicieron el proceso de la gestión para poder conseguir apoyo de parte de la empresa y así realizar el desarrollo del proyecto.

Los primeros avances realizados en el campo de la nanoelectrónica que se llevaron a cabo en la Universidad del Valle de Guatemala se dieron en el año 2013, donde se introdujo VLSI y en el 2014 se hizo un acuerdo con la empresa de Synpsys, donde se lograron obtener credenciales para obtener las herramientas necesarias para poder realizar el diseño y simulación de un nano chip. El primer trabajo de graduación realizado fue de parte del ingeniero en electrónica Jonathan de los Santos 2 donde se relaciona con las herramientas de VLSI y fue la base para el continuo desarrollo para los trabajos que se siguen realizando el día de hoy.

Seguido en el año 2020, los ingenieros electrónicos, Luis Abadia 3, Marvin Flores 4 y Matthias Sibrian 5 realizaron simulaciones del nano chip que se sigue diseñando actualmente con librerías de TSMC en los programas de IC Validator y IC Compiler I.

En el año 2021 el ingeniero electrónico Elmer Torres 6, realizo y verifiko la fase de diseño más importante para un chip que es la etapa de síntesis lógica en el flujo de diseño, en ese mismo año Antonio Altuna 7, Jose Ayala 8 y Julio Shin 9, desarrollaron la siguiente etapa del desarrollo que es la síntesis física del nano chip, seguido de las verificaciones de Antena y corrección de errores que obtuvieron durante el desarrollo y diseño que ellos realizaron.

Durante los trabajos que se realizaron hubo una actualización importante en el cambio de versión de software donde se mudó de IC Compiler I a IC Compiler II, este cambio para mayor beneficio en la utilización de la herramienta produjo atrasos en el seguimiento del diseño del nano chip trayendo consigo 6 errores que no se han logrado solucionar que son errores de densidad en 6 metales.



En el año 2022 se replicó todos los avances de los años anteriores para tener el conocimiento adecuado para poder continuar el diseño y corregir errores que se pueden presentar en el camino, siendo los más claros lo 6 errores de densidad.

Las réplicas que se realizaron fueron enumeradas por secciones más adelante para tener un orden un poco más separado, ordenado y estructurado para una mayor facilidad de lectura.

## 2.1 Instalación de aplicaciones de Synopsys

Es de vital importancia tener instalado las herramientas de software que la empresa de Synopsys nos brindó. Estos pasos están detallados en el trabajo de graduación del ingeniero electrónico Jonathan de los Santos [\[2\]](#)

## 2.2 Generación de archivo Verilog

En trabajos previos se hizo la creación de un código en Python para poder generar un archivo de Verilog, en el cual describe el nano chip que se está realizando. Dentro de este código contiene dos textos que se desean mostrar en las salidas del mismo nano chip, este hace la conversión de texto ASCII a binario para obtener una salida de 1 byte. Este tiene un contador que avanza y recorre el texto para realizar la conversión de todo el texto.

Este código de último tiene una salida de archivo de Verilog que describe en este lenguaje el nano chip y es utilizado en la etapa de diseño de síntesis lógica.

## 2.3 Síntesis lógica

Esta etapa de diseño se dividió en dos partes importantes, Síntesis Cell y Síntesis Cell\_ - IO. En la primera se define la síntesis del nano chip, pero este no contiene las entradas y salidas, con lo que se pasa a la segunda parte de la síntesis donde se generan los archivos con los pines de entradas y salidas ya incluidos en el nano chip. [\[6\]](#)

## 2.4 Síntesis Cell

En la Figura [\[1\]](#) se puede ver el comportamiento del nano chip utilizando la herramienta de software de DVE.

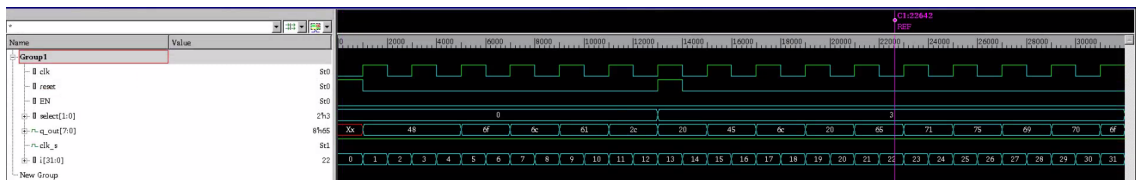


Figura 1: Comportamiento del nanoChip

## 2.5 Síntesis Cell\_IO

En la Figura 2 se observa cómo después de pasar la parte de diseño de Sintesis\_Cell\_IO utilizando la herramienta de software de Design Vision se genera el módulo del nano chip ya sintetizado con sus pines adecuados de entradas y salidas.



Figura 2: Esquemático del nanoChip

En la Figura 3 se puede observar el esquemático del nano chip con sus pines de entrada y salida ya sintetizados. Para observar esto se selecciona el módulo del chip de la Figura 2 dos veces y se entra dentro de este gran Black box que dentro contiene el core del nano chip ya sintetizado conjunto con sus pines de entrada y salida. Se selecciona de nuevo dos veces el core del nano chip y nos mostrará dentro el esquemático que contiene el nano chip ya sintetizado.

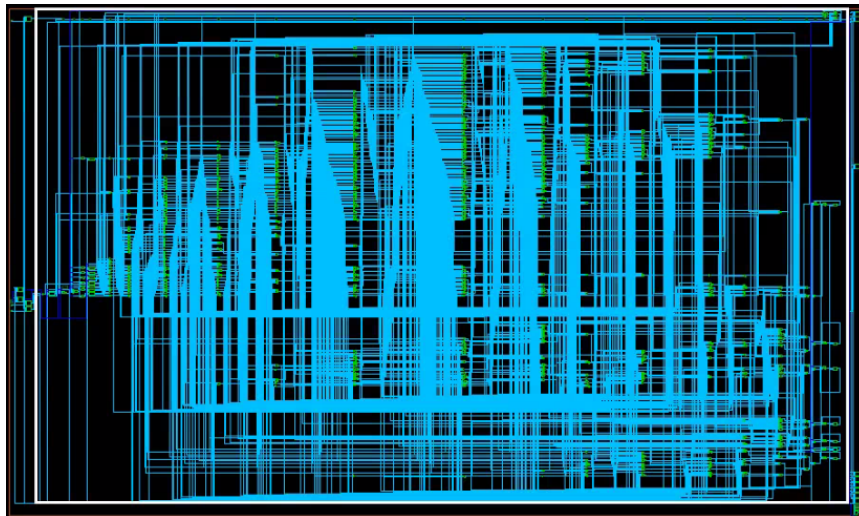


Figura 3: Core del nanoChip

## 2.6 Síntesis física

En esta etapa de diseño se utilizan archivos que se generan en la etapa de diseño de síntesis lógica, esta etapa es para poder representar la etapa anterior de forma física y visualizar las interconexiones que tiene por dentro. [7] [8] [9]

En la Figura 4 se puede ver después de realizar la correcta síntesis física el core del chip ya físico listo para poder pasar a las verificaciones Físicas y que puedan pasar las pruebas, ya que son necesarias para dar el visto bueno del diseño del nano chip.

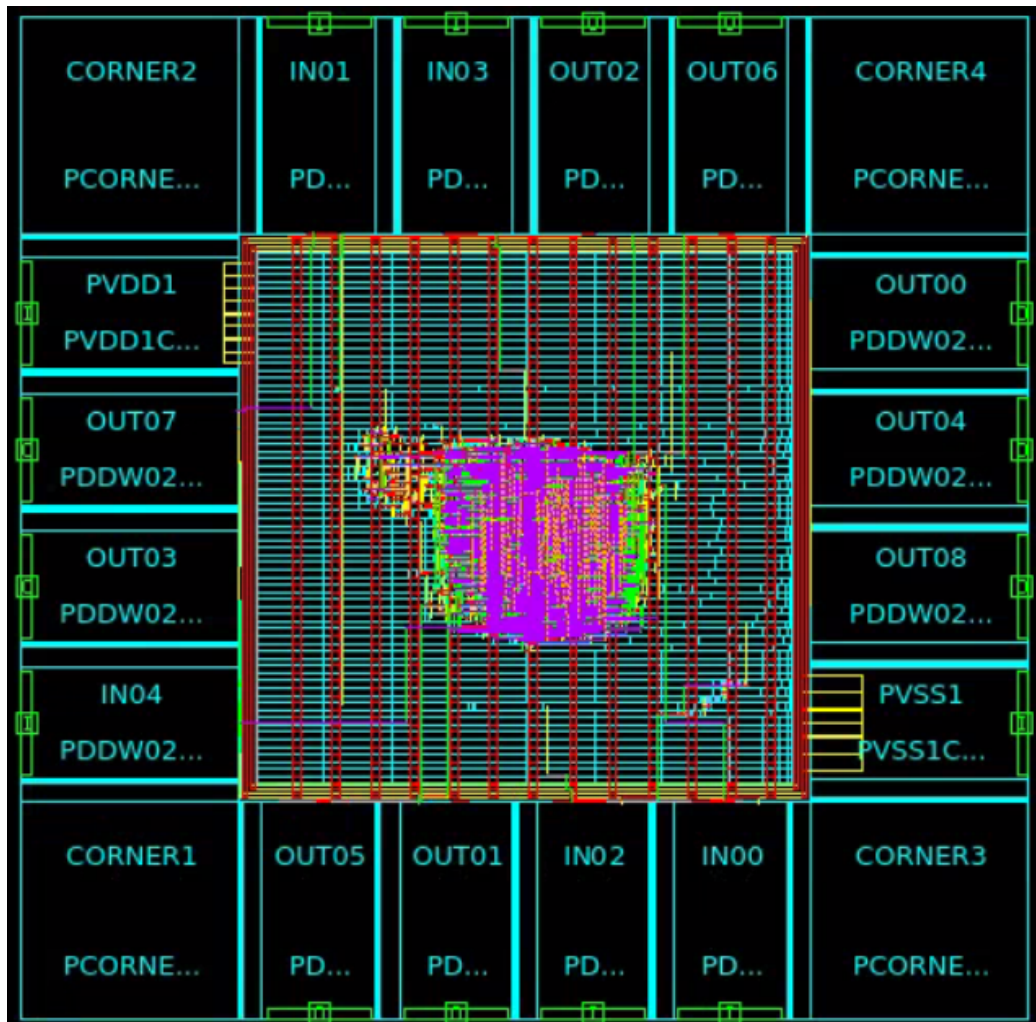


Figura 4: Core del nanoChip

## 2.7 Comprobaciones físicas

- DRC (Design Rule Check)

En esta verificación se ve que cumpla todos los requerimientos de fabricación, tal tecnología de fabricación la delimita la tecnología nano métrica que se utilice, como las distancias que debe de tener entre metales, difusiones y polisilicios. En estas verificaciones se encontraron los 6 errores en los metales que no han tenido una solución.

Como se pueden observar en el Figura 5 se puede ver que la verificación de DRC no muestra un mensaje de PASS, esto debido a que no en el diseño del nano chip hay errores y en el Figura 6 se pueden observar claramente los 6 errores de densidad en estos metales.

```

RESULTS: NOT CLEAN

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

=====
-----
ICV Execution
-----

                          IC Validator

Version S-2021.06-SP3-2 for linux64 - Jan 06, 2022 cl#7200131

Copyright (c) 1996 - 2022 Synopsys, Inc.
This software and the associated documentation are proprietary to Synopsys,
Inc. This software may only be used in accordance with the terms and conditions
of a written license agreement with Synopsys, Inc. All other use, reproduction,
or distribution of this software is strictly prohibited. Licensed Products
communicate with Synopsys servers for the purpose of providing software
updates, detecting software piracy and verifying that customers are using
Licensed Products in conformity with the applicable License Key for such
Licensed Products. Synopsys will use information gathered in connection with
this process to deliver software updates and pursue software pirates and
infringers.

Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on
Inclusivity and Diversity" (Refer to article 000036315 at
https://solvnetplus.synopsys.com)

Called as: icv -icc2 -f NDM -i EL_GRAN_JAGUAR.ndm -p /home/nanoelectronica/Documents/D
icc2_error_categories -I /home/nanoelectronica/Documents/Diseno/Entrega/DRC -icc2_erro
signoff_check_drc.rc -I /home/nanoelectronica/Documents/Diseno/Entrega/DRC /home/nanoe

-----

Input file name:          EL_GRAN_JAGUAR.ndm
Top cell name:           chip_IO
Layout format:           NDM
Runset file:             /home/nanoelectronica/Documents/Diseno/Entrega/ICVLM18_LM16_LM
Working directory:       /home/nanoelectronica/Documents/Diseno/Entrega/DRC
Run details directory:   /home/nanoelectronica/Documents/Diseno/Entrega/DRC/run_details
Layout error file:       chip_IO.LAYOUT_ERRORS
User name:               nanoelectronica
Time started:            2022/03/10 09:16:32PM
Time ended:              2022/03/10 09:20:59PM

```

Figura 5: DRC mostrando NOT CLEAN

-----  
**Results Summary**  
-----

**Rule and DRC Error Summary**

192 total rules were run.  
226 rules NOT EXECUTED.  
6 rules have violations.  
There are 6 total violations.  
Refer to chip\_I0.LAYOUT\_ERRORS

-----

Rule	Violations Found
M1.R.1	v = 1
M2.R.1	v = 1
M3.R.1	v = 1
M4.R.1	v = 1
M5.R.1	v = 1
M6.R.1	v = 1

Figura 6: DRC mostrando errores en la densidad de los 6 metales

- **LVS (LayOut vs Schematic)**

En esta verificación se compara que los netlist obtenidos en la síntesis física y la síntesis lógica sean comparados para que no existan errores, redundancias o duplicidades para no tener errores en el diseño.

- **Antenna Rule Check**

En esta verificación se busca efectos de antena en el diseño del nano chip, nos referimos a este efecto cuando se acumula carga en un conductor, lo que se conoce como una carga parásita, lo que puede llegar a dañar los CMOS.

En la verificación de Antenna Rule Check se puede ver que es un visto bueno y que no hay errores en esta verificación, esto se observa en la Figura [7](#)

RESULTS: CLEAN

```
#### # ##### ## # #  
# # # # # ## #  
# # ##### ##### # # #  
# # # # # # ##  
#### ##### ##### # # # #
```

-----  
ICV Execution  
-----

IC Validator

Version S-2021.06-SP3-2 for linux64 - Jan 06, 2022 cl#7200131

Copyright (c) 1996 - 2022 Synopsys, Inc.

This software and the associated documentation are proprietary to Synopsys, Inc. This software may only be used in accordance with the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, or distribution of this software is strictly prohibited. Licensed Products communicate with Synopsys servers for the purpose of providing software updates, detecting software piracy and verifying that customers are using Licensed Products in conformity with the applicable License Key for such Licensed Products. Synopsys will use information gathered in connection with this process to deliver software updates and pursue software pirates and infringers.

Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on Inclusivity and Diversity" (Refer to article 000036315 at <https://solvnetplus.synopsys.com>)

Called as: icv -i EL\_GRAN\_JAGUAR.gds -c chip\_I0 -sf ICV -vue ICVLM18\_LM16\_LM152\_6M.215a\_pre041518

-----  
Input file name: EL\_GRAN\_JAGUAR.gds  
Top cell name: chip\_I0  
Layout format: GDSII  
Runset file: ICVLM18\_LM16\_LM152\_6M.215a\_pre041518  
Working directory: /home/nanoelectronica/Documents/Diseno/Entrega/antenna  
Run details directory: /home/nanoelectronica/Documents/Diseno/Entrega/antenna/run\_details  
Layout error file: chip\_I0.LAYOUT\_ERRORS  
User name: nanoelectronica  
Time started: 2022/03/10 09:36:05PM  
Time ended: 2022/03/10 09:40:36PM

Figura 7: Verificación de Antenna

- Extracción de parásitos** Esta verificación agarra los parámetros eléctricos para que las características del circuito estén bien. Si en caso existiese una capacitancia o resistencia parásita, se genera un archivo SPICE donde indica cuál o cuáles son los problemas.

En la verificación de Extracción de Parásitos se puede ver que hay muchos WARNING que se deben a los errores de densidad en los metales, pero a pesar de ello aún es posible poder mandar a fabricar el nano chip. Esto se observa en la Figura 8

```

Open [icon] stdout.lpe.log
~/Documents/Diseño/Entres

StarRC (TM)

Version S-2021.06-SP4 for linux64 - Nov 18, 2021

Copyright (c) 1999 - 2021 Synopsys, Inc.
This software and the associated documentation are proprietary to Synopsys,
Inc. This software may only be used in accordance with the terms and conditions
of a written license agreement with Synopsys, Inc. All other use, reproduction,
or distribution of this software is strictly prohibited. Licensed Products
communicate with Synopsys servers for the purpose of providing software
updates, detecting software piracy and verifying that customers are using
Licensed Products in conformity with the applicable License Key for such
Licensed Products. Synopsys will use information gathered in connection with
this process to deliver software updates and pursue software pirates and
infringers.

Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on
Inclusivity and Diversity" (Refer to article 000036315 at
https://solvnetplus.synopsys.com)

ExecName: /usr/synopsys/starrc/S-2021.06-SP4/linux64_starrc/bin/StarXtract
Version: S-2021.06-SP4
Built on: Nov 18 2021 18:35:04
Start Time: Thu Mar 10 21:46:54 2022

Host ..... uvgiemtbcmit11810

xTractorName: /usr/synopsys/starrc/S-2021.06-SP4/linux64_starrc/bin/xTractor
Version: S-2021.06-SP4
Built on: Nov 18 2021 18:34:54
INFORMATION: TCAD_GRD_FILE (nxtgrd file) generated by >4 years older version than the StarRC
version is used for this extraction. (SX-3595)
WARNING: LVS Black Box cell 'INR4D0BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'INR4D0BWP7T' is added to SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AN4D1BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AN4D1BWP7T' is added to SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AOI31D0BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AOI31D0BWP7T' is added to SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AOI222D0BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AOI222D0BWP7T' is added to SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AOI211D1BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AOI211D1BWP7T' is added to SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'NR4D0BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'NR4D0BWP7T' is added to SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'OA211D0BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'OA211D0BWP7T' is added to SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AOI22D0BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'AOI22D0BWP7T' is added to SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'IOA21D1BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'IOA21D1BWP7T' is added to SKIP_CELLS list automatically. (SX-3070)
WARNING: LVS Black Box cell 'CKAN2D1BWP7T' is added to SCHEMATIC_SKIP_CELLS list automatically. (SX-3070)

```

Figura 8: Verificación de extracción de parásitos





---

### Justificación

---

Hay una gran diferencia entre electrónica y nanoelectrónica, los conceptos que se pueden aprender en la rama de la electrónica son el estudio de aplicación del comportamiento del movimiento de los electrones en diversos escenarios; sin embargo, en la nanoelectrónica se utilizan componentes electrónicos como los transistores que tienen un tamaño de 100 nm o menos de tamaño y al tener un tamaño tan reducido se necesita un estudio más minucioso debido a las interacciones y las propiedades de los materiales a una escala muy pequeña, ya que hay muchos fenómenos que pueden afectar de manera drástica a los transistores. Este trabajo se realiza con el fin de poder adentrarnos en la rama de la nanoelectrónica y poder comprender a nivel físico los sucesos que pasan cuando se consideran o no los fenómenos físicos y como esto ayuda o empeora el rendimiento de un chip.

La tarea de diseñar, estudiar y fabricar un chip no es sencilla, el reto que tomaron varios estudiantes al incursionar en este gran proyecto da merito a la Universidad del Valle de Guatemala al innovar e investigar en una rama que no es muy común en Guatemala y al lograrlo será una dicha, ya que se tiene un proceso para el diseño, se motiva a jóvenes estudiantes a conocer más sobre la nanoelectrónica y se obtiene para Guatemala un título en poder ser los primeros de muchos otros países el poder realizar una tarea tan importante como es esta ya mencionada.

Este proceso no es sencillo, ya que se utilizan herramientas de software avanzadas y poco exploradas por estudiantes de esta misma casa de estudios, pero al tener manuales de guía de uso y conocimientos sobre scripting en una distribución de Linux se puede automatizar todo el proceso de diseño, simulación y corrección del procesador para tener una forma más eficiente a la hora de poder obtener un resultado final de un chip dedicado a un propósito en general y no cometer errores que puedan afectar en la manufacturación del mismo.



### 4.1 Objetivo general

Automatizar la etapa de síntesis lógica de forma en que no se altere el proceso al no utilizar la interfaz gráfica de Desing Vision, tener un orden estricto y adecuado para obtener archivos Verilog necesarios para pruebas en un FPGA Genesys Xilinx Virtex-5 LX50T

### 4.2 Objetivos específicos

- Realizar un script para automatizar la fase de diseño de síntesis lógica, eliminar archivos en las carpetas donde se encuentre para evitar duplicidades de archivos y tener un orden estricto en carpetas de entradas y salidas de la etapa de síntesis lógica.
- Obtener un archivo de salida Verilog donde se describa el chip con entradas y salidas para darlo a un grupo de trabajo y mejorar el trabajo en equipo y la comunicación para resolver problemas que se puedan presentar.
- Mejorar comunicación con equipo de síntesis y verificaciones físicas para acoplar partes de automatización separadas para tener un correcto funcionamiento con un solo script.
- Verificar y corregir posibles errores que surjan en el acoplamiento de scripts de automatización para garantizar un correcto flujo de diseño y simulación.
- Realizar un script para automatizar la fase de extracción de parásitos y explorar más el manual para poder realizar posibles mejoras a esta fase antes mencionada.



El alcance de este trabajo de graduación se centra en tres aspectos importantes. El primero poder automatizar cualquier circuito descrito en lenguaje Verilog y poder leer y pasar correctamente la síntesis lógica sin ninguna falla, con esto pasar la síntesis cell donde no se necesitan los pines de entrada ni salida y por último crear de forma autónoma el módulo necesario que sé instancia para poder declarar los pines de entrada y salida, teniendo en cuenta el orden y la sintaxis adecuada que se utiliza para llamar y declarar adecuadamente la librería de TSMC.

El segundo es poder integrar los archivos de salida de la etapa de síntesis lógica y poder entregar dentro de una carpeta compartida con los demás compañeros donde se mejoró la comunicación y se creó una carpeta compartida en un servidor y así no tener que esperar a mandar archivos y que puedan corromperse, de igual manera esta carpeta sirve para las demás verificaciones que se realizan y así tener un orden para el proceso completo del diseño del nano chip.

El tercero es poder automatizar y mejorar a través de comandos de los manuales y documentación de Synopsys la verificación de LPE que se conoce mayormente como extracción de parásitos. Con estas mejoras se busca poder obtener los pines de entrada y salida que no aparecen dentro del archivo, con lo que es importante tenerlo para completarlo con éxito.



## 6.1 Design Vision

Los scripts se utilizan cuando es rutina ejecutar una serie o listado de comandos de forma repetitiva, como establecer restricciones o definir otros atributos al diseño. Se pueden aprovechar scripts TCL's que ya existan en Design Compiler en la línea de comando, así como en la GUI. Se puede crear un único archivo que contenga todas estas instancias, restricciones y reglas de diseño en un archivo de texto. Cualquier dcnext\_shell o dc\_shell se puede ejecutar dentro de un archivo con extensión script.

## 6.2 VCS

Para poder realizar una simulación se debe de contar con un archivo de test bench en formato de Verilog, en este se deben de describir el estado de las entradas en diferentes tiempos. Este archivo debe de tener las siguientes líneas de código [10](#):

```
1 $fsdbDumpfile ("file_name.fsdb");  
2 $fsdbDumpvars (0, file_name_tb);  
3 $dumpfile ("file_name. vcd");  
4 $dumpvars (0, file_name_tb);
```

Cuadro 1: Código para realizar una simulación con VCS

Estas líneas que se ven en el Cuadro [1](#) nos permiten generar un archivo .vcd que contiene

el waveform de la simulación y un archivo .fsdb que es importante para las siguientes etapas de diseño del nano chip.

Para realizar la simulación se debe de utilizar la herramienta de software VCS que nos genere un ejecutable de la simulación del archivo, luego se ejecuta la simulación para obtener un waveform del diseño con lo que se ejecuta el comando [10]

---

```
1 vcs -v tpd018nv.v tcb018gbwp7t.v file_name.v file_name_tb.v -o simulacion
  ↪ -full64 -debug_access+all
```

---

Cuadro 2: Línea para creación de archivo con extensión .vcd

- -v nos establece el formato del archivo que se le carga al software
- -o especifica el nombre del ejecutable
- -full64 compila el diseño en modo de 64 bits y crea un ejecutable de 64 bits
- -debug\_accesall activa el dumping de los archivos FSDB/VPD y en conjunto habilita las capacidades de depuración.

Luego se ejecuta la simulación con `./simulacion` luego de ejecutarlo realiza la simulación del diseño con la línea del Cuadro [2] y creará los archivos vcd y fsdb.

## 6.3 Scripting

Los scriptings antiguos se utilizaban más bien para aplicaciones muy concretas. Con la aparición de la World Wide Web, se establecieron una serie de lenguajes de scripting para la utilización de servidores web. Puesto que los lenguajes de scripting simplifican el procesamiento de texto, lo que hace y facilita la creación DINAMICA de páginas en la Web.

En comparación con otros lenguajes de programación utilizados hoy el lenguaje de scripting no con lleva compilación, es decir que no se genera ningún archivo binario a partir del texto fuente éxito por el programador. Como consecuencia, reducen la carga para el programador, pero aumentan la carga al procesador.

La idea de facilitar el trabajo del programador es el hilo conductor de muchos lenguajes de scripting. De esta forma podemos evitar la gestión manual, un método mas eficiente, pero mas recurrente a tener errores en la ejecución. Además, no se indica el tipo de variable y puesto que no se compila no necesita un main. Con un scriptin se puede programar software de manera mas directa y con menos texto fuente.



Los lenguajes de scripting se categorizan en función del uso y el lugar de utilización. Algunos se utilizan en la línea de comandos para agrupar sucesión de órdenes y conseguir así la automatización. A este tipo de lenguajes pertenecen Bash y PowerShell.

Para este script se hizo uso de Bash con lo que se consulto el manual del profesor Francisco Ruiz Sala [11] para poder encontrar los comandos en lenguaje scripting adecuados para realizar una tarea en específico, completar la síntesis física sin necesidad de utilizar la GUI.

- De primero se tiene que colocar el nombre del archivo con extensión **“.bash”** para que este puede ser ejecutado desde una terminal.
- En la primera línea se debe de colocar la siguiente línea de comando para que el archivo cuando se guarde se quede como un ejecutable, de lo contrario por mas que tenga la extensión adecuada este no funcionara. **#!/bin/bash**
- El comando **“echo”** es para poder imprimir en la terminal un mensaje. Este es recomendable colocarlo en partes esenciales de un script ya que es posible ver de una manera mas ordenada por donde va la secuencia de los comandos que se están ejecutando, por ejemplo: **echo “Example: Hello World!”**.
- El comando **“cd”** se utiliza para poder ingresar a un directorio, luego del comando se debe de colocar **./file\_name**, para especificar el nombre del directorio, este comando tiene una variante que es **cd ..** este es para regresar un directorio atrás. Este comando es utilizado para navegar entre directorios. **cd ./Documents**.
- El comando **“mv”** es para mover archivos hacia un directorio especificado, la forma en que se debe de colocar luego del comando es el nombre del archivo que se desea mover de directorios seguido del directorio a donde se desee mover. **mv archive ./Documents**.
- El comando **“cp”** es utilizado para copiar archivos, al igual que el comando **“mv”** se debe de colocar seguido del comando el nombre del archivo que se desea copiar seguido del directorio al que se copiara el archivo. **archive1 ./Documents**.
- El comando **“cat”** es utilizado para unir dos archivos con la misma extensión. Se debe de colocar de la siguiente manera: **archive1.extension archive2.extension > archive3.extension**, se especifica el nombre de los dos primeros archivos luego se coloca el signo mayor y seguido el nombre que tendrá la salida.
- El comando **“chmod”** es utilizado para poder cambiar los permisos de un archivo, se deben de conocer previamente que **“rwx”** son abreviaciones de read, write, execute, estos son los permisos que se quieren cambiar con este comando. Luego de escribir el comando se especifica **“+rwx”** y se dejan que permisos son los que se cambiaran, seguido del nombre del archivo.

## 6.4 StarRC

StarRC es una herramienta de la empresa de Synopsys que extrae parásitos, tales como “Resistencias”, “Capacitores” e “Inductores” de la base de datos que corresponde al diseño de circuitos integrados. Esta herramienta utiliza StarRC para poder obtener las netlists y generará análisis de sincronización, ruido y electro-migración.

### 6.4.1 Características de StarRC

StarRC es parte de las herramientas que dan aprobación, ya que este proporciona un análisis completo, eficiente y preciso de la extracción de espectro para un chip que contiene millones de transistores o examinar una red pequeña importante del mismo circuito.

Algunas de las características de la herramienta StarRC son las siguientes:

- Extracción de parásitos de chip completo para su uso con verificación de tiempo, electro-migración y ruido.
- Solucionador de campo integrado para un análisis tridimensional preciso de las redes seleccionadas.
- Extracción tanto a nivel de compuertas como de transistor.
- Extracción plana y jerárquica.
- Interoperabilidad y layout vs schematic de terceros, personalizado de herramientas de diseño, análisis de tiempo y simulación.
- Capacidad para analizar diseños en etapa inicial que tienen aberturas y otras reglas de diseño y violaciones.

Las características de StarRC Ultra ofrecen capacidades avanzadas, que incluyen:

- Técnicas eficientes de procesamiento distribuido y creación de netlist para optimizar el tiempo de ejecución y uso de memoria para diseños muy grandes.
- Un flujo rápido de orden de cambio de ingeniería que agiliza la extracción al analizar solo aquellas redes que han sido cambiadas del diseño de referencia.
- Análisis de FinFET y otros diseños de transistores avanzados.
- Extracción para procesos de patrones dobles o múltiples.
- Capacidad para crear archivos de caracterización personalizados utilizando un proceso integral modelado para capas de interconexión.

- Análisis de múltiples esquinas simultáneo, que ahorra tiempo de ejecución al examinar las esquinas relacionadas en paralelo.

## 6.4.2 Flujo de diseño de StarRC

Una vez que se completa un diseño como en la Figura 9, se debe probar la sincronización del circuito. Esto requiere de tiempo para el análisis, ya que solicitan que todas las resistencias y capacitancias parásitas se tomen en cuenta. La herramienta de StarRC usa el diseño del chip junto con la descripción del proceso para extraer millones de dispositivos parásitos.

*StarRC Extraction in the Design Flow*

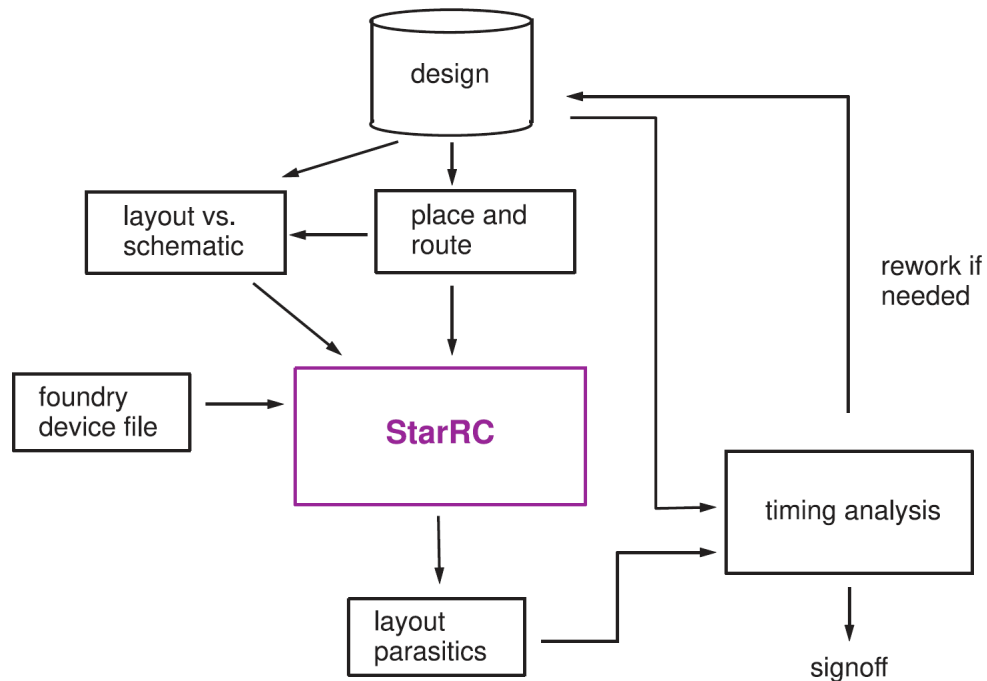


Figura 9: Flujo de diseño de StarRC

Si el diseño debe ser modificado para corregir violaciones de tiempo, los parásitos extraídos también deben actualizarse.

Extraído los parásitos también son importantes para otras herramientas como simuladores de circuitos y electro-migración, herramientas de análisis.

## 6.5 Síntesis lógica

### 6.5.1 Flujo del diseño

Para completar satisfactoriamente la etapa de síntesis lógica es necesario seguir correctamente el flujo de diseño que se tiene para no cometer errores en la síntesis, estos permiten

crear un archivo HDL hasta obtener un *netlist* final.

El flujo de diseño que se sigue es la Figura 10 este flujo fue establecido por estudiantes que trabajaron anteriormente en este proyecto [6] [12].

Al conocer adecuadamente esta etapa se pudo conocer el uso adecuado de las herramientas y se llevó a cabo la creación de un script que permite facilitar el proceso de la etapa de síntesis lógica.

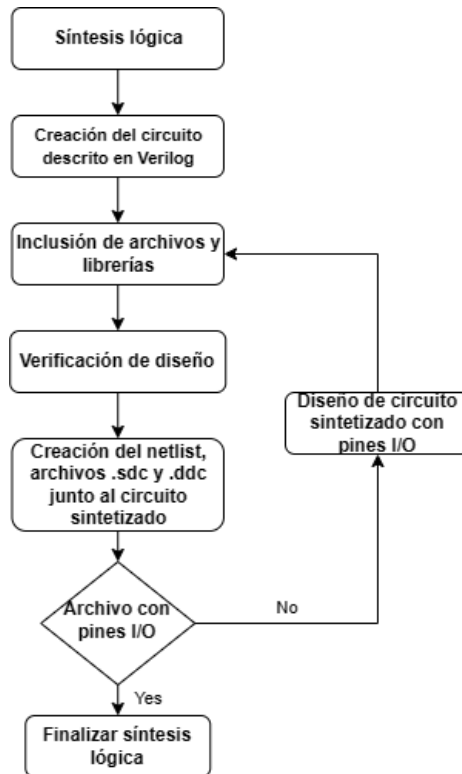


Figura 10: Diagrama del flujo de diseño

Para ello se hizo uso de un archivo *DV\_chip.script* donde en este se encontraban todos los comandos necesarios para culminar con éxito la síntesis, luego con una terminal se navega a través de los directorios para colocar de una forma ordenada los archivos de salida y no tener complejidad de saber en donde se encuentra.

---

### Script de Python para lectura universal

---

Para la primera parte de la automatización se requiere que se pueda efectuar una lectura de cualquier archivo Verilog. Se iniciaron las pruebas con una compuerta lógica NOT, esto debido a que nos es muy compleja y con facilidad se pueden detectar errores, se escaló y avanzo con una compuerta XOR para luego continuar con un Ring Oscillator.

Luego de estas pruebas se hizo la última prueba con el nano Chip para verificar que saliese bien, esto se hizo comparando con lo que se proporcionó en años anteriores y lo obtenido son idénticos.

El objetivo del script es poder leer e interpretar las entradas y salidas del módulo **chip\_SP** esto para poder generar un módulo donde se declara más adelante los pines de entrada y salida utilizando las librerías apropiadas de TSMC, este archivo verilog se puede encontrar en la sección de anexos.

Con este script de Python se puede agilizar de una gran manera el proceso de Síntesis Cell I/O, en esta etapa se requiere de un módulo donde se instancia con una sintaxis en específico, está dada por la documentación de las herramientas de Synopsys y la cual es necesaria para poder agregar de forma correcta los pines de entrada y salida utilizando la librería de TSMC.

Este es el código de Python que se creó y se explicara con más detalle y el código completo con nombre **lectrua.py** se encuentra en la sección de anexos.

## 7.1 Código explicado

---

```
1 from asyncore import write
2 from contextlib import ContextDecorator
3 import re
```

---

Cuadro 3: Importación de librerías útiles en Python

En las primeras líneas se importan las librerías que se utilizarán para que el código funcione adecuadamente, estas no son librerías que se deban de instalar, ya que Python cuenta ya con estas, la primera es para poder escribir en un archivo que se crea o existente y proviene del `asyncore` que es para controlar sockets de forma asincrónica.

La segunda es para que se puedan modificar funciones que modifican la funcionalidad de otras funciones para hacer más corto el código y proviene del `contextlib` que es un módulo propio de Python que contiene utilidades de administrador de contexto.

Y la última importación es “re” este se emplea para expresiones regulares de Python. [3](#)

---

```
1 archivo = open("circuito.v", 'r')
2 lista=archivo.readlines()
3 archivo.close()
```

---

Cuadro 4: Lectura de un archivo

En esta línea se abre el archivo deseado en forma de lectura y con la siguiente línea se almacena dentro de “lista” y se leen todas las líneas una por una sin importar el tamaño del archivo y se almacena dentro de una lista. [4](#)

---

```
1 sup=lista.index('endmodule\n')
```

---

Cuadro 5: Límite de lectura

Con esta línea lo que se hace es declarar que lea el primer módulo del circuito hasta el primer “endmodule” con esto aseguramos que se haga de forma correcta, ya que solo nos interesa leer con exactitud los pines de entrada y salida del módulo que describe el funcionamiento total del circuito. [5](#)

---

```

1 inputsito=[]
2 outputsito=[]
3 quitacorinput = []
4 quitacoroutput = []
5 solocorinput = []
6 solocorinput1 = []
7 solocoroutput = []
8 solocoroutput1 = []
9 sincorinput = []
10 sincoroutput = []

```

---

Cuadro 6: Listas utilizadas

Toda lista es necesaria para hacer filtros y poder contener dentro de estas variables con un mejor orden para poder escribirlas de una mejor manera, con lo que se tienen 10 listas para hacer bien la escritura del nuevo archivo que se necesita. [6](#)

---

```

1 for a in range(sup):
2     b=lista[a]
3     if b.find("input")>=0:
4         inputsito.append(re.sub("input |;", "", b.rstrip('\n')))
5     if b.find("output")>=0:
6         outputsito.append(re.sub("output |;", "", b.rstrip('\n')))

```

---

Cuadro 7: Búsqueda dentro del rango establecido

Dentro de este for se hace una búsqueda dentro del rango que definimos arriba que es hasta el primer “endmodule” dentro de este se buscara todos los “inputs” y todos lo “outputs” y los colocara dentro la lista correspondiente a cada uno, estos sin importar si es una entrada o salida de varios bits. [7](#)

---

```

1 for c in inputsito:
2     if c.find('[')>=0:
3         quitacorinput.append(re.sub("\[.*?\]", "", c))
4         solocorinput.append(c)
5         solocorinput1.append(re.sub("\[.*?\]", "", c))
6     else:
7         quitacorinput.append(c)
8         sincorinput.append(c)

```

---

Cuadro 8: Búsqueda del número de bits que conforma la señal

---

```

1 for c in outputsito:
2     if c.find('[')>=0:
3         quitacoroutput.append(re.sub("\[.*?\]", "", c))
4         solocoroutput.append(c)
5         solocoroutput1.append(re.sub("\[.*?\]", "", c))
6     else:
7         quitacoroutput.append(c)
8         sincoroutput.append(c)

```

---

Cuadro 9: Búsqueda del número de bits que conforma la señal parte de salidas

Dentro del for se hace una búsqueda de un corchete, este con el objetivo de saber si alguna de las entradas o salidas no es única y tiene más de un bit, con lo que es importante identificar esta característica ya que si lo tiene se almacena dentro de una lista, de igual manera se realizan dos listas extras para poder separar únicamente los valores que tienen los corchetes y la palabra que pertenece a esta característica. En ambos casos se hace para los “inputs” y para los “outputs”. [89](#)

---

```

1 zinput2 = str('module')
2 with open("montar.v", 'r') as file:
3     for line in file:
4         if zinput2 in line:
5             modulito = line
6             break

```

---

Cuadro 10: Búsqueda especial para encontrar caracteres

Esta función se encarga de almacenar dentro de la variable zinput2 una palabra que se desea buscar dentro del archivo, siendo esta “module” porque es está donde se tiene que ir a buscar el nombre del módulo que sé instancia para que este sea colocado en el nuevo módulo que se crea donde se agregan los pines de entrada y salida para la segunda etapa de síntesis cell I/O con lo que es importante tener el nombre correcto del módulo. [10](#)

---

```

1 buscarini = modulito.find("module")
2 buscarfin = modulito.find("(")
3 mensaje9 = modulito
4 startLoc = buscarini
5 endLoc = buscarfin
6 mensaje9b = mensaje9[startLoc: endLoc]

```

---

Cuadro 11: Filtro para la palabra que se buscó anteriormente en el cuadro anterior



De la línea 22 a la 27 se tienen variables donde se buscan palabras específicas como “module” y “(“. Esto con el fin de poder hacer unos filtros en el texto y que puedan quedar ordenados y con la estructura requerida, así mismo al no conocer el número específico de entradas y salidas se buscan esas palabras para recortarlas y que esos sean los parámetros que se requieren como inicio y final. [11](#)

---

```
1 namemodulito = mensaje9b.replace('module ', '')
2 file.close()
```

---

Cuadro 12: Filtro del nombre módulo y almacenaje en una lista diferente

En esta parte se hacen un filtro para el nombre del módulo y se almacena en la variable “namemodulito” donde se reemplaza el nombre “module” y no se deja vacío. Luego se cierra el archivo “file”, ya que no se requiere más para los parámetros que se necesitan. [12](#)

---

```
1 f = open('circuito.v', 'w')
```

---

Cuadro 13: Se crea el archivo final que se utilizará

En esta línea se abre el archivo “circuito.v” y se le especifica que será un archivo donde podremos escribir dentro del mismo, si este archivo no existe la misma función creará el archivo con el nombre especificado. [13](#)

---

```
1 f.write("module circuit(" + new_separadoinput3 + "," + new_separadooutput3 +
↪ ");")
2 f.write( '\n\n')
```

---

Cuadro 14: Se escriben las variables dentro de la descripción del módulo nuevo

Con “f.write” escribimos dentro del archivo las variables que necesitamos que con anterioridades se hicieron uso de los filtros para que quedasen de la forma en que se requerían, ya que comprendían dentro caracteres que para esta línea donde se declaran las variables de entrada y salida. [14](#)

---

```

1 for i in separadooutput:
2     f.write('\t' + "output " + i + ";" + '\n')
3 for i in separadoinput:
4     f.write('\t' + "input " + i + ";" + '\n')
5 for i in separadoinput:
6     f.write('\t' + "wire " + i + "_w;" + '\n')
7 for i in separadooutput:
8     f.write('\t' + "wire " + i + "_w;" + '\n')
9 f.write( '\n')

```

---

Cuadro 15: Escritura dentro del archivo todas las entradas, salidas y los wires

Con el ciclo “for” escribimos dentro del archivo todas las entradas y salidas al igual que los wires pertenecientes a cada uno de ellos, con esto podemos escribir N cantidades de entradas y salidas con la certeza de que no hará falta ninguno. [15](#)

---

```

1 extra_line = namemodulito + " circuit_new ("
2
3 for i in separadooutput:
4     extra_line = extra_line + "." + i + "(" + i + "_w), "
5 for i in separadoinput:
6     extra_line = extra_line + "." + i + "(" + i + "_w), "
7 extra_line2 = extra_line + ");"

```

---

Cuadro 16: Línea para descripción de instancias

En el archivo nuevo se necesita especificar una línea donde se escribe el nombre del módulo que sé instancia seguida de un nombre diferente al módulo que sé instancia dentro de este deben de ir todas las entradas y salidas y se colocan dentro de un segundo paréntesis el wire perteneciente al mismo. [16](#)

---

```

1 #filtro de la linea extra
2 lineaextra = extra_line2.replace(', )', ')')
3 f.write(lineaextra + '\n')
4 f.write( '\n')

```

---

Cuadro 17: Filtro para signos de puntuación

En este filtro de la misma línea se reemplaza en la última palabra la última coma, dejando únicamente el paréntesis para que no haga conflictos con la sintaxis que se desea obtener en esa línea en específico. [17](#)

---

```

1 f.write("//Entradas (C):" + '\n')
2 contadorsito = 0
3
4 for i in sincorinput:
5     f.write("PDDW0204SCDG IN" + f"{contadorsito:000}" + " (1'b0,1'b0,1'b1,"
6         ↪ + i + "," + i + "_w," + "1'b0,1'b1);" + '\n')
7     contadorsito = contadorsito + 1
8
9 e = 0
10 for c in solocorinput:
11     k = int(c[c.index(':') + 1:c.index(')')])
12     j = int(c[c.index('(') + 1:c.index(':')])
13     p = solocorinput1[e]
14     for k in range(j+1):
15         f.write("PDDW0204SCDG IN" + f"{contadorsito:000}" + "
16             ↪ (1'b0,1'b0,1'b1," + p + "[" + str(k) + "], " + p + "_w" + "[" +
17             ↪ str(k) + "], " + "1'b0,1'b1);" + '\n')
18         contadorsito = contadorsito + 1
19     e = e + 1

```

---

Cuadro 18: Escritura de los pines de entrada y salida con librería de TSMC

Se hace un comentario donde se indicará que seguido del mismo se escribirán dentro del nuevo documento únicamente las entradas. Para identificar bien las diferencias en las entradas se hicieron dos listas y se usan respectivamente para cada ciclo for. Dentro de la línea que se escribe se especifica la librería de TSMC “PDDW0204SCDG” que es tanto para entradas como para las salidas, luego se define un nombre para la entrada, en este caso se definió como “IN0” como la primera entrada, seguido y por último se tiene dentro del paréntesis “1'b0,1'b0,1'b1,clk,clk\_w,1'b0,1'b1” de esta manera es como se define una entrada, para el cual se utilizó de la documentación porque utilizando la tabla de verdad [13](#) se puede encontrar que sé si define de otra forma se pueden tener errores. [18](#)

El mismo procedimiento se aplica para las salidas, la diferencia se basa en cómo se define dentro del paréntesis la posición de los bits y con la tabla de verdad se determina que es un pin de salida, con lo que la escritura es muy diferente a la usada para las entradas.

1. Para el primero ciclo for se utiliza la lista de “sincorinput”, ya que esta lista contiene únicamente las entradas que no contienen la característica especial de incluir los corchetes, por el hecho de que estos nos indican que la entrada es de más de un bit.

2. Para el segundo ciclo for se utilizan la lista de “solocorinput” dentro se definen 3 variables nuevas donde las primeras dos son para obtener el número de bits que se encuentra del corchete y la tercera para indicar que el segundo ciclo dentro del mismo pare. Dentro del primer for se hace uno extra que es donde se escribe la línea de cómo se definen una entrada, con la tercera variable de nombre “e” se hace que vaya contando y sumando en una unidad para escribir todos los bits necesarios que se definen dentro de los corchetes y para que este no se repita y cuente dos veces se reinicia fuera del primer ciclo for.

---

```
1 f.write("//Pines de VDD y VSS")
2 f.write('\n' + "PVDD1CDG PVDD() ;"+'\n')
3 f.write("PVSS1CDG PVSS() ;"+'\n')
4 f.write("endmodule")
5 f.close()
```

---

Cuadro 19: Escritura de pines de alimentación

Por último, se definen los pines de alimentación, esto para facilitar a la etapa de síntesis física el no tener que agregar manualmente estos pines, con lo que se utiliza el nombre apropiado que nos indica para utilizar la librería de TSMC y poder definirlos siendo “PVDD1CDG” para VDD y “PVSS1CDG” para VSS respectivamente, seguido se coloca el nombre con el cual se identificará en la etapa de síntesis física añadiendo una letra **P** antes del propio nombre.[19](#)

---

## Planificación de la automatización

---

### 8.1 Script de BASH

En esta etapa se diseñó un script de bash donde se tienen comandos para mostrar mensajes en la terminal, eliminar archivos y carpetas, ejecutar archivos de Python, ingresar y regresar de directorias, crear carpetas, copiar archivos y mover otros para mantener un orden y no tener complejidad a la hora de buscar un archivo en específico.

Por último, otorgar permisos a archivos para ejecutar el script donde se encuentran los comandos de Design Compiler y realizar efectivamente la síntesis lógica. [11].

En código completo con nombre **script.bash** que se utilizó se encuentra en la sección de anexos.

#### 8.1.2 Resultados del script

En el escritorio de la propia computadora se tienen los mismos archivos que se utilizan en el servidor con fines de realizar pruebas sin arruinar el progreso que se tienen con las otras etapas y se actualiza el servidor cuando se efectúan cambios importantes en la etapa de síntesis lógica.

En la Figura [11] se pueden ver tanto las carpetas como los scripts que se utilizan para la automatización de síntesis lógica.

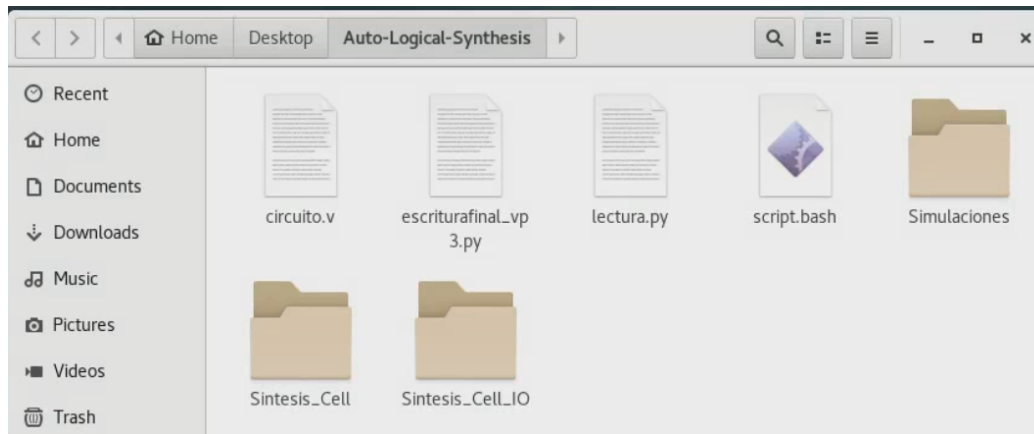


Figura 11: Carpeta ubicada en el escritorio

Dicho script se ejecuta por partes, esto para tener un orden y no obviar ningún paso del flujo del diseño. Dentro de la primera carpeta que es **Sintesis\_Cell** se encuentran dos carpetas y un script más que es el que se usa para **Design Compiler** como se puede ver en la Figura 12.

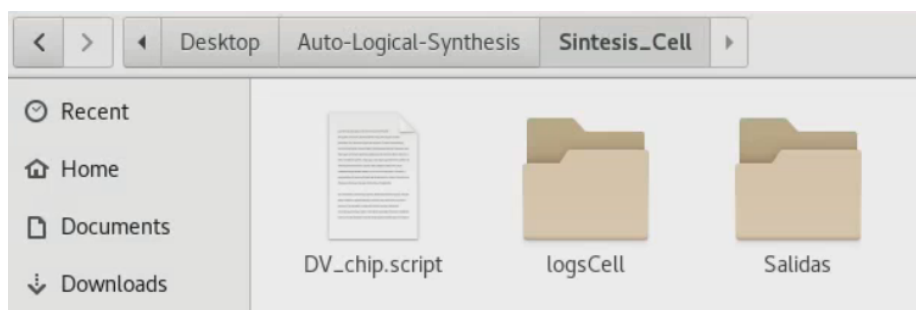


Figura 12: Archivos dentro de Sintesis Cell

En la carpeta de salidas se encuentran los archivos de salida del primer **DV\_chip.script** que se ve en la Figura 12 este script que contiene los comandos necesarios para obtener la primera fase de la síntesis lógica en la cual aún no contiene los pines de entrada y salida.

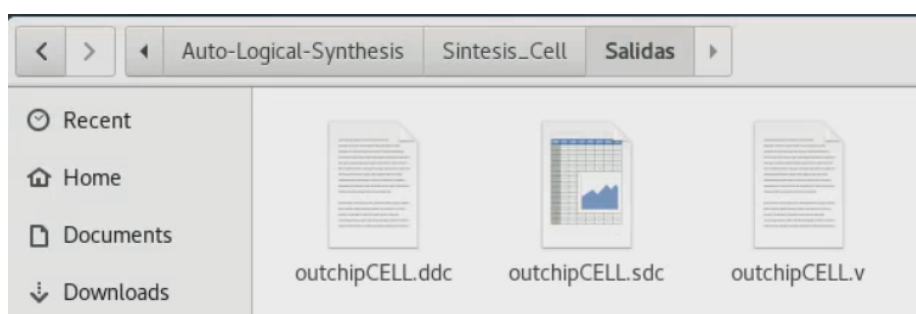


Figura 13: Archivos de salida dentro de la carpeta de salidas

Para la segunda carpeta se tiene un nombre diferente que es **Sintesis\_Cell\_IO**, dentro se encuentran dos carpetas más y un script que contiene los comandos para llevar a cabo la segunda fase de la síntesis lógica como se ve en la Figura [14](#).

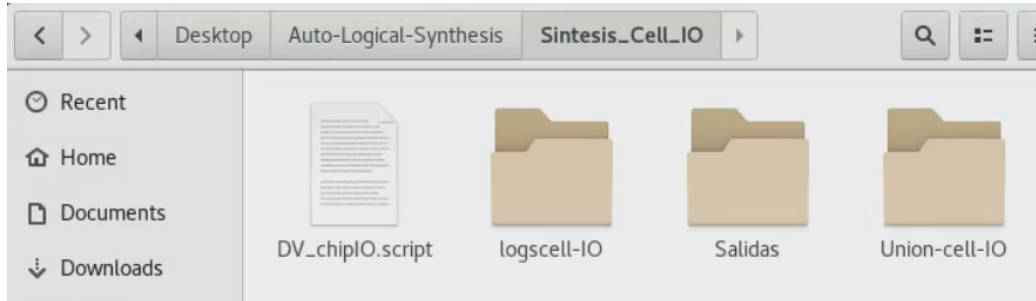


Figura 14: Archivos dentro de Sintesis Cell IO

Dentro de la misma hay dos carpetas importantes con nombre de **Salidas** y **Union-cell-IO**. Dentro de la carpeta de **Union-cell-IO** se crean tres archivos de los cuales el más valioso es el que tiene de nombre **modulochipio.v** este es imperante antes de ejecutar la segunda parte de la síntesis lógica.

Dentro del archivo se encuentran descrito utilizando las librerías de TSMC para definir los pines de entrada y salida, luego se hace una copia de este archivo con el **outchiipCELL.v** que viene de la etapa anterior que y se juntan ambos en uno solo con nombre **out-FINAL.v** para luego pasar a la ejecución del archivo y finalizar la síntesis lógica. [15](#)

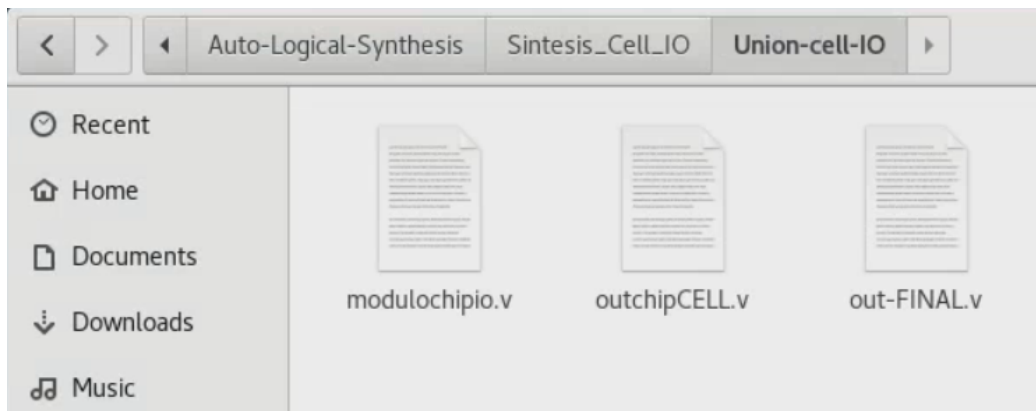


Figura 15: Archivos dentro de Union-cell-IO

Dentro de la carpeta de **Salidas** se encuentra los archivos resultantes y esperados de la síntesis lógica, estos son derivados del segundo script **DV\_chip.script** que contiene los comandos necesarios y se utilizan los archivos ya necesarios para completar la síntesis lógica. [16](#)

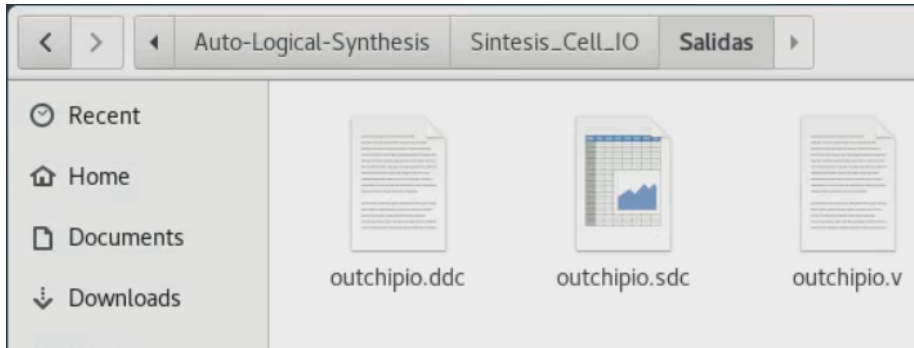


Figura 16: Archivos de salida con pines de entrada y salida dentro de la carpeta de salida

## 8.2 Script para Design Compiler

Los exalumnos de la carrera de ingeniería electrónica Sophia Cardona<sup>[12]</sup> y Elmer Torres<sup>[6]</sup>, trabajaron en este proyecto, en la síntesis lógica hace ya años anteriores, con lo que se tenía un archivo donde se tienen todos los comandos necesarios para poder realizar la síntesis lógica adecuadamente y no obtener ningún error.

En este archivo se importan las librerías, se hace lectura del archivo que describe un circuito en lenguaje verilog, se definen los parámetros para el diseño y se generan por último los archivos de salida con extensión *.sdc*, *.ddc* y *.v*

El código completo con nombre **DV\_chip** que se utilizó se puede encontrar en la sección de anexos.

En esta parte se explicará más a detalle la importancia de cada comando y por qué se utiliza.

### 8.2.1 Inclusión de librerías

---

```
1 lappend search_path /home/nanoelectronica/Documents/libs
```

---

Cuadro 20: Código de inclusión de librerías utilizando la ruta de ubicación

Se especifica la ubicación del archivo del diseño utilizando dos formas, utilizando la ruta completa o solo el nombre del archivo. Si se usa el nombre del archivo Design Compiler emplea la ruta de búsqueda que es definida en la variable *search\_path*. Con esta forma, la herramienta busca los archivos de diseño que comienzan con él más a la izquierda del directorio donde se especifica la variable *search\_path* y utiliza el primer archivo de diseño que encuentra y para especificar otro directorio, además de la ruta de búsqueda predeterminada se usa "lappend search\_path project"<sup>[20]</sup>



---

```
1 set link_library " * tcb018gbwp7ttc.db tcb018gbwp7twc.db tcb018gbwp7tbc.db  
  ↪ tpd018nvtc.db"
```

---

Cuadro 21: Restricciones de celdas de la biblioteca

Design Compiler le permite restringir la selección de celdas de la biblioteca para que se elijan de un subconjunto de las bibliotecas especificadas por la variable `link_library`. Normalmente, las celdas de la biblioteca se seleccionan de cualquier biblioteca en la variable `link_library`. Para especificar un subconjunto de bibliotecas pertenecientes a la variable `link_library`, use el comando `set_link_library_subset` con el argumento `library_list`. Para cualquier nombre de referencia que aparece en una biblioteca del subconjunto, solo las celdas de la biblioteca dentro del subconjunto en uso.

Con nombres de referencia que no aparecen en el subconjunto, se seleccionan celdas de biblioteca desde cualquier biblioteca en la variable `link_library`. [21](#)

---

```
1 set target_library "tcb018gbwp7ttc.db"
```

---

Cuadro 22: Restricción de un bloque de la celda

Design Compiler le permite restringir la optimización de un bloque de diseño utilizando un subconjunto de la biblioteca de destino. Normalmente, la optimización puede seleccionar cualquier celda de biblioteca de la biblioteca de destino. Si no se especifica el argumento `library_list`, se pueden emplear las bibliotecas enumeradas en la variable `target_library`. Para especificar las celdas de la biblioteca que no se pueden usar para la optimización, incluso si están dentro de las bibliotecas especificadas por el argumento `library_list`, use la opción `-dont_use`. [22](#)

---

```
1 read_file -format verilog {../circuito.v}
```

---

Cuadro 23: Lectura del archivo descrito en HDL

El comando `read_file` analiza el diseño luego lo traduce en una tecnología independiente (GTECH) `read_file -format verilog my_design.v` y los formatos de HDL que soporta son:

- `.v`
- `.verilog`
- `.v.gz`
- `.verilog.gz` extensions

Si no especifica el formato de diseño, el comando `read_file` infiere el formato basado en la extensión del archivo. Si no se usa una extensión conocida, la herramienta usa el formato `.ddc`.

Soportado las extensiones para la inferencia automática no distinguen entre mayúsculas y minúsculas. El comando `read_file` puede leer archivos comprimidos para todos los formatos, excepto el `.ddc` formato, que se comprime internamente a medida que se escribe. [14] [15] [23]

### 8.2.2 Revisión de diseño

- El comando **`reset_design`** remueve toda la información de diseño, incluyendo los relojes, entradas y salidas, los grupos de path, condiciones de operación, rangos del timing y los modelos de las interconexiones. Como resultado se obtiene un equivalente a iniciar el proceso de diseño a sus ajustes predeterminados, esto con el objetivo de no tener errores de utilizar información de otras ejecuciones.
- El comando **`set_max_area 0`** nos proporciona la máxima área representa el número de compuertas en el diseño, no el área física que el diseño ocupa. Los requisitos de área para el diseño se definen con el diseño más pequeño que cumple con el objetivo de mayor rendimiento. Design Compiler calcula esta área sumando las áreas de cada componente en el nivel más bajo del diseño y el área de las redes y las áreas de celdas y de red dependen mucho de la tecnología que se utilizara. Design Compiler obtiene toda esta información de las bibliotecas lógicas.
- El comando **`link`** localiza la referencia para cada celda en el diseño.
- El comando **`uniquify`** y **`link`** juntos hacen que se aseguren que en las librerías se encuentre las celdas que se quieren sintetizar y verifica que no existan duplicidades en las librerías.
- El comando **`create_clock clk -period 40 -waveform 0 20`** no crea un circuito síncrono, se necesita crear un reloj con un periodo de 40 nanosegundos y un ciclo de trabajo del 50%, este se utiliza en las restricciones de las rutas de datos, los circuitos como son los flip-flops del mismo diseño.
- Puede usar el comando **`check_design`** después de la compilación para identificar las instancias de celda que tienen salidas no conectadas y así comprueba la representación interna del diseño actual para la consistencia y emite mensajes de error y advertencia según corresponda. [15] [24]

---

```
1 reset_design
2 set_max_area 0
3 link
4 uniquify
5 create_clock clk -period 40 -waveform {0 20}
6 check_design
```

---

Cuadro 24: Comandos de revisión de diseño

### 8.2.3 Compilación

Para llevar a cabo la síntesis se debe de ejecutar el comando de `compile`", dentro de las opciones varias que este contiene se pueden configurar el comportamiento como el `map_effort` que es para establecer el esfuerzo que realiza en el mapeo y se pone en el máximo esfuerzo que este puede realizar, con el `area_effort`, se utiliza para configurar el esfuerzo que hará para configurar el área del diseño y de igual forma se coloca en el máximo. [15](#) [25](#)

---

```
1 compile -map_effort high -area_effort high
```

---

Cuadro 25: Comando de compilación

### 8.2.4 Reportes

El comando `report_area` proporciona información de área y estadísticas sobre el diseño actual.

El comando `report_design` muestra las condiciones de funcionamiento, el modelo y el modo de carga del cable, los rangos de tiempo, entrada y salida interna, y arcos de temporización deshabilitados definidos para el diseño actual.

El comando `report_power` divide los números de potencia en dos categorías: netlist power (basado en celdas que ya están en el diseño) y potencia estimada (una estimación del árbol del reloj energía). [15](#) [26](#)

---

```
1 report_area
2 report_design
3 report_power
```

---

Cuadro 26: Comandos de reportes

### 8.2.5 Outputs

---

```
1 write -format ddc -h -o ./Salidas/outchipCELL.ddc
2 write -hierarchy -format verilog -output ./Salidas/outchipCELL.v
3 write_sdc ./Salidas/outchipCELL.sdc
```

---

Cuadro 27: Escritura de archivos de salida

Estos comandos son para generar archivos de salida en un formato específico que el usuario desee, con lo que se hacen 3 para hacer tanto simulaciones como poder pasar a la etapa de diseño de Síntesis Física 15 27

---

## Modificaciones importantes de archivos

---

Para poder generar el circuito que describe en lenguaje HDL se hizo uso de un script de python, este se encarga de generar un archivo nuevo con extensión `.v` donde se tiene todo el funcionamiento al igual que los pines de entrada como de salida.

El archivo de python con nombre `escriturafinal_vp3.py` se encuentra en la sección de anexos. En este capítulo se habla específicamente de los cambios importantes que se realizaron a este archivo.

### 9.1 Modificación del texto

En el script original de Python donde es donde se define el comportamiento del circuito, cuáles serán los pines de entrada/salida y dentro del mismo se incluye el texto que se mostrará en la salida luego de pasar satisfactoriamente y sin errores el flujo de diseño para luego ser fabricado. [29](#)

En el archivo de Python original se pueden ver los dos textos que se tenían siendo ambos:

---

```
1 texto = "Hola, soy El Gran Jaguar, el primer nanochip elaborado en
↪ Centroamerica por una Universidad. Desarrollado completamente en la
↪ Universidad del Valle de Guatemala por alumnos graduandos de Ingenieria
↪ en Electronica entre 2019 y 2022."
```

---

Cuadro 28: Textos originales

---

```

1 texto2 = "El equipo encargado de mi creacion se conformo por: Geovanni
↳ Flores, Matthias Sibrian, Steven Rubio, Julio Shin, Karol Cardona, Luis
↳ Najera, Luis Abadia, Joel Gonzalez, Jose Ayala, Jose Ruiz, Ricardo
↳ Giron, Carlos Esquit, Charlie Ayenci, Elmer Torres, Gerardo Cardoza,
↳ Jonathan de los Santos, Antonio Altuna, Kurt Kellner, Luis Rivera,
↳ Carlos Letona Torres, Helder Ovalle, Jose Ponce, Israel Arevalo, Allison
↳ Chocooj, Paola Mendizabal, Diego Equite, Estuardo Mancio, Stefan
↳ Schwendener y Luis Gomez."

```

---

Cuadro 29: Segunda parte de los textos originales

Luego de pasar de completar la síntesis lógica con este texto se pueden ver los resultados con la herramienta de **Design Vision** donde se ve la estructura de la lógica combinacional que se utiliza para mostrar el texto. [18](#)

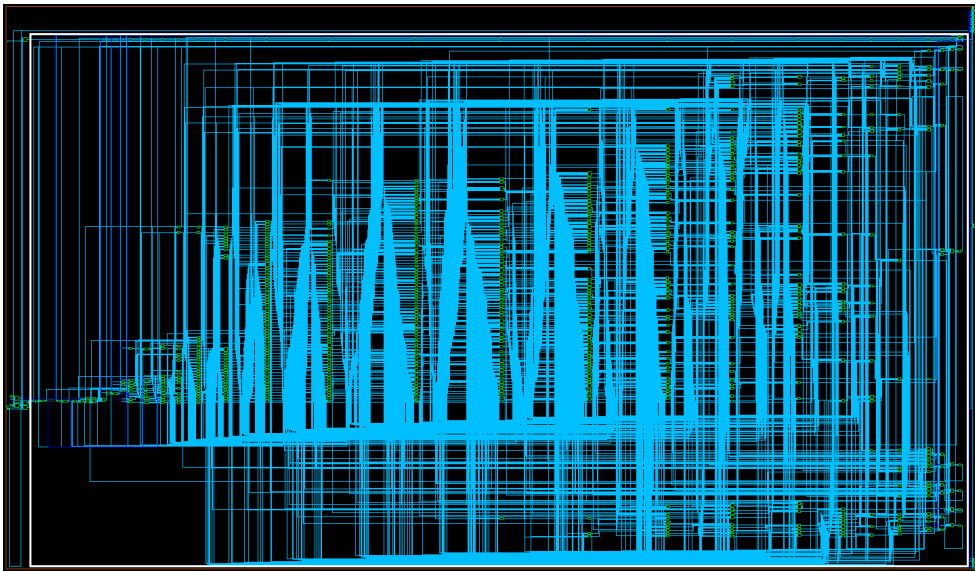


Figura 17: Estructura de la lógica combinacional del texto anterior

El mayor cambio que se puede hacer para ambos textos es que el **texto** [30](#)[32](#) y **texto2** [33](#)[34](#) se unificaron en **texto**, en este se colocaron los nombres completos de cada uno de los integrantes del equipo que pertenecen a la creación, la idea principal del motivo de fabricar el primer nano Chip en Guatemala y la historia del progreso que ha tenido a lo largo desde los inicios de estudio y diseño, este se puede ver:

---

```

1 texto = "Hola, soy El Gran Jaguar, el primer nanochip elaborado en
↳ Centroamerica por una Universidad. Desarrollado completamente en la
↳ Universidad del Valle de Guatemala por alumnos graduandos de Ingenieria
↳ en Electronica entre dos mil diecinueve y dos mil veintidos."

```

---

Cuadro 30: Primer texto modificado

---

1 "El equipo encargado de mi creacion se conformo por: Carlos Alberto Esquit  
↪ Hernandez, Jonathan Alberto de los Santos Chonay, Kurt Emmanuel Kellner  
↪ Juarez, Luis Alberto Rivera Estrada, Luis Arturo, Najera Vasquez, Steven  
↪ Hiram Rubio Vasquez, Marvin Geovanni Flores Espino, Matthias Sibrian  
↪ Illescas, Julio Enrique Shin Jo, Karol Sophia Cardona Polanco, Luis  
↪ Estuardo Abadia Lopez, Joel Andres Gonzalez Herrera, Jose Adrian Ayala  
↪ Escobar, Jose Alejandro Ruiz Orozco, Juan Ricardo Giron Rubio, Charlie  
↪ Ayenci Cruz Giron, Elmer Otoniel Torres Garza, Gerardo Cardoza, Antonio  
↪ Altuna Hernandez, Carlos Andres Letona Torres, Helder Arnoldo Ovalle  
↪ Barrios, Jose David Ponce del Cid, Israel Antonio Arevalo Gonzalez,  
↪ Allison Estuardo Aguilar Chocooj, Paola Alejandra Mendizabal Batres,"

---

Cuadro 31: Segunda parte del primer texto modificado

---

1 **texto** = "Diego Rodrigo Equite Pirir, Estuardo Geovany Mancio Ruballos,  
↪ Stefan Alexander Schwendener Farrington y Luis Ricardo Gomez Velasquez.  
↪ La idea de fabricar un CHIP nacio a inicios del dos mil diecinueve,  
↪ donde los alumnos graduandos consiguieron el apoyo de Euro-practice para  
↪ financiar gran parte de la fabricacion de El Gran Jaguar. Tambien a  
↪ traves de IMEC se tuvo acceso a librerias de fabricacion implementadas  
↪ por la Taiwan Semiconductor Manufacturing Company en un proceso de  
↪ ciento ochenta nanometros. Desde entonces se hicieron importantes  
↪ avances en esta linea de investigacion, obteniendo importantes  
↪ conclusiones y recomendaciones para continuar con el proyecto.  
↪ Actualmente en el 2022, los alumnos trabajan en encontrar soluciones a  
↪ los errores de densidad, errores en la generacion del deck resultado de  
↪ la extraccion de parasitos. Al mismo tiempo se realizan pruebas fisicas  
↪ en un FPGA con el fin de validar el resultado de la sintesis logica.  
↪ Tambien se trabaja para automatizar todo el proceso logrado a lo largo  
↪ de la linea de investigacion desde la instalacion y actualizacion de las  
↪ aplicaciones de Synopsys hasta la simulacion final del circuito. El Gran  
↪ Jaguar es un proyecto ambicioso que dara paso a una infinidad de  
↪ oportunidades en la investigacion y desarrollo de nuevas tecnologias y  
↪ aplicaciones en beneficio de la sociedad global y principalmente en la  
↪ sociedad guatemalteca."

---

Cuadro 32: Continuación del primer texto modificado

Con lo que se tomó la decisión de tener el **texto2** en inglés para tenerlo en dos idiomas diferentes.

---

```
1 texto2 = "Hello, I am El Gran Jaguar, the first nanochip made in Central
  ↳ America by a University. Completely developed at the Universidad del
  ↳ Valle de Guatemala by undergraduate students of Electronic Engineering
  ↳ between 2019 and 2022. The team in charge of my creation was made up of:
  ↳ Carlos Alberto Esquit Hernandez, Jonathan Alberto de los Santos Chonay,
  ↳ Kurt Emmanuel Kellner Juarez, Luis Alberto Rivera Estrada, Luis Arturo
  ↳ Najera Vasquez, Steven Hiram Rubio Vasquez, Marvin Geovanni Flores
  ↳ Espino, Matthias Sibrian Illescas, Julio Enrique Shin Jo, Karol Sophia
  ↳ Cardona Polanco, Luis Estuardo Abadia Lopez, Joel Andres Gonzalez
  ↳ Herrera, Jose Adrian Ayala Escobar, Jose Alejandro Ruiz Orozco, Juan
  ↳ Ricardo Giron Rubio, Charlie Ayenci Cruz Giron, Elmer Otoniel Torres
  ↳ Garza, Gerardo Cardoza, Antonio Altuna Hernandez, Carlos Andres Letona
  ↳ Torres , Helder Arnoldo Ovalle Barrios, Jose David Ponce del Cid, Israel
  ↳ Antonio Arevalo Gonzalez, Allison Estuardo Aguilar Chocooj, Paola
  ↳ Alejandra Mendizabal Batres, Diego Rodrigo Equite Pirir, Estuardo
  ↳ Geovany Mancio Ruballos, Stefan Alexander Schwendener Farrington and "
```

---

Cuadro 33: Segundo texto modificado

---

```
1 texto2 = "Luis Ricardo Gomez Velasquez. The idea of manufacturing a CHIP was
  ↳ born at the beginning of 2019, when graduating students obtained the
  ↳ support of Euro-practice to finance a large part of the manufacture of
  ↳ El Gran Jaguar. Also through IMEC there was access to manufacturing
  ↳ libraries implemented by the Taiwan Semiconductor Manufacturing Company
  ↳ in a process of one hundred and eighty nanometers. Since then, important
  ↳ advances have been made in this line of research, obtaining important
  ↳ conclusions and recommendations to continue with the project. Currently
  ↳ in 2022, students work on finding solutions to density errors, errors in
  ↳ the generation of the deck resulting from the extraction of parasites.
  ↳ At the same time, physical tests are performed on an FPGA in order to
  ↳ validate the result of the logic synthesis. Work is also being done to
  ↳ automate the entire process achieved throughout the research line, from
  ↳ the installation and updating of Synopsys applications to the final
  ↳ simulation of the circuit. El Gran Jaguar is an ambitious project that
  ↳ will give way to endless opportunities in the research and development
  ↳ of new technologies and applications for the benefit of global society
  ↳ and mainly in Guatemalan society."
```

---

Cuadro 34: Continuación del segundo texto modificado

Se completa la síntesis lógica con el texto nuevo y cuando se usa la herramienta de **Design Vision** se puede ver el cambio enorme en la estructura de la lógica combinacional,



es un resultado esperado y notorio si se compara la cantidad de caracteres en el texto nuevo que se está utilizando.

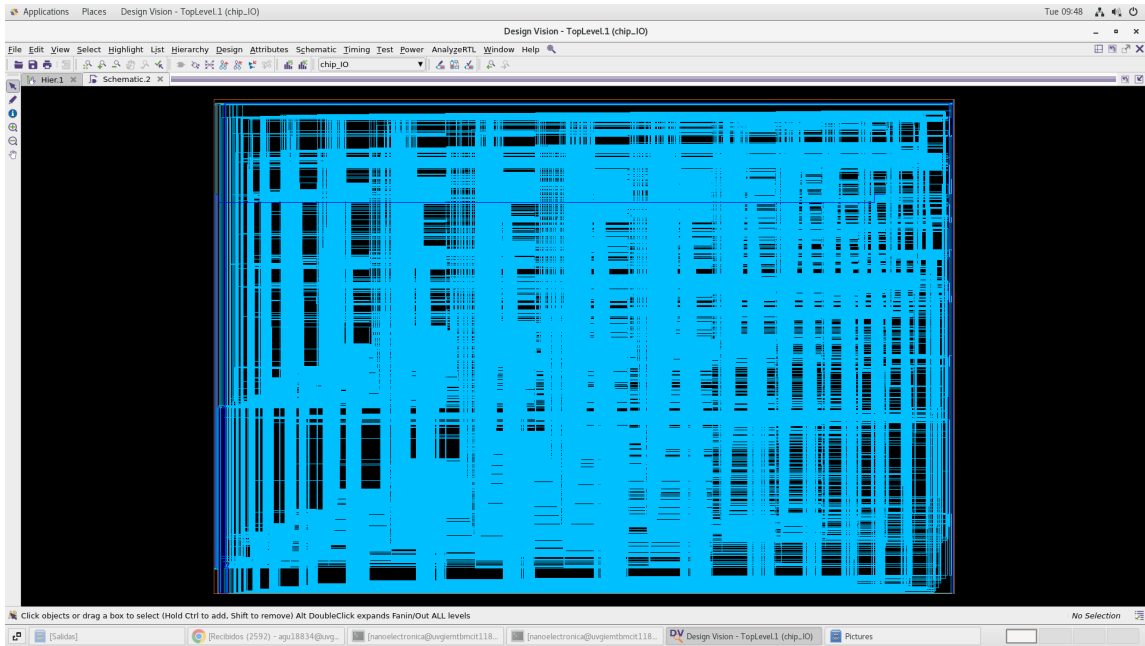


Figura 18: Estructura de la lógica combinacional del texto nuevo que es utiliza

## 9.2 Modificación del contador

Al tener estas modificaciones importantes en el texto que se mostrara es de importancia conocer el archivo de Python con el que se cuenta, ya que dentro está definido el contador, este tiene la funcionalidad de poder ir contando de carácter en carácter y poder mostrarlo adecuadamente en la salida del nano Chip. El texto original era en comparación más pequeña en palabras y caracteres al texto final que se decidió colocar [35](#), esto es un grave problema para el contador, puesto que no es suficiente para terminar de pasar por todos los caracteres, con lo que se hizo un cambio en el contador pasándolo de 8 bits a uno de 12 bits [36](#). Con esta modificación en dos líneas del código se logra contar todos los caracteres de los textos, sin que falte uno solo.

En el Python original se ve de la siguiente manera:

---

```
1 archivo.writelines("reg [8:0] contador;\n")
2 archivo.writelines("contador<=9'b00000000;\n")
```

---

Cuadro 35: Contador original

Y el Python con los cambios importantes respecto al contador se ve de la siguiente manera:

---

```
1 archivo.writelines("reg [11:0] contador;\n")
2 archivo.writelines("contador<=12'b000000000000;\n")
```

---

Cuadro 36: Contador modificado

En el archivo de Python original es la línea 13 y 62 del código que se encuentran estas dos líneas de código que se deben de modificar específicamente para aumentar el contador en bits y así realizar bien la lectura de cada uno de los caracteres dentro de los textos.

### 9.3 Modificación de la tabla de verdad

En el código de Python se hizo una última modificación que fue en la forma en cómo se selecciona el texto, este se modificó por qué se quería hacer que el nano Chip Físico sé completamente simétrico, que sea cuadrado y que pueda contar con sus 8 pines físicos, esto no era posible antes, ya que solo se contaba con un selector al tenerlo de esta forma en el código sin modificación:

S1	S0	Out
0	0	0
0	1	1
1	0	1
1	1	0

Cuadro 37: Tabla de verdad donde se selecciona el texto S1=S0=0 y S1=0 y S0=1. Para el texto2 S1=1 S0 =0 y S1=S0=1

En la tabla de verdad se hizo el cambio en la forma de poder seleccionar el texto, teniendo así el siguiente resultado:

S1	S0	Out
0	0	0
0	1	1
1	0	1
1	1	0

Cuadro 38: Tabla de verdad donde se selecciona el texto S1=S0=0 y S1=S0=1. Para el texto2 S1=0 S0 =1 y S1=1 S0=0

Esta modificación en la forma en cómo se selecciona se cambió el orden de los bits de 0 y 3 para el *texto* y para el *texto2* se utilizó 1 y 2 en los estados, con este cambio se puede

obtener el mismo resultado en la selección del texto; sin embargo, cuando este pasa por la síntesis lógica donde aún no se incorpora los pines de entrada y salida son estos mismos archivos de salida que el equipo de pruebas físicas necesitan para realizar pruebas en el FPGA *Genesys Xilinx Virtex-5 LX50T*.

Con esto se tiene la posibilidad de contar con dos selectores gracias a que se genera una compuerta XOR que es necesaria para tener los dos selectores y de igual forma poder completar los 8 pines físicos del nano Chip. [40](#)

En el Python original estas son las líneas de código que se tenían:

---

```
1 archivo.writelines("else if(select==2'b00 || select==2'b01) begin\n")
2 archivo.writelines("else if(select==2'b10 || select==2'b11) begin\n")
```

---

Cuadro 39: Selector del texto original

Y ya con la modificación en el mismo archivo se tiene como resultado:

---

```
1 archivo.writelines("else if(select==2'b00 || select==2'b11) begin\n")
2 archivo.writelines("else if(select==2'b01 || select==2'b10) begin\n")
```

---

Cuadro 40: Selector del texto modificado

En el Python original son las líneas 63, 69, 76 y 96 donde se tiene que ir a modificar para tener el orden correcto y que se logre el objetivo de generar la compuerta XOR y completar el octavo pin físico cuando este pase la etapa de *Síntesis Física* [39](#)

Este script nos genera un archivo donde se describe el funcionamiento en lenguaje HDL del nano chip, este resultado con los cambios nuevos se puede encontrar en la sección de anexos.



---

## Integración de la automatización con la etapa de síntesis física y extracción de parásitos

---

Para esta integración, se hizo uso de una carpeta compartida, esta se creó en el servidor de la Universidad del Valle de Guatemala. Esto se hace montando dentro de la propia computadora la carpeta donde se incluyen todos los archivos necesarios para realizar de igual manera el mismo proceso que se hace fuera del servidor. Para realizar las pruebas en el servidor es de vital importancia tener instaladas todas las herramientas necesarias para poder realizar el diagrama de flujo correctamente.

### 10.1 Montar servidor compartido

Para poder montar la carpeta en la computadora personal se hace uso del siguiente comando:

---

```
1 mount -t nfs 192.168.6.252:/home/nanoelectronica/Desktop/Automation  
  ↪ /mnt/nfs/var/Automation
```

---

Cuadro 41: Comando para montar un servidor compartido

Con el comando "mount" montamos el sistema de archivos que se encuentra en un dispositivo a la estructura de la computadora; con el t"se especifica el tipo de archivos del

sistema; con el "nfs.es la dirección IP del servidor NFS del servidor a compartir y que tiene acceso el resto de estudiantes. 41

Luego se especifica la dirección IP del servidor, seguido se coloca el directorio donde se encuentra la carpeta específica dentro de la computadora, se separa por un espacio y se coloca el directorio donde se encontrará específicamente el directorio dentro del servidor.

Dentro del servidor compartido se tienen varias carpetas que pertenecen a cada estudiante que está realizando una etapa del diagrama de flujo 19.

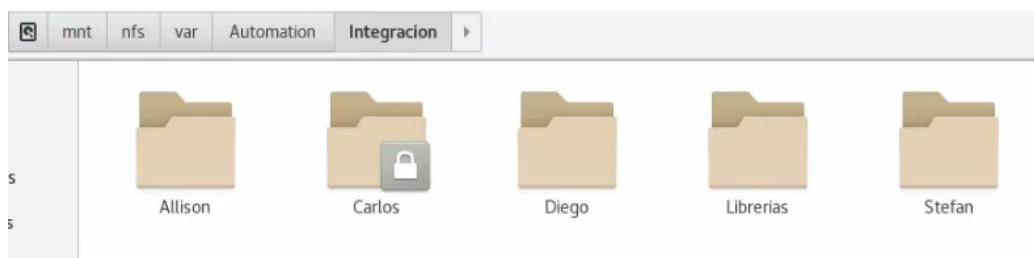


Figura 19: Carpetas dentro del servidor compartido

Dentro del servidor de igual manera se tienen que colocar las librerías que cada estudiante necesita para poder realizar con éxito la ejecución de cada etapa 19.

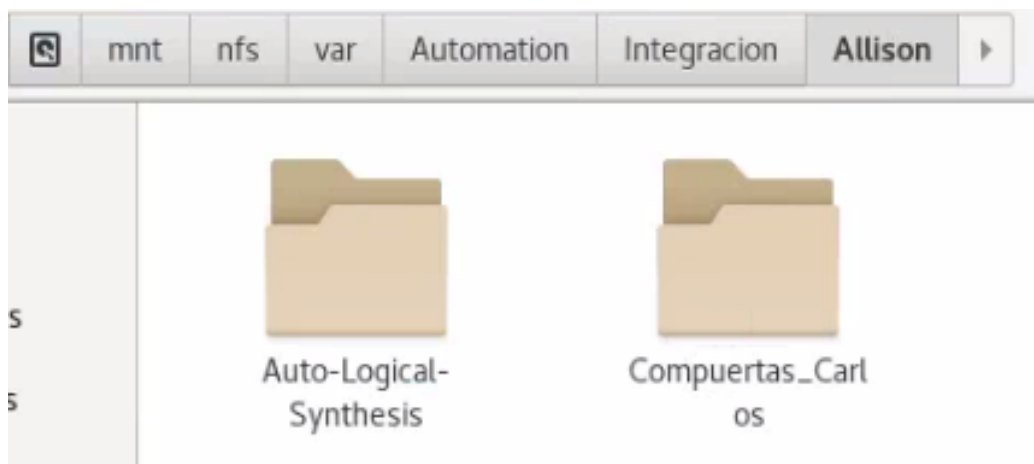


Figura 20: Carpetas contenidas dentro de la sección designada

Dentro de la carpeta que se me designo, se encuentran dos carpetas, una donde se encuentra la automatización del proceso de síntesis lógica y una segunda donde se tiene archivos destinados al estudiante Carlos Andres Letona Torres, el cual contiene compuertas lógicas que pasaron de forma exitosa la síntesis lógica 20.

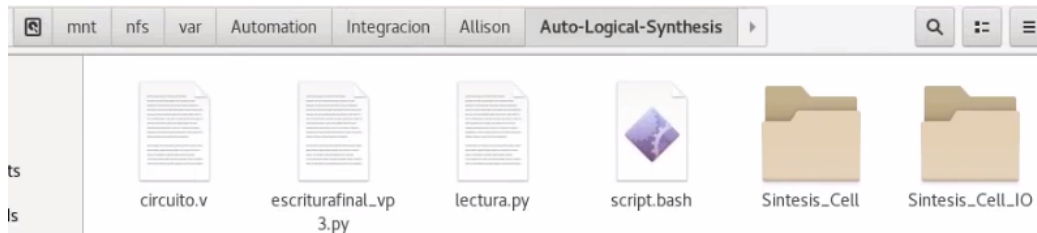


Figura 21: Automatización de síntesis lógica

## 10.2 Script modificado para uso dentro del servidor

Dentro de la carpeta de **Auto-logical-synthesis** se encuentran las carpetas destinadas y ordenadas de la forma en que se organizó para tener un fácil acceso a los archivos de salida que se generan.

En el capítulo 9 en la sección 9.0.1 se explica a detalle el script que se utiliza fuera del servidor, este cambia en una pequeña parte, ya que para automatizar de mejor manera la entrega de archivos a la etapa de síntesis física se agregaron líneas de código donde se copian los archivos de salida en la carpeta de Sintesis\_Cell\_IO a la carpeta de entradas en la carpeta de Diego [19](#).

Se adjunta las líneas de código extras que se utilizan dentro del archivo para un mayor entendimiento:

---

```

1 cd ../../../../Diego/Sintesis_Fisica/Inputs/
2 rm outchipio.ddc
3 rm outchipio.sdc
4 rm outchipio.v
5
6 cd ../../../../Allison/Auto-Logical-Synthesis/Sintesis_Cell_IO/Salidas
7
8 cp outchipio.ddc ../../../../Diego/Sintesis_Fisica/Inputs/
9 cp outchipio.sdc ../../../../Diego/Sintesis_Fisica/Inputs/
10 cp outchipio.v ../../../../Diego/Sintesis_Fisica/Inputs/
11
12 cd ../../

```

---

Cuadro 42: Comandos extras para el servidor compartido

Con esto se logra copiar los archivos de la síntesis lógica donde se colocan los pines de entrada y salida dentro de la carpeta de **Inputs** esto para entregar los archivos a la siguiente etapa con la seguridad que no serán archivos corruptos y se entregaran de forma inmediata.

[42](#)

## Integración de extracción de parásitos

Se necesitaba automatizar la verificación de extracción de parásitos, con el cual se hizo uso del mismo servidor para tener archivos compartidos con los demás integrantes del proyecto. Dentro del servidor se creó una carpeta con el nombre de **Parásitos** como se ve en la Figura 22.



Figura 22: Automatización de extracción de parásitos

Dentro de esta carpeta se tienen dos archivos importantes y una carpeta, dentro de la carpeta de **libs** se encuentran todos los archivos necesarios para llevar con éxito la verificación, el archivo con nombre **CMD\_letona.cmd** tiene los comandos de entrada a utilizar, los comandos para generar los archivos de salida, y las opciones varias para generar más información sobre dicha verificación. Por último se tiene el script con nombre **ejecutar.bash**, este realiza la automatización de forma rápida, crea la carpeta de **Salidas** y **Logs**, esto con el objetivo de ordenar los archivos de las opciones extras que se colocan en el archivo de **CMD\_letona.cmd** como se observa en la Figura 23

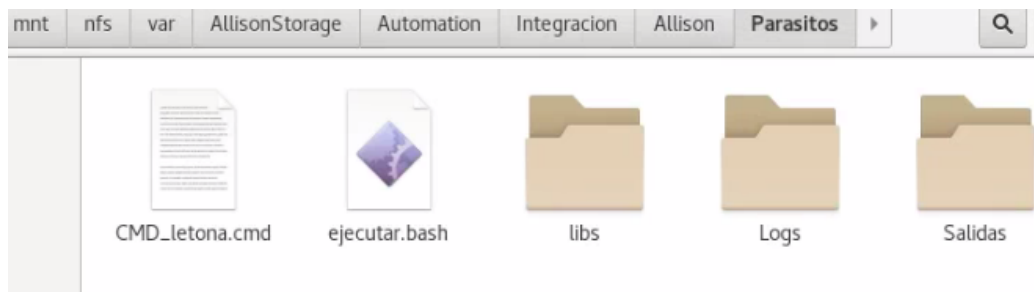


Figura 23: Dentro de la carpeta de parásitos dentro del servidor compartido

Dentro de la carpeta de **libs** se tienen 3 archivos y una carpeta, que son archivos base proporcionados por la empresa de TSMC para llevar a cabo la extracción de parásitos, dentro de la carpeta **MILKYWAY\_XTR** se encuentra la base de datos que también es proporcionada por TSMC. Los últimos dos archivos son proporcionados por etapas anteriores, que son archivos de salida, pero utilizados como entrada para la extracción de parásitos, como se ven en la Figura 24





Figura 24: Dentro de la carpeta de parásitos dentro de libs

Dentro de la carpeta de **logs** hay un archivo con nombre **stdout.lpe.log**, este archivo contiene la información para dar a conocer al usuario si existe alguna falla en la verificación, poder mostrarle cuál es el error. En la carpeta que se crea con nombre **Other\_Files** están los archivos adicionales que generan a partir de las opciones adicionales que se colocan para dar más información detallada sobre esa opción que se coloca, esto se observa en la Figura [25](#)

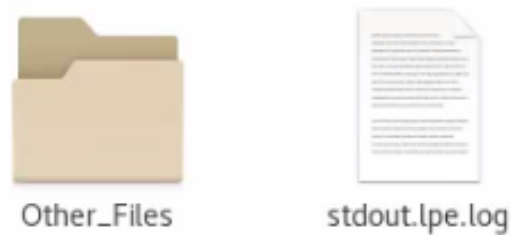


Figura 25: Dentro de la carpeta de parásitos dentro de logs

En la carpeta de **Salidas** se encuentra el archivo **chip\_IO.spf** que es el esperado de la verificación como se ve en la Figura [26](#), este contiene un listado detallado de los circuitos que se analizaron para verificar la propia extracción y que cumpla con los parámetros establecidos por la tecnología y la misma empresa de TSMC para validar la fabricación.

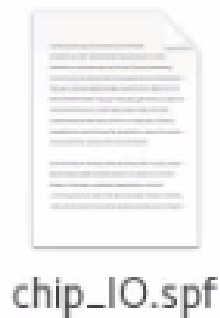


Figura 26: Dentro de la carpeta de parásitos dentro de salidas



---

## Automatización de la verificación de extracción de parásitos

---

En el escritorio de la misma computadora se cuenta con una carpeta especial para la automatización de extracción de parásitos, esto con el propósito de realizar pruebas sin ocasionar errores al progreso que se tienen en el servidor compartido y este se actualiza cuando se hacen cambios realmente importantes.

En la Figura 27 se pueden ver el script que se utiliza para la automatización, el archivo que contiene todos los comandos para llevar a cabo la verificación, la carpeta de las librerías que son necesarias para completar la verificación, la carpeta de los archivos de reportes y la carpeta de salidas donde se ubica el archivo final de la verificación.



Figura 27: Carpeta de extracción de parásitos en el escritorio

### 11.1 Script para la automatización de la verificación

Para automatizar esta etapa se hizo uso de un script simple que ejecute el comando para utilizar la herramienta de StarRC y completar de forma exitosa la verificación. Con este script se manda a llamar el archivo con nombre **CMD\_letona.cmd** el cual contiene todos

los comandos que se utilizan, tanto para utilizar librerías proporcionadas por la empresa de TSMC como los comandos para poder brindar un archivo de salida y de reporte de dicha verificación ejecutada anteriormente.

En el siguiente script se muestra como se remueven archivos con el objetivo de no tener duplicidades y tener archivos erróneos, se crean dos carpetas, una con nombre de **Logs** y la segunda con nombre **Salidas** porque todos los archivos que se generan son colocados donde se ejecuta el propio script, con lo que es necesario crear dichas carpetas para mover los archivos resultantes en su lugar respectivo. 43

---

```
1  #!/bin/bash
2
3  rm -rf Logs
4  rm -rf Salidas
5
6  StarXtract CMD_letona.cmd > ./stdout.lpe.log 2>&1
7
8  mkdir Salidas
9  mkdir Logs
10 cd Logs/
11  mkdir Other_Files
12 cd ../
13
14  mv chip_IO.spf Salidas
15
16  mv stdout.lpe.log Logs
17  mv chip_IO.star_sum Logs/Other_Files
18  mv PLACEMENT_chipIO Logs/Other_Files
19  mv .nodeIdDynamicRanges Logs/Other_Files
20  mv .tech_file.asc.lck Logs/Other_Files
21  mv .tech_file.lock Logs/Other_Files
22  mv star Logs/Other_Files
23  mv chip_IO.gpd Logs/Other_Files
24
25  echo "Ejecucion completa"
26
27  exit
```

---

Cuadro 43: Script para ejecutar la verificación de extracción de parásitos

El script anterior hace uso del archivo **CMD\_letona.cmd**, este contiene todos los comandos que se requieren para hacer uso de las librerías que nos proporciona la empresa de TSMC.

Este para tener un orden se separa por INPUTS, OUTPUTS y OPTIONS con lo que incluye los comandos generales para la correcta configuración inicial, así como los comandos

específicos que se desean utilizar al momento de realizar la extracción de parásitos.

Para el primer INPUT se hace uso de la base de datos de **MILKYWAY\_XTR** que contiene archivos base que lo que hacen es comparar con el circuito que se tiene la arquitectura y si esta misma tiene compuertas lógicas que también se encuentra en la base de datos se tiene un resultado ya hecho con lo que esto ayuda a facilitar a la herramienta la verificación. [28]

Para los siguientes tres inputs se utilizan archivos base que son proporcionados por la compañía de TSMC, el archivo de **STARRCXT.mapping** contiene información de la conductividad de los metales y policilicios de los transistores a utilizar.

El archivo **STARRCXT.runset\_rep** contiene la información del circuito que se está verificando y se compara con la base de datos de MILKYWAY.

El último archivo es el **cm018g\_1p6m\_4x1u\_mim5\_40k\_cbest.nxtgrd** es de base para la extracción de parásitos, ya que este contiene información que solo la propia compañía puede ver, puesto que esta no es visible. [28]

El último input es el archivo **netlist.icv** que se genera de la herramienta de icv en una de las verificaciones anteriores y que es necesaria, ya que contiene las blackboxes de todas las celdas del circuito que se está realizando la verificación. [28]

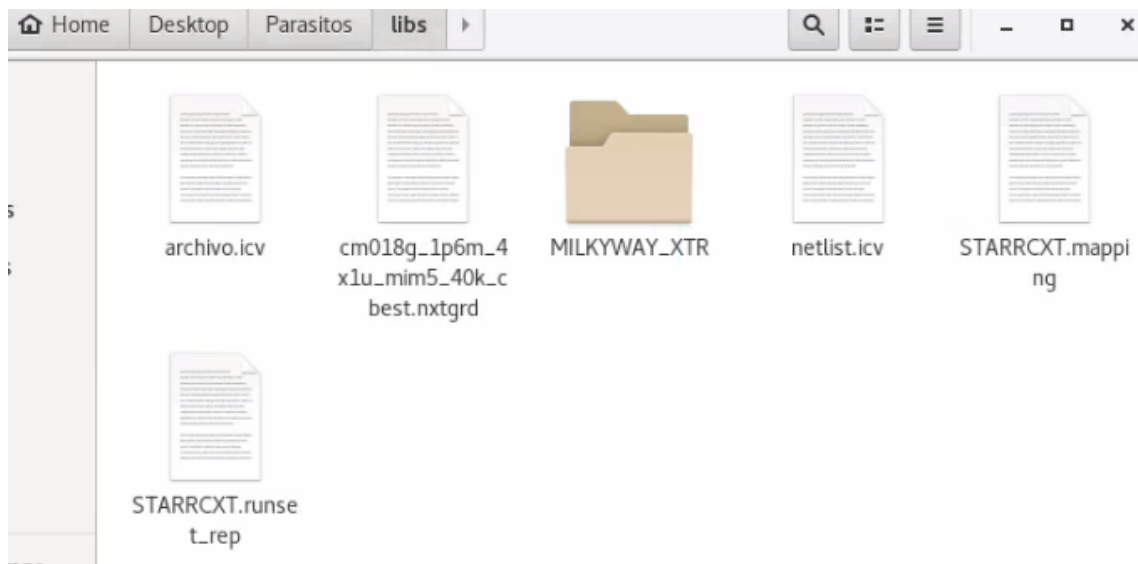


Figura 28: Carpeta de extracción de parásitos en el escritorio dentro de las librerías

Para el archivo de salida se tiene de nombre al **chip\_IO.spf** este contiene un listado de toda la arquitectura y como es que está estructurada todas las conexiones entre todos los transistores. [29]

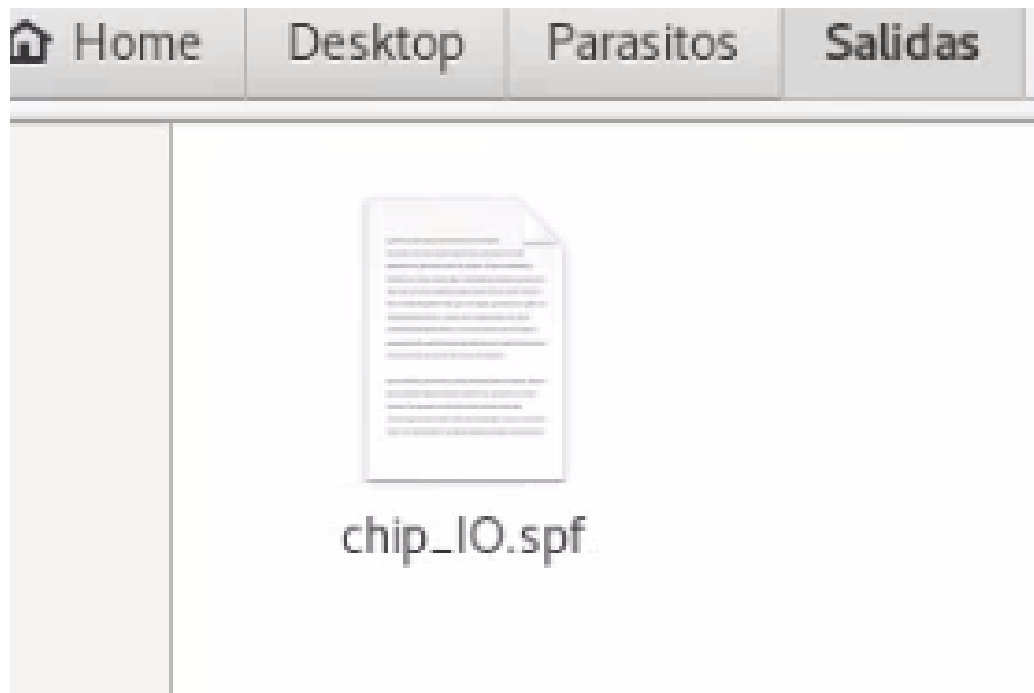


Figura 29: Archivo de salida de la extracción de parásitos

En el código que se presenta más adelante [44](#) se puede ver en la sección de **OUTPUT** que el primero nombre es el *NETLIST\_FORMAT* el cual nos permite tener varias opciones que se detallan en la siguiente listado, el siguiente es que el netlist se le puede definir la dirección y el nombre del archivo que se generara luego de realizar la extracción.

- **SPF:** Es el formato estándar y soporta únicamente `EXTRACTION: RC` con el comando `COUPLE_TO_GROUND: YES | NO`
- **SPEF:** Todos los nombres se encuentran mapeados internamente para reducir el tamaño del netlist.
- **OA:** Crea un archivo OpenAccess y solo se puede utilizar con una extracción a nivel transistor.
- **NETNAME:** Exclusivo para la extracción a nivel transistor, genera nodos internos con nombres como `netname:1`, `netname:2` y así sucesivamente hasta nombrar todos los nodos existentes.
- **STAR:** Exclusivo para la extracción a nivel transistor, emplea nomenclatura de sub nodos tipo SPICE.
- **PARAMETERIZED\_SPICE:** Es un netlist SPICE parametrizado que permite un análisis Monte Carlo.

Código completo utilizado para la extracción de parásitos.

```

1  **-----INPUT-----**
2  BLOCK: chip_IO
3  MILKYWAY_DATABASE: libs/MILKYWAY_XTR
4  TCAD_GRD_FILE: libs/cm018g_1p6m_4x1u_mim5_40k_cbest.nxtgrd
5  MAPPING_FILE: libs/STARRCXT.mapping
6  ICV_RUNSET_REPORT_FILE: libs/STARRCXT.runset_rep
7  SPICE_SUBCKT_FILE: libs/netlist.icv
8  **-----OUTPUT-----**
9  NETLIST_FORMAT: SPF
10 NETLIST_FILE: chip_IO.spf
11 COUPLING_REPORT_FILE: Salidas/ejercicio_cc.rep
12 **-----OPTIONS-----**
13 PLACEMENT_INFO_FILE: YES
14 PLACEMENT_INFO_FILE_NAME: PLACEMENT_chipIO
15 MILKYWAY_USE_CELL_PINS: NO
16 NDM_USE_DESIGN_PINS: NO
17 CASE_SENSITIVE: NO
18 HIERARCHICAL_SEPARATOR: /
19 COUPLE_TO_GROUND: YES
20 EXTRACTION: RC
21 REDUCTION: YES
22 DENSITY_BASED_THICKNESS: YES
23 *** For 0.13um and above process
24   EXTRACT_VIA_CAPS: NO
25 *** DataBase Processing
26 *****
27 *Original:
28 POWER_NETS: VDD VSS
29 SKIP_PCELLS : cfmom* cfmom_mx* cfmom_rf* crtmmom* crtmmom_rf* ind_std*
   ↪ ind_std_40k* ind_sym* ind_sym_40k* ind_sym_ct* ind_sym_ct_40k* jvar*
   ↪ lcesd1_rf* lcesd2_rf* lowcpad_rf* mimcap_rf* mimcap_rf_2p0* mos_var*
   ↪ mos_var33* moscap_rf* moscap_rf33* moscap_rf33_nw* moscap_rf_nw*
   ↪ ndio_hia_rf* ndio_sbd_mac* pdio_hia_rf* rfnmos2v* rfnmos2v_6t* rfnmos3v*
   ↪ rfnmos3v_6t* rfpmos2v* rfpmos2v_5t* rfpmos2v_nw* rfpmos2v_nw_5t*
   ↪ rfpmos3v* rfpmos3v_5t* rfpmos3v_nw* rfpmos3v_nw_5t* rphpoly_rf*
   ↪ rphripoly_rf* rplpoly_rf* sbd_rf* sbd_rf_nw* spiral_std_m2u_a_33k*
   ↪ spiral_std_m2u_x_33k* spiral_std_mu_a_33k* spiral_std_mu_x_20k*
   ↪ spiral_std_mu_x_33k* spiral_std_mu_x_40k* spiral_sym_ct_m2u_u_a_33k*
   ↪ spiral_sym_ct_m2u_u_x_33k* spiral_sym_ct_mu_x_20k*
   ↪ spiral_sym_ct_mu_x_33k* spiral_sym_ct_mu_x_40k*
   ↪ spiral_sym_ct_mu_x_a_33k* spiral_sym_m2u_u_33k* spiral_sym_mu_x_20k*
   ↪ spiral_sym_mu_x_33k* spiral_sym_mu_x_40k*
30

```

Cuadro 44: Comandos e inclusión de librerías para completar la verificación

Al finalizar la extracción de parásitos se crea una carpeta con nombre **Logs** donde dentro se coloca el archivo de reportes que se genera con nombre **x**, donde dentro se tienen todos los procesos que se realizaron para llevar a cabo la verificación y poder ver si realmente se tienen errores en la estructura. [30](#)

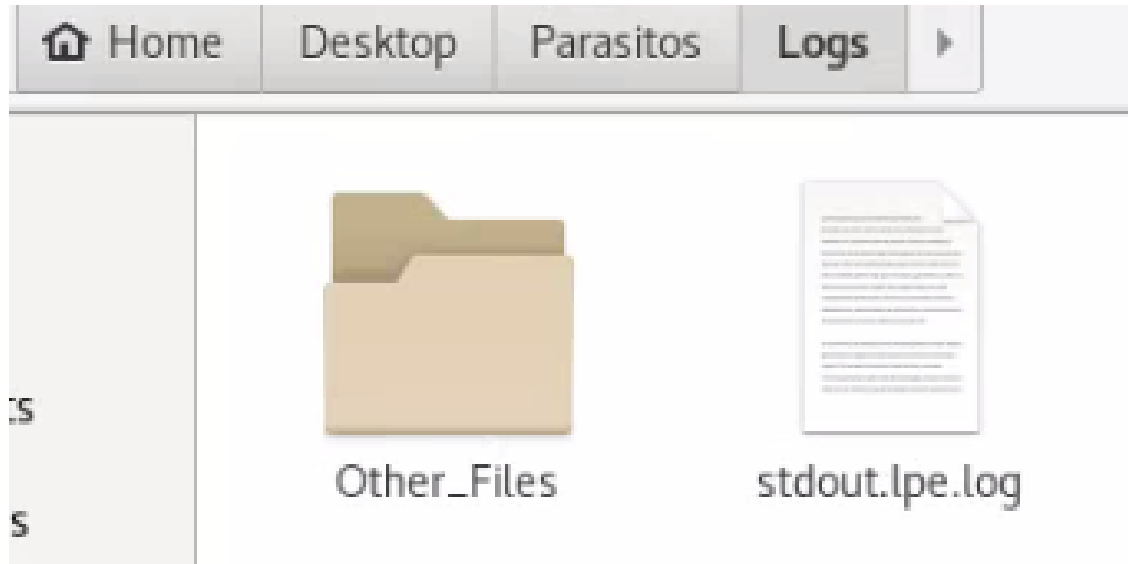


Figura 30: Archivo de reporte de la extracción de parásitos

Dentro de la carpeta **Other\_Files** se encuentran más archivos que contienen la información más detallada de las opciones que en el script de **CMD\_letona.cmd** se colocan, estas opciones pueden ayudar a obtener información sobre circuitos RC sea el caso que se coloque, así como opciones más detalladas y de la información que se desea obtener.



---

### Conclusiones

---

- Se logró automatizar el proceso de síntesis lógica, esto se realizó con éxito utilizando 3 scripts, cada uno realiza una tarea importante y en el orden correcto de ejecución no altera, no genera errores y mantiene un orden adecuado para una mayor facilidad de entrega de archivos.
- Utilizando los comandos necesarios en BASH se logra eliminar archivos y carpetas en la ubicación ya establecida, para luego tener un orden muy explícito se crean las carpetas con los nombres apropiados para identificar de manera rápida su ubicación
- Los archivos de salida de síntesis lógica se entregaron al grupo de pruebas físicas para corroborar que todo funcionase adecuadamente en un *FPGA Genesys Xilinx Virtex-5 LX50T*
- La comunicación entre los integrantes permitió la creación de un servidor compartido para entregar más eficientemente los archivos resultantes de cada etapa y no atrasar el diseño del nano Chip.
- Se tienen varios scripts unidos adecuadamente donde los archivos de salida de síntesis lógica se copian directamente a la carpeta de entradas de síntesis física dentro de la carpeta compartida y de igual forma se hace de síntesis física con verificaciones físicas, con lo que no existen problemas de flujo de diseño y concreta con los requerimientos para terminar el diseño del nano Chip.
- Se logró automatizar la verificación de extracción de parásitos, esto utilizando un script que elimina archivos de pruebas anteriores, al igual que ordena los archivos nuevos en carpetas determinadas para una buena organización y mayor facilidad de lectura de archivos.



---

### Recomendaciones

---

- Con el script que hace la lectura de archivos HDL siempre se debe colocar de primero el "Top Module" para que funcione conjunto con el script de Python de "Lectura Universal"
- Las carpetas de librerías se pueden mejorar colocándolas en el directorio de origen para poder ejecutar únicamente el script de síntesis lógica como extracción de parásitos en cualquier parte de la computadora y garantizar su funcionamiento colocando las salidas donde se ejecute el script.
- En la colocación de los "Pads" mejorar la interfaz para tener la opción de elegir entre las diferentes configuraciones que se pueden usar.
- Apoyarse con los manuales que se pueden obtener de SolvNet para cambiar o mejorar el proceso de Síntesis lógica.



---

## Bibliografía

---

- [1] Solvnet.com, «Synopsys Corporate Backgrounder Spring 2018,» Synopsys, 2008.
- [2] J. A. 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.,» en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2014.
- [3] L. Abadia, «Posicionamiento e interconexión entre componentes de un circuito sintetizado para el flujo de diseño de un circuito a escala nanométrica utilizando la herramienta de IC Compiler,» en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2021.
- [4] M. Flores, «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,» en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2021.
- [5] M. Sibrian, «Verificación de reglas de diseño (DRC) para el desarrollo de un flujo funcional de un circuito integrado con tecnología nanométrica,» en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2021.
- [6] E. O. Torres Garza, «Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: Ejecución y simulación para la etapa de síntesis lógica,» en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2014.
- [7] A. Altuna, «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.,» en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2021.

- [8] J. Ayala, «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,» en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2021.
- [9] J. Shin, «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,» en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2021.
- [10] Synopsys.Inc, «VCS User Guide Q-2020.03,» Solvnet, 2020.
- [11] F. R. Sala, «Programacion en SHELL (Bash) User Guide Q-2022.03,» Ubuntu, 2022.
- [12] K. S. Cardona, «Mejoramiento del proceso de síntesis lógica llevada a cabo para la elaboración de un circuito integrado a escala nanométrica,» en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2021.
- [13] T. S. M. Company, «TPD018NV TSMC 0.18um Standard I/O Library,» en *Databook*, Taiwan Semiconductor Manufacturing Company, 2009.
- [14] S. Synopsys, «Design Vision Datasheet,» Solvnet, 2019.
- [15] Synopsys, «Design Vision User Guide R-2020.09,» Solvnet, 2020.

## 15.1 Códigos útiles y de resultados

En el siguiente enlace de drive se pueden encontrar todos los códigos completos que se utilizaron. [Códigos](#).

