Automatización de las verificaciones físicas de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC

Stefan Alexander Schwendener Farrington

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Automatización de las verificaciones físicas de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC

Trabajo de graduación presentado por Stefan Alexander Schwendener Farrington para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

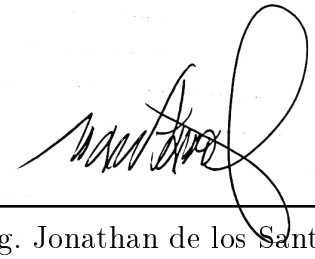
Automatización de las verificaciones físicas de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC

Trabajo de graduación presentado por Stefan Alexander Schwendener Farrington para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

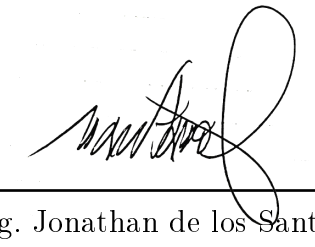
Vo.Bo.:



(f) _____

Ing. Jonathan de los Santos

Tribunal Examinador:

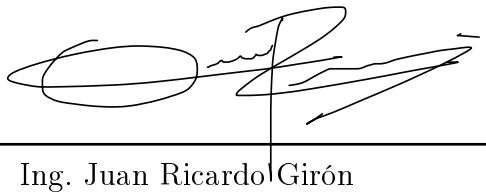


(f) _____

Ing. Jonathan de los Santos

(f) _____

MSc. Carlos Esquit



(f) _____

Ing. Juan Ricardo Girón

Fecha de aprobación: Guatemala, 8 de Diciembre de 2022.

Prefacio

Este trabajo de graduación no hubiera sido posible sin el apoyo de mis amigos y familia. Quisiera agradecerle a mi familia por el apoyo que me han dado a lo largo de toda mi carrera y por la oportunidad de estudio que me han otorgado. Quiero agradecerle al Ingeniero Jonathan de los Santos por todo el apoyo que me ha brindado con respecto al uso de comandos en Linux y el uso de las aplicaciones de Synopsys. Además, también se le agradece al Msc. Carlos Esquit por introducirme al área de la nanoelectrónica y por siempre apoyarme con su conocimiento del tema. Por último, quiero darle las gracias a mis compañeros de trabajo del proyecto del *chip* por toda la ayuda que me dieron a lo largo de este proyecto.

Índice

Lista de figuras

Lista de cuadros

Lista de tablas

Resumen

Este trabajo trata de la automatización de asimismo la fase de diseño de *Verificaciones Físicas* de un circuito integrado con tecnología de 180 nm, abarca las siguientes *Verificaciones Físicas*: a) reglas de diseño o *Design Rules Check* (DRC), b) reglas eléctricas o *Electrical Rules Check* (ERC), c) *Layout Vs. Schematic* (LVS) y d) Verificación de Antena. Cabe señalar que su intención es la de facilitar el proceso de diseño de un circuito integrado por medio de scripts que realicen de forma automática procesos largos, repetitivos y/o tediosos.

Para realizar las verificaciones de este trabajo se utilizaron aplicaciones suministradas por la empresa *Synopsys*. Por medio de la aplicación de *IC Validator* se permitió realizar todas las *Verificaciones Físicas* sin necesidad de una interfaz gráfica de usuario y de la interacción del mismo. Además de lo anterior también se trabajó con la herramienta de *Synopsys NetTran* la cual permitió realizar las conversiones necesarias de formatos de archivos.

Dado a que el entorno de las aplicaciones de *Synopsys* ocurre en la terminal, se optó trabajar con scripts del tipo *Bourne Again Shell* (BASH). Se decidió utilizar este tipo de script ya que no requiere de ninguna librería adicional y es bastante robusto debido a que su funcionamiento no cambia es independiente de la máquina en que se ejecuta. Cabe mencionar que algunos *Sistemas Operativos Linux* tienen comandos un poco distintos por lo que podría haber una necesidad de editar los comandos si cambiara de *Sistema Operativo*.

Abstract

This piece covers the automatization of the Physical Verifications phase in the design of an integrated circuit with 180 nm technology. This work will encompass the distinct physical verifications of: a) *Design Rules Check* (DRC), b) *Electrical Rules Check* (ERC), c) *Layout Vs. Schematic* (LVS), and d) antenna verification. This work focuses on facilitating the process of *Integrated Circuit* (IC) design, using scripts that automate large and complex processes.

The tools and applications used to complete the verifications in this work were all provided by the company *Synopsys*. The application *IC Validator* was used to complete all the Physical Verifications without the need for a graphical interface or any sort of user input. Another tool in the automation process was the *NetTran* tool provided by *Synopsys* which facilitated the conversion of the required file formats.

Given that most *Synopsys* applications require to be executed through the Linux terminal, it was decided to use *Bourne Again Shell* (BASH) scripts. This type of script was chosen since it does not require the use of additional libraries, thus ensuring the compatibility between systems wouldn't be a problem. It's worth mentioning that in some *Linux Operating Systems* commands might vary slightly, which would require the editing of the scripts to satisfy the needs of the new OS.

CAPÍTULO 1

Introducción

En la actualidad los dispositivos electrónicos forman una gran parte de nuestra vida cotidiana. La capacidad que tienen estos dispositivos para realizar varias tareas se ve impactada directamente por la tecnología que tienen sus circuitos integrados debido a que el espacio suele ser un factor limitante. El campo de la nanoelectrónica surgió a base de necesidad de reducir el tamaño de los circuitos electrónicos sin comprometer su funcionamiento. Los circuitos integrados producidos en la actualidad son capaces de realizar múltiples tareas por lo que se utilizan en una gran variedad de campos.

Al pasar los años, los circuitos se han hecho más pequeños. La ley de Moore describe que la cantidad de transistores dentro de un circuito integrado denso se ha duplicado cada dos años[1]. Debido a las limitaciones de las tecnologías de la actualidad, esto ya no es del todo cierto debido a que mediante los circuitos se hacen más pequeños, las leyes de la física que describen su funcionamiento se vuelven más complejas por lo que son más susceptibles a perturbaciones externas. Debido a lo anterior, durante el periodo de diseño de un Circuito Integrado es necesario realizar algunas verificaciones para asegurar el funcionamiento adecuado del mismo.

En los siguientes capítulos se presentará información acerca de las *Verificaciones Físicas* y su proceso de automatización utilizando comandos del entorno Linux. Todo esto se realizó con herramientas de *Synopsys* como *IC Validator* y *NetTran*. Adicionalmente se utilizaron las librerías de diseño de 180 nm y 6 capas de metal de TSMC. Luego también se describe como el trabajo podría replicarse en futuras generaciones del grupo de trabajo de *El Gran Jaguar*.

CAPÍTULO 2

Antecedentes

A inicios del año 2009 la Facultad de Ingeniería de la Universidad del Valle de Guatemala comienza a expandir su aprendizaje en el área de tecnología nanométrica. Cuando el ingeniero Carlos Esquit obtiene el puesto de Director de carrera de Ingeniería Electrónica él modifica la carrera para expandir los conocimientos de los estudiantes en distintas áreas, incluyendo el área de tecnología nanométrica. En el año 2013 se incorpora el curso de Introducción al Diseño de Sistemas VLSI en el pénsum de los estudiantes.

En el año 2014 Jonathan de los Santos presentó el primer trabajo de graduación cuyo propósito fue realizar el diseño de un sumador/restador de 32 bits con tecnología CMOS de 28 nm utilizando aplicaciones de Synopsys, se puede consultar este trabajo en [2]. El presente trabajo de graduación es la base del proyecto que se está llevando a cabo en la actualidad debido a que la instalación de los programas y otros requisitos necesarios para realizar el diseño de un circuito integrado fueron realizados en este trabajo de graduación. Con este trabajo se introdujeron las aplicaciones de Synopsys que se están empleando en el proyecto actual.

En el año 2019 se le presenta una oportunidad a la Universidad del Valle de Guatemala por medio de la compañía Taiwan Semiconductor Manufacturing Company (TSMC) para diseñar y fabricar un circuito integrado. En el trabajo [3] se definió el flujo de diseño para la fabricación de un circuito integrado capaz de desplegar un mensaje con caracteres ASCII utilizando máquinas de estados finitos y aplicaciones de Synopsys. Los diagramas de flujo propuestos en este trabajo, de las fases de diseño de front-end y back-end se muestran en las figuras 1 y 2. También se presentó la primera fase del flujo de diseño: implementación de circuitos a nivel de netlist a partir de un lenguaje descriptivo a base de hardware consultado de [4]. Estos trabajos representan la propuesta inicial del proyecto que se está realizando actualmente.

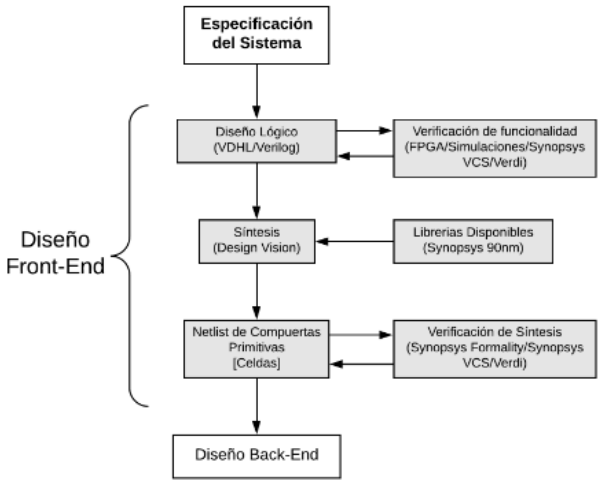


Figura 1: Diagrama de Flujo Front-End

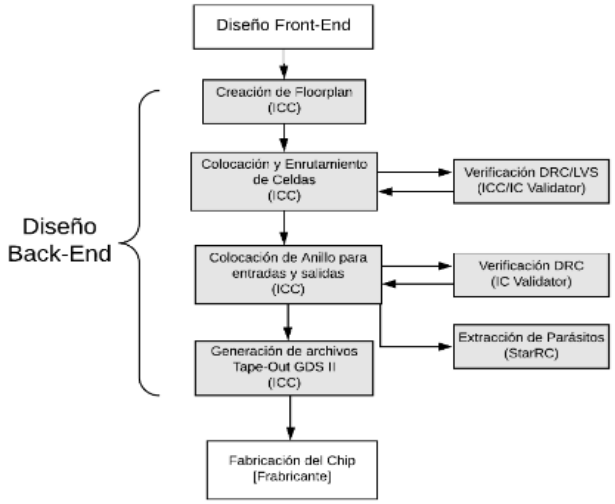


Figura 2: Diagrama de Flujo Back-End

Más tarde, en el año 2020 se continuó con el trabajo propuesto en el año anterior con un equipo de estudiantes mucho más grande, pero es importante mencionar que en este año tuvo lugar el inicio de la pandemia causada por el COVID-19 lo cual ha tenido un impacto en el trabajo hasta la actualidad. En uno de estos trabajos de graduación [5] se definió el flujo en la herramienta VCS para simular los documentos HDL necesarios en la fabricación del circuito integrado. Asimismo, en este año también se documentaron y realizaron algunas de las verificaciones físicas como la verificación de reglas de diseño (DRC) de una compuerta NOT utilizando la aplicación de Custom Compiler [6], comparación de Silicio y Esquemático (LVS) [7],verificación de reglas eléctricas (ERC) y pruebas de antena [8]. Por último, se realizó la primera extracción de parásitos y algunas simulaciones en la aplicación de Synopsys HSPICE [9]. Estos trabajos han sido importantes en la línea de investigación debido a que documentan el proceso a seguir para realizar las verificaciones

físicas de un circuito integrado de nanoescala.

A finales del año 2021 otro grupo de estudiantes realizó correcciones y completó otras fases de diseño. El trabajo se separó en cuatro grupos, encargados de las diferentes fases de diseño: síntesis lógica, síntesis física, *Verificaciones Físicas* y por último, se empezó con la fase de automatización. Se realizó una mejora al proceso de síntesis lógica [10] al igual que a su ejecución y simulación [11]. En este año también se realizó la síntesis física junto con las verificaciones de reglas de diseño (DRC) [12] [13], verificaciones de reglas eléctricas (ERC) [14], comparación de Silicio y Esquemático (LVS) [15] y algunas correcciones a los errores observados en ambas verificaciones. También cabe mencionar que se realizó el posicionamiento e interconexión entre componentes del circuito integrado con la herramienta de IC Compiler [16]. Algo que es importante de mencionar para este trabajo de graduación, es que aún se tienen errores de densidad al ejecutar el DRC y además se tienen errores al ejecutar LVS sin entradas y salidas.

CAPÍTULO 3

Justificación

En este trabajo de graduación se hace énfasis la automatización de las fases de desarrollo de diseño de un circuito integrado. El proceso de fabricación se realiza con herramientas de Synopsys y esto es un proceso largo y tedioso, por esto es necesario encontrar una forma de automatizar el proceso para disminuir la carga de trabajo que cada integrante del equipo de desarrollo tiene. La automatización se debe realizar para que cada miembro pueda tener un mejor enfoque en el objetivo principal de su fase de diseño sin desperdiciar tiempo ingresando comandos en la terminal.

Una de las fases que se busca automatizar es la fase de verificaciones físicas, esta es una fase bastante larga de realizar sin la automatización. Anteriormente, se hacía necesario modificar archivos de HSPICE y Verilog a mano para poder extraer un archivo de formato *ICV* el que es de suma importancia para realizar la verificación de Electrical Rule Check (ERC). Asimismo, para realizar la verificación ERC también es necesario añadir cajas negras manualmente al documento que se carga a IC Validator; es por esto que la fase de *Verificaciones Físicas* debe automatizarse.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Simplificar el proceso de diseño de un circuito integrado por medio del uso de scripts tipo BASH o Shell e integrar la automatización de las *Verificaciones Físicas* con otras fases de diseño.

4.2. Objetivos específicos

- Permitir que el usuario copie los scripts y archivos necesarios en cualquier directorio sin afectar el funcionamiento del mismo.
- Realizar un script flexible que sea completamente independiente de lo que el usuario le ingrese.
- Simplificar lo más posible la fase de diseño de *Verificaciones Físicas*.
- Utilizar las aplicaciones de Synopsys para encontrar comandos que permitan la automatización de las fases de diseño del circuito.
- Incorporar los scripts de cada fase para simplificar aún más el proceso de diseño.

CAPÍTULO 5

Alcance

El alcance de este trabajo de graduación es simplificar el proceso de las *Verificaciones Físicas* de manera que el usuario solo introduzca las salidas de otras fases para obtener los resultados de cada verificación. Para que esto sea posible, se utilizaron las aplicaciones de *IC Validator* y *NetTran* de la empresa *Synopsys*. Se utilizó la aplicación de *IC Validator* para realizar las *Verificaciones Físicas* desde la terminal de Linux y se utilizó *NetTran* para traducir formatos de archivos obtenidos en otras fases de diseño.

Este trabajo se limita a realizar la automatización y algunas correcciones a la fase de diseño de *Verificaciones Físicas*, por lo que es importante tomar en cuenta los siguientes trabajos de graduación de años anteriores. Del proceso de Síntesis Física realizado por Antonio Altuna, Julio Shin y José Ayala se obtuvieron las verificaciones de reglas eléctricas (ERC), reglas de diseño (DRC) y prueba de antena. La etapa de Verificación Física de Diseño en Silicio vs. Esquemático (LVS) se obtuvo del trabajo realizado por Juan Ricardo Girón y Marvin Flores.

En este trabajo se explica cómo se realiza cada verificación física junto con la parte del script, de formato *Bourne Again Shell* (BASH), que le corresponde. También se explicaron las diferencias entre cada *runset* de las distintas pruebas realizadas. Por último, se incluye una sección de pruebas realizadas al script para asegurar su robustez.

CAPÍTULO 6

Marco teórico

El proceso de automatización del flujo de diseño de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC requiere de algunos conocimientos adicionales que se profundizan en esta sección. En esta sección se hablará brevemente sobre el flujo de diseño de las *Verificaciones Físicas*, un poco acerca de scripts tipo BASH del sistema operativo linux y también se mencionaran los comandos utilizados en el proceso de automatización.

6.1. Flujo de diseño

6.1.1. Síntesis lógica

La síntesis lógica es el proceso de producción automática de componentes lógicos. Este es el proceso por medio del cual un circuito abstracto a nivel de diseño de nivel "Register Transfer Level" (RTL) se lleva a un nivel de compuertas lógicas o nivel de transistores. Es decir, este es el proceso por el cual un diseño abstracto pasa a utilizar componentes reales que pueden fabricarse. [17]

6.1.2. Síntesis física

El proceso de síntesis física utiliza los resultados de la síntesis lógica para proponer un esquemático a nivel físico. En esta etapa se selecciona el tamaño y la organización que los componentes tendrán en el plano físico; además, también se toma en cuenta el ruteo necesario para que el circuito se comporte como se había propuesto.[18]

6.1.3. *Verificaciones Físicas*

Las *Verificaciones Físicas* tienen como objetivo verificar que el diseño propuesto cumpla con ciertas reglas de diseño sin ningún problema. Estas verificaciones incluyen: Design Rule Check (DRC), Electric Rule Check (ERC), comparación entre Silicio y Esquemático (LVS), y verificación de Antena. El diagrama de flujo del programa de automatización de las *Verificaciones Físicas* se muestra en la Figura 3.

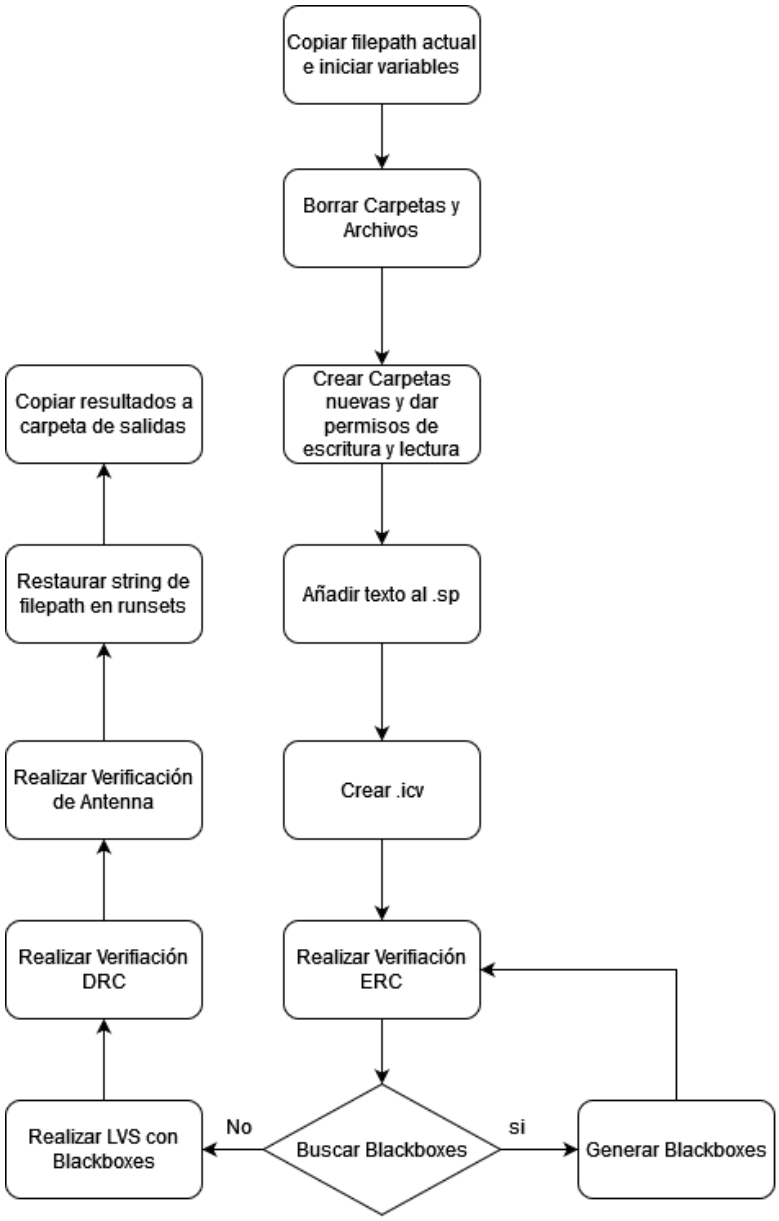


Figura 3: Diagrama de flujo del programa de automatización de *Verificaciones Físicas*

6.2. Scripting en BASH

Los scripts realizados en este trabajo de graduación se realizaron en un formato tipo Bourne-Again Shell (conocido como BASH). BASH es un intérprete de lenguaje de la linea de comandos del sistema operativo Linux. BASH nos permite escribir scripts que controlan a la terminal de una computadora utilizando Linux o MacOS directamente sin necesidad de ingresar los comandos manualmente en la terminal. Esto se utilizará para automatizar una gran parte de las fases de diseño del circuito integrado ya que la mayoría de los programas de Synopsys se pueden abrir utilizando comandos de la terminal sin necesidad de una interfaz gráfica.[19]

6.2.1. Redirecciones

La manera en que se ingresan datos en una computadora usualmente se realiza a través de stdin (usualmente por el teclado) y la salida resultante va a stdout (usualmente un Shell). A estos caminos se les conoce como *streams*, es posible alterar las ubicaciones de entrada y salida, lo cual hace que la máquina obtenga datos de un lugar que no sea stdin o que mande los resultados a un lugar que no sea stdout. A esto se le conoce como redirección. En el Cuadro 6.1 se muestran unos ejemplos de como se realizan redirecciones, se utilizara el comando *date* para realizar algunas de las demostraciones. Como puede observarse a continuación el comando *date* despliega la fecha actual en la terminal.

Cuadro 6.1: Salida de comando 'date

```
[admin@localhost ~]$ date
Tue Dec 29 04:07:37 PM MST 2022
[admin@localhost ~]$
```

El operador > se le conoce como redirector de salidas y su función es dirigir algún dato para escritura en una salida. Como se muestra en el Cuadro 6.2, se añade la salida del comando *date* un archivo de texto llamado *especificaciones.txt*. Es importante mencionar que este comando elimina todo lo que se encontraba dentro del archivo al que se está redireccionando.

Cuadro 6.2: Ejemplo de redirector de salidas

```
[admin@localhost ~]$ date > especificaciones.txt
[admin@localhost ~]$ cat especificaciones.txt
Tue Dec 29 04:08:44 PM MST 2022
[admin@localhost ~]$
```

Al operador » se le conoce en inglés como *append* y su función es redireccionar los datos hacia un archivo sin eliminar lo que ya se encontraba dentro de ese archivo. En el

Cuadro 6.3 se muestra un ejemplo en el que se añade una fecha inicial a *especificaciones.txt* con el comando > y luego se añade una segunda linea con el comando ». Los resultados se despliegan en la terminal utilizando el comando *cat*.

Cuadro 6.3: Ejemplo de operador 'append de salidas

```
[admin@localhost ~]$ date > especificaciones.txt
[admin@localhost ~]$ date >> especificaciones.txt
[admin@localhost ~]$ cat especificaciones.txt
Tue Dec 29 04:00:00 PM MST 2022
Tue Dec 29 04:08:44 PM MST 2022
[admin@localhost ~]$
```

Al operador < se le conoce como redirector de entrada y lo que hace es redirigir datos de un stream de una fuente determinada para ser ingresados a un comando. En el siguiente ejemplo, que se muestra una lista en el Cuadro 6.4, que luego es ingresada al comando sort y mostrado en el Cuadro 6.5.

Cuadro 6.4: Texto guardado en *milista.txt*

```
[admin@localhost ~]$ cat milista.txt
gato
perro
caballo
vaca
[admin@localhost ~]$
```

Los resultados de ingresar el archivo mostrado en el 6.4 son los siguientes:

Cuadro 6.5: Ejemplo de redirector de entrada

```
[admin@localhost ~]$ sort < milista.txt
caballo
gato
perro
vaca
[admin@localhost ~]$
```

El operador | es conocido en inglés como *pipe* y permite ejecutar varios comandos de manera consecutiva. En el Cuadro 6.6 se muestra un ejemplo en el que se utiliza el comando *ls* para listar los contenidos de una carpeta y luego el comando *less* para separar los resultados en distintas páginas.

Cuadro 6.6: Ejemplo de operador 'pipe'

```
[admin@localhost ~]$ ls /etc | less
```

El cual produce el siguiente resultado en la terminal. Los dos puntos al final de la lista indican que hay más información abajo.

Cuadro 6.7: Resultado del ejemplo del operador 'pipe'

```
abrt
adjtime
aliases
aliases.db
alsa
alternatives
anacrontab
asound.conf
at.deny
audisp
audit
autofs.conf
autofs_ldap_auth.conf
auto.master
auto.master.d
auto.misc
auto.net
auto.smb
avahi
bash_completion.d
bashrc
binfmt.d
bluetooth
:
```

El operador *2>* se utiliza para redirigir errores de operación que han sido clasificados como errores de file descriptor 2.

Cuadro 6.8: Ejemplo del operador '2>

```
[admin@localhost ~]$ png 2> /dev/null
[admin@localhost ~]$
```

[20]

6.2.2. Declaración de variables

Las variables son símbolos que representan a un string o un valor numérico. Para crear una variable es importante señalar que debe crearse un nombre para la variable el cual no puede empezar con un número o contener espacios. Un ejemplo de como crear una variable en un script se muestra en el Cuadro 6.9; para observar la definición de la variable se utiliza el comando *echo* junto con un símbolo de dollar $.

Cuadro 6.9: Ejemplo de declaración de variables

```
[admin@localhost ~]$ yo=stefan
[admin@localhost ~]$ echo $yo
stefan
```

También existen variables de entorno, estas definen y guardan las propiedades del entorno cuando el sistema se inicia, asimismo las variables de entorno guardan información tal como el nombre del usuario, la localización de ejecutables, etc. Cabe señalar que estas variables se pueden observar con el comando *env* como se muestra en el Cuadro 6.10.

Cuadro 6.10: Ejemplo de variables de entorno

```
[admin@localhost ~]$ env | less
GNOME_SHELL_SESSION_MODE=ubuntu
COLORTERM=truecolor
...
```

[21]

6.2.3. Comando de terminal 'cat'

El comando *cat* es uno de los comandos más útiles en Linux, su nombre proviene de concatenar. Este comando permite crear, fusionar, imprimir archivos de texto desde la terminal de Linux. La sintaxis se muestra en el Cuadro 6.11.

Cuadro 6.11: Sintaxis del comando 'cat

```
[admin@localhost ~]$ cat [opcion(es)] [ARCHIVO]
```

[22]

6.2.4. Comando de terminal 'echo'

El comando *Echo* es uno de los comandos más utilizados y conocidos en el ambiente de Linux. Este comando es utilizado para desplegar en la terminal algún mensaje que se indique en un script o determinado por el usuario en la terminal. Si no se le selecciona ninguna opción adicional, el comando desplegará solamente el texto que esta a la par del comando. Una de las opciones que se utilizaron en el script de automatización fue la opción de activar interpretación de barra invertida que es la opción 'interpret'$-e$ [23]. La sintaxis del comando se muestra en el Cuadro 6.12.

Cuadro 6.12: Sintaxis del comando 'echo'

```
[admin@localhost ~]$ echo [opcion(es)] [string(s)]
```

Comando de terminal 'cd'

El comando *cd* es uno de los comandos más utilizados en la terminal de Linux debido a que este permite al usuario navegar en distintas direcciones dentro del ambiente Linux. Este comando es conocido por sus siglas en inglés 'Change Directory' o en español 'Cambiar de Directorio'. Este comando funciona con rutas absolutas y rutas relativas. Una ruta absoluta es aquella que contiene toda la ruta hacia el directorio o el archivo mientras que una ruta relativa funciona en base al directorio en el que se está trabajando en el momento. [24] La sintaxis del comando *cd* se muestra en el Cuadro 6.13.

Cuadro 6.13: Sintaxis del comando 'cd'

```
[admin@localhost ~]$ cd [direccion]
```

6.2.5. Comando de terminal 'rm'

El comando *rm* es un comando utilizado para remover archivos o carpetas en sistemas operativos Linux. Sus siglas significan 'remove' en inglés o 'remover' en español. Para poder remover archivos no es necesario ingresar ninguna opción adicional, por lo tanto sólo debe ingresarse el nombre del archivo, si se está utilizando una dirección relativa, o la dirección absoluta junto con el nombre del archivo. En este trabajo de graduación, se utilizó el comando *rm* para remover archivos al igual que carpetas. Para remover una carpeta es necesario utilizar la opción 'recursivo' $-r$ para remover directorios y sus contenidos de manera recursiva. Si existe una carpeta o un archivo que ya no existe o tiene algún argumento faltante, se utiliza la opción de 'force' $-f$ la cual hace que el comando ignore archivos o argumentos que estén vacíos. [25] La sintaxis del comando se muestra en el Cuadro 6.14.

Cuadro 6.14: Sintaxis del comando 'rm'

```
[admin@localhost ~]$ rm [opcion(es)][direccion de archivo o carpeta]
```

6.2.6. Comando de terminal 'mkdir'

El comando de *mkdir* permite al usuario crear carpetas dentro del directorio en que se está trabajando. Sus siglas en inglés significan 'make directory', en español 'crear directorio'. Con el comando también es posible configurar permisos y crear múltiples carpetas al mismo tiempo.[26] La sintaxis que se debe utilizar se muestra en el Cuadro 6.15.

Cuadro 6.15: Sintaxis del comando 'mkdir'

```
[admin@localhost ~]$ mkdir [opcion(es)][nombre del directorio]
```

6.2.7. Comando de terminal 'if & else'

Al realizar un script en un lenguaje BASH es posible utilizar condicionales por medio de los comandos de *if & else*. Estos comandos nos permiten verificar si una condición se cumple y realizar una operación si fuese necesario.

- if: representa la condición que se quiere verificar
- then: si la condición anterior se cumple, entonces se ejecuta un comando específico.
- else: Si la condición anterior es falsa, entonces ejecutar otro comando.
- fi: finaliza la declaración

[27]

La sintaxis a utilizar dentro de un script se muestra en el Cuadro 6.16.

Cuadro 6.16: Sintaxis del comando 'if & else'

```
#!/bin/bash
if [[condicion]]
then
    <comando a ejecutar>
else
    <comando a ejecutar>
fi
```

6.2.8. Comando bucle 'for'

El comando bucle *for* es un comando utilizado para automatizar un código que requiere que un comando sea ejecutado varias veces. [28] En este trabajo de graduación el bucle 'for' se utiliza para automatizar la parte de generación de blackboxes nuevos. La estructura principal de un bucle *for* se muestra en el Cuadro 6.17.

Cuadro 6.17: Estrucutra de un bucle 'for'

```
#!/bin/bash
for <variable> in <lista de objetos>;
    do <algun comando> <variable>;
done;
```

6.2.9. Comando de terminal 'Sed'

El comando *sed* nos permite realizar operaciones de búsqueda, reemplazo, inserción y eliminación dentro de un documento de texto. Su nombre es un acrónimo en inglés que significa 'Stream Editor'.[29] En este trabajo de graduación se utiliza la función de sed para encontrar una palabra clave y reemplazarla con la dirección absoluta de la carpeta de trabajo. Otra función de sed que se utiliza en este trabajo de graduación es la opción de búsqueda e inserción debido a que se tiene que insertar los nuevos blackboxes dentro del runset que ejecuta el LVS o ERC. La sintaxis del comando *sed* se muestra en el Cuadro 6.18.

Cuadro 6.18: Sintaxis del comando 'sed'

```
[admin@localhost ~]$ sed [opcion(es)] [script] [ingreso-de-archivo]
```

CAPÍTULO 7

Estructura de carpetas

Para mantener el trabajo de la manera más organizada, el script generado elimina y crea las carpetas necesarias junto con los archivos utilizados en la ejecución. Se utilizó el comando *rm -rf* para eliminar todas las carpetas y archivos de corridas anteriores. Luego utilizando el comando *mkdir* se crean las carpetas nuevamente y se les añaden permisos de escritura y lectura utilzando el comando *chmod*.

Cuadro 7.1: Parte del script de estructura de carpetas

```
20 ##############################################################################
21 # Eliminacion de remanentes
22 ##############################################################################
23 rm -rf ICV
24 rm -rf ERC
25 rm -rf LVS
26 rm -rf DRC
27 rm -rf Antenna
28 rm -rf SALIDAS
29 rm $filepath/archivos_importantes/nuevo_LVS.4a
30 rm $filepath/archivos_importantes/nuevo_ERC.4a
31 rm -f archivo.icv
32 echo -e "\n\n\nRemanentes eliminados\n\n\n" >> $filepath/output.txt
33 echo -e "\n\n\nRemanentes eliminados\n\n\n"
34 ##############################################################################
35 # Creacion de Folders y Archivos
36 ##############################################################################
37 mkdir ICV
38 mkdir ERC
39 mkdir LVS
```

```
mkdir DRC
mkdir Antenna
mkdir SALIDAS
chmod gou+wr ICV
chmod gou+wr ERC
chmod gou+wr LVS
chmod gou+wr DRC
chmod gou+wr Antenna
chmod gou+wr SALIDAS
touch output.txt
echo -e "\n\n\nFolders Creados\n\n\n" >> $filepath/output.txt
echo -e "\n\n\nFolders Creados\n\n\n"
```

En la Figura 4 se muestra la estructura de las carpetas generadas por el script de automatización de las verifiaciones físicas.

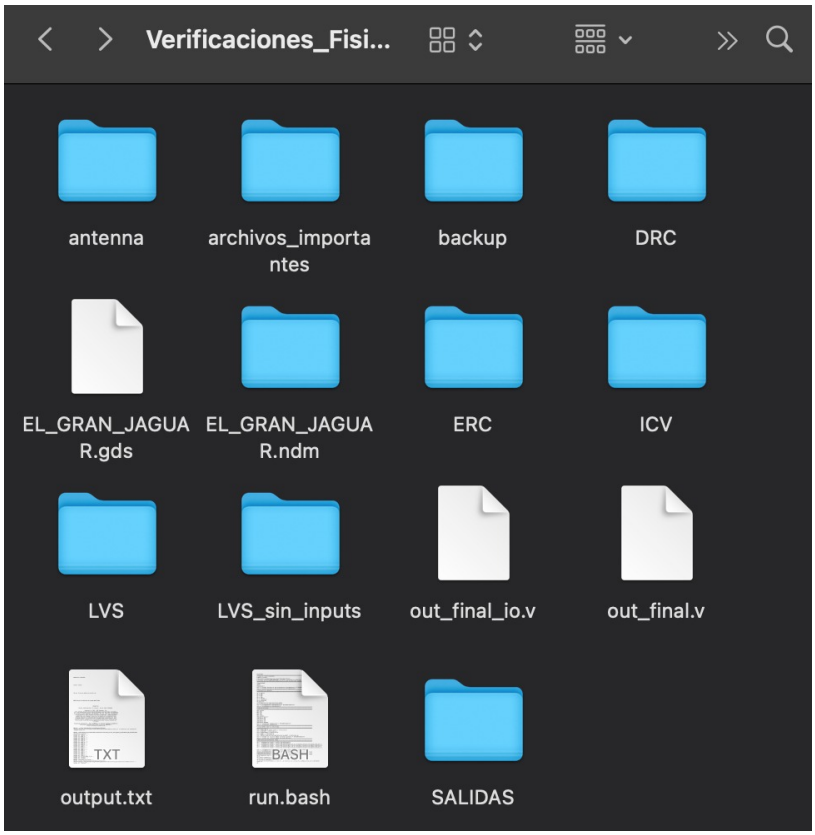


Figura 4: Estructura de carpetas de automatización

En la Figura 4 se pueden observar los resultados de ejecutar el script anterior. También se genera un archivo llamado output.txt el cual se encuentra en la carpeta de salidas. En este archivo se guardan todas las salidas de la terminal de Linux para poder realizar pruebas con facilidad en el caso de que se tuviese algún problema.

CAPÍTULO 8

Proceso de *Design Rules Check* (DRC)

En la verificación física del DRC se utiliza un *runset* proporcionado por **IMEC** para asegurar que todas las reglas de diseño cumplan con los estandares de fabricación de TSMC. El runset se encuentra en la carpeta de *archivos_importantes* y por facilidad ha sido renombrado como `DRC.215a_pre041518`, este originalmente se llamaba `ICVLM18_LM16_-LM152_6M.215a_pre041518`. Este *runset* es para uso específico de tecnologías de: $0.18\mu m$, $0.16\mu m$ y $0.152\mu m$. El *script* de esta fase nos permite verificar el *layout* producido por la síntesis física.

En el Cuadro 8.1 se muestra un script tipo BASH que automatiza el proceso de DRC. Como se puede observar el comando que se utiliza es de IC Validator y se tienen dos opciones disponibles dependiendo de lo que el usuario desee realizar. En la opción que se encuentra descomentada, se realiza la verificación de DRC utilizando el comando de IC Validator utilizado durante la síntesis física utilizando ICC2 por lo que es necesario incluir un archivo de formato .ndm además de los archivos necesarios para realizar la verificación de DRC. En esta opción se añaden todos los sign offs que se realizaron en el proceso de síntesis física, mientras que con la opción comentada no se estarán tomando en cuenta. Los archivos necesarios para realizar esta prueba se encuentran dentro de la carpeta *archivos_importantes*. Este proceso se explica a mas detalle en [13]. Si se desea realizar la prueba sin considerar a los sign offs se puede descomentar la linea del comando de ICV que tan solo requiere de un archivo tipo GDS y el *runset*.

Cuadro 8.1: Parte del script de verificación *DRC*

```
156 ##############################################################################
157 # Prueba DRC
158 ##############################################################################
159 cd DRC
160 echo -e "\n\n\nINICIANDO PRUEBA DRC\nESTA PRUEBA PODRIA TARDAR Y NO MOSTRARA NADA EN LA TERMINAL\n\nPORFAVOR
    ↪  ESPERE A QUE TERMINE\n\n\n"
161 echo -e "\n\n\nINICIANDO PRUEBA DRC\n\n\n" >> $filepath/output.txt
162 icv -i $filepath/archivo.gds -c $cell_name -vue $filepath/archivos_importantes/DRC.215a_pre041518 >>
    ↪  $filepath/output.txt
163 #icv -icc2 -f NDM -i $ndm_name.ndm -p $filepath -c $cell_name -clf $filepath/archivos_importantes/slnFile.txt
    ↪  -icc_density_blockage -icc2_error_categories -I
    ↪  $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run -icc2_error_browser INST
    ↪  -icc2_error_cell signoff_check_drc.err -clf
    ↪  $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run/rule_pattern.clf -rc
    ↪  $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run/signoff_check_drc.rc -I
    ↪  $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run
    ↪  $filepath/archivos_importantes/DRC.215a_pre041518 >> $filepath/output.txt
164 cd ..
165 echo -e "\n\n\nVerificacion DRC completada\n\n\n" >> $filepath/output.txt
166 echo -e "\n\n\nVerificacion DRC completada\n\n\n"
```

Al finalizar la verificación se generan los archivos que se pueden observar en la Figura No. 3. Dentro de esta carpeta llamada DRC se encuentra el archivo .RESULTS que nos indica el resultado de la verificación física. Si el resultado se completa con éxito, se desplegara un mensaje de "CLEAN" mientras que si la verificación no se completa con éxito se despliega un mensaje de "ABORT" o "NOT CLEAN". Como se puede observar en la Figura No. 4, los resultados de la verifiación de DRC tienen errores por lo que el resultado fue "NOT CLEAN".

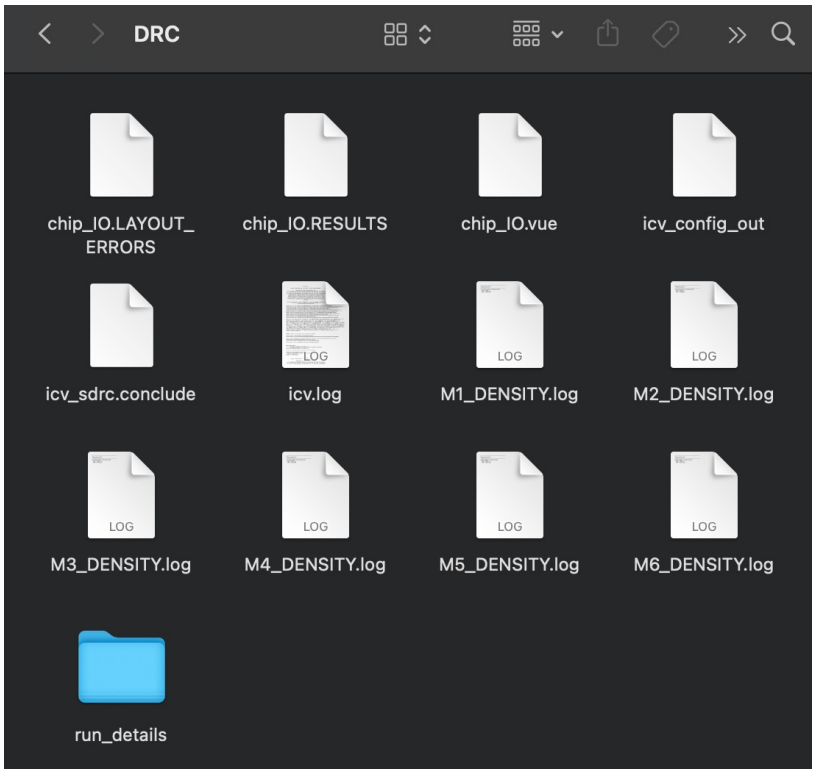


Figura 5: Archivos creados luego de ejecutar *DRC*

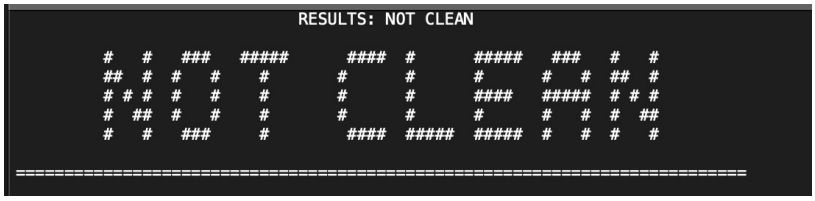


Figura 6: Resultado exitoso de verificación *DRC*

Los errores que se dan dentro de esta prueba se muestran en el Cuadro 8.2. Como se puede observar se tienen 6 errores de densidad. Estos errores suceden cuando dentro del núcleo no se cumple con el requisito de utilización mínima definida en el techfile.

Cuadro 8.2: Violaciones encontradas al realizar verificación de *DRC*

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

CAPÍTULO 9

Creación automatizada del Netlist CDL con formato ICV

Para poder realizar algunas de las *Verificaciones Físicas* es necesario un netlist CDL de formato ICV el cual se encuentra a nivel de transistor, en esta sección se menciona como obtener este archivo. Debido a que el acuerdo que tiene la universidad con el fabricante no cubre la creación de este archivo, este se debe crear utilizando las librerías de *Standard Cell Library* y /emphI/O Cell Library. Para obtener este archivo se toman los dos archivos HDL tipo verilog de la fase de síntesis lógica como entrada, los cuales se encuentran a nivel de compuerta. Utilizando el comando *cat* se juntan ambos verilogs en uno solo llamado *headers.v*. A estos verilogs se les remueve todo lo relacionado con entradas, salidas y módulos utilizando el comando *sed*. Luego, utilizando la herramienta de Synopsys NetTran se convierte el archivo anterior a un archivo de formato SPICE. Nuevamente se ejecuta NetTran con el archivo SPICE y el archivo Verilog original para realizar la última conversión a el netlist CDL de formato ICV. Todo esto se muestra en el siguiente script tipo BASH.

Cuadro 9.1: Parte del script de creación de el netlist ICV

```
83 ##############################################################################
84 # Generacion de netlist ICV
85 ##############################################################################
86 #cat $filepath/out_final_io.v $filepath/out_final.v>$filepath/ICV/headers.v
87 cat $filepath/out_final.v>headers.v
88 cd ICV
89 sed '/module\|endmodule\|input\|output\|inout/!d' headers.v >headers_mod.v
90 icv_nettran -verilog headers_mod.v -outType SPICE -outName libraries.sp
91 cat inicial.sp libraries.sp>libraries_f.sp
92 icv_nettran -verilog $filepath/out_final.v -sp libraries_f.sp -outType ICV
   ↪  -outName netlist.icv >> $filepath/output.txt
93 cd ..
```

```
94 echo -e "\n\n\nCreacion del netlist .icv completada\n\n\n" >>
   ↪ $filepath/output.txt
95 echo -e "\n\n\nCreacion del netlist .icv completada\n\n\n"
```

Lo siguiente es lo que agrega al archivo SPICE generado por los documentos verilog en el proceso de creación del archivo icv.

Cuadro 9.2: Parte del archivo *SP*

```
.GLOBAL VDD VSS VDDPST VSSPST
*.EQUATION
*.SCALE METER
*.MEGA
.PARAM
.INCLUDE direccion/archivos_importantes/source.added
```

CAPÍTULO 10

Proceso de *Electrical Rules Check* (ERC)

Para realizar el *Electrical Rules Check* es necesario utilizar los archivos con extensiones ICV y GDS al igual que el *runset* como se demostró en [8]. Dentro del runset hay que modificar algunas variables dentro de "ENVIRONMENT SETUP" las cuales indican las ubicaciones de los archivos mencionados y el nombre del topcell. También hay que añadir la dirección del archivo *unit.cdl*. Como se puede observar en la parte extraída del *runset*, la dirección de la carpeta y el nombre de topcell tienen un nombre genérico que se les asignó para realizar este proceso de forma automatizada.

Cuadro 10.1: *Environment Setup* del *runset* de *ERC*

```
761 //////////////////////
762 // ENVIRONMENT SETUP //
763 //////////////////////
764
765 library(
766   library_name = "direccioncarpeta/archivos_importantes/EL_GRAN_JAGUAR.gds",
767   cell = "topcell_name",
768   format = GDSII
769 );
770
771 SCHEMATIC_TOPCELL : string = "topcell_name"; // Set schematic top cell name
772 here
773 sch_db = schematic(
774   schematic_file = {{"direccioncarpeta/archivos_importantes/archivo.icv",
775   ICV}},
776   schematic_library_file =
777   {{"direccioncarpeta/archivos_importantes/unit.cdl", SPICE}},
778   expand_multiple_devices = true,
```

```
  spice_settings = {slash_is_space = false}
);
//#define USER_EQUIV_FILE // Turn on for user-specified equivalent point
file flow
#ifdef USER_EQUIV_FILE
#include "./user.equiv" // Set equivalent point file here
#endif
```

También es necesario modificar la ubicación del archivo *icv_ compare.rh* que se encuentra en "COMPARE", todos estos archivos se encuentran dentro de una carpeta llamada *archivos_ - importantes*.

```
user_functions_file = "direccioncarpeta/archivos_importantes/icv_compare.rh",
```

Las Reglas que deberemos editar dentro del *runset* se muestran en la tabla 1

Regla	**Configuracion**
Well to PG Check	Habilitado
Gate to PG Check	Habilitado
Path Check	Habilitado
DS to PG Check	Habilitado
Floating Gate Check	Habilitado
Floating Well Check	Habilitado
NW Ring	Deshabilitado

Tabla 1: Reglas del *netlist* para realizar verificación de *ERC*

Para que se realicen todas las pruebas del ERC es necesario realizar los siguientes cambios al archivo *runset*.

Cuadro 10.2: Configuración específica de verificación *ERC*

```
/* EDIT: The following section contains all of the runset variables for RC
extraction tools. */
// #define RC_DECK // Turn on for LPE/RC extraction
// #define CROSS_REFERENCE// Turn on for source cross reference in LPE
    ↪  extraction
// #define ZERO_NRS_NRD // Turn off when this deck would calculate NRS and
    ↪  NRD
// #define FILTER_DGS_TIED_MOS // Turn on to filter MOS with D, G and S tied
    ↪  together (default filter MOS with all pins tied)

#define WELL_TO_PG_CHECK // Default is on. Turn on to highlight if nwell
    ↪  connects to ground or psub connects to power.
#define GATE_TO_PG_CHECK // Default is off. Turn on to highlight if a mos
    ↪  gate directly connects to power or ground.
#define PATH_CHECK // Default is off. Turn on to highlight if
// (1) nodes have a path to power but no path to ground
```

```
797 // (2) nodes have a path to ground but no path to power
798 // (3) nodes have no path to power or ground
799 // (4) nodes have no path to any label net
800 #define DS_TO_PG_CHECK // Default is on. Turn on to highlight if drain
    ↪ connects to power and source connects to ground.
801 #define FLOATING_GATE_CHECK // Default is on. Turn on to highlight if there
    ↪ are floating gates.
802 #define FLOATING_WELL_CHECK^^I// Default is on. Turn on to highlight if well
    ↪ does not connect to power or ground
803 // The nwell of moscaps and nwell-resistor are excluded
804 //#define NW_RING // Turn on to enable NW ring to separate the node from
    ↪ BULK
```

El siguiente script utiliza el comando *sed* para reemplazar el nombre del topcell y la dirección de la carpeta en el runset necesario para ejecutar la verificación del *ERC*. Para que esto sea posible, es necesario utilizar una versión modificada del *runset* con los nombres que el comando *sed* buscará. También se utiliza el comando *pwd* para obtener la carpeta de trabajo actual. Si se quisiera cambiar el nombre del topcell, este se cambia en la variable llamada cell_name.

Cuadro 10.3: Parte del script de reemplazo de texto en el *runset* de verificación *ERC*

```
filepath=$(pwd)
cell_name=chip_IO
sed -i "s:direccioncarpeta:$filepath:" archivos_importantes/ERC.4a
sed -i "s:topcell_name:$cell_name:" archivos_importantes/ERC.4a
```

Luego de estos cambios al runset es posible ejecutar la verificación de ERC utilizando el siguiente comando de IC Validator. Donde el $filepath es la ubicación de la carpeta de trbajo y $cell_name es el nombre del topcell.

Cuadro 10.4: Comando de *IC Validator* utilizado para realizar verificación *ERC*

```
icv -i $filepath/EL_GRAN_JAGUAR.gds -c $cell_name -s
↪ $filepath/ICV/archivo.icv -sf ICV -vue
↪ $filepath/archivos_importantes/ERC.4a >> $filepath/output.txt
```

Si el resultado de la verificación es exitoso, esto se mostrara con un PASS y CLEAN en el archivo ṘESULTS como se muestra en la Figura No. 4. También se generaran los archivos que se muestran en la Figura No. 3. Si la verificación no se completa con éxito, se podrá leer un mensaje de FAIL y NOT CLEAN.

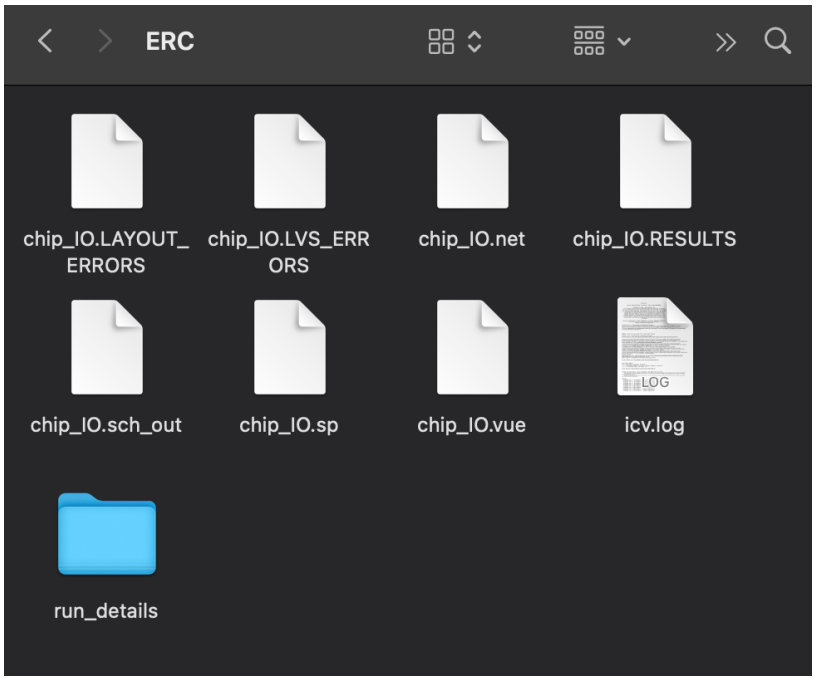


Figura 7: Archivos creados luego de ejecutar *ERC*

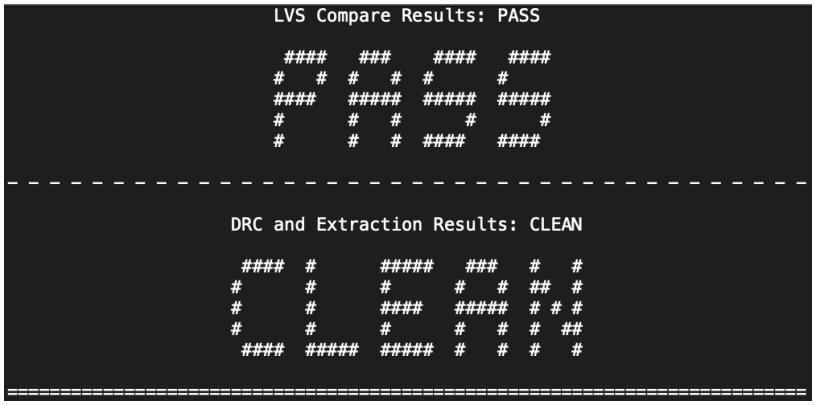


Figura 8: Resultado exitoso de *ERC*

CAPÍTULO 11

Proceso de *Layout Vs. Schematic* (LVS)

Similar al *Electrical Rules Check*, el *Layout vs Schematic* requiere de los mismos archivos, pero su runset es un poco diferente. A diferencia de la verificación de LVS requiere de encender la función de extracción de parásitos. Luego al igual que con el *runset* de ERC, es necesario modificar las variables de "ENVIRONMENT SETUP" de ubicaciones de archivos y el nombre del topcell como se muestran en el Cuadro 11.1.

Cuadro 11.1: *Environment Setup* del *runset* de *LVS*

```
761 //////////////////////
762 // ENVIRONMENT SETUP //
763 //////////////////////
764
765 library(
766   library_name = "direccioncarpeta/archivos_importantes/EL_GRAN_JAGUAR.gds",
767   cell = "topcell_name",
768   format = GDSII
769   );
770
771 SCHEMATIC_TOPCELL : string = "topcell_name"; // Set schematic top cell name
    ↪  here
772 sch_db = schematic(
773   schematic_file = {{"direccioncarpeta/archivos_importantes/archivo.icv",
    ↪  ICV}},
774   schematic_library_file =
    ↪  {{"direccioncarpeta/archivos_importantes/unit.cdl", SPICE}},
775   expand_multiple_devices = true,
776   spice_settings = {slash_is_space = false}
777 );
```

```
//#define USER_EQUIV_FILE // Turn on for user-specified equivalent point
↪  file flow
#ifdef USER_EQUIV_FILE
#include "./user.equiv" // Set equivalent point file here
#endif
```

Las Reglas que deberemos editar dentro del *runset* se muestran en la tabla 2:

Regla	**Configuracion**
Well to PG Check	Habilitado
Gate to PG Check	Deshabilitado
Path Check	Deshabilitado
DS to PG Check	Habilitado
Floating Gate Check	Habilitado
Floating Well Check	Habilitado
NW Ring	Deshabilitado

Tabla 2: Reglas del *netlist* para realizar verificación de *LVS*

También es necesario modificar la ubicación del archivo *icv_ compare.rh* que se encuentra en "COMPARE".

```
user_functions_file =
↪  "direccioncarpeta/archivos_importantes/icv_compare.rh",
```

Luego para asegurarnos de que se ejecuten todas las pruebas que deseamos realizamos los siguientes cambios al *runset*.

Cuadro 11.2: Configuración específica de verificación *LVS*

```
/* EDIT: The following section contains all of the runset variables for RC
↪  extraction tools. */
#define RC_DECK // Turn on for LPE/RC extraction
//#define CROSS_REFERENCE // Turn on for source cross reference in LPE
↪  extraction
//#define ZERO_NRS_NRD // Turn off when this deck would calculate NRS and
↪  NRD
//#define FILTER_DGS_TIED_MOS // Turn on to filter MOS with D, G and S tied
↪  together (default filter MOS with all pins tied)

#define WELL_TO_PG_CHECK // Default is on. Turn on to highlight if nwell
↪  connects to ground or psub connects to power.
//#define GATE_TO_PG_CHECK // Default is off. Turn on to highlight if a mos
↪  gate directly connects to power or ground.
//#define PATH_CHECK // Default is off. Turn on to highlight
// (1) nodes have a path to power but no path to ground
// (2) nodes have a path to ground but no path to power
```

```
797 // (3) nodes have no path to power or ground
798 // (4) nodes have no path to any label net
799 #define DS_TO_PG_CHECK // Default is on. Turn on to highlight if drain
    ↪ connects to power and source connects to ground.
800 #define FLOATING_GATE_CHECK // Default is on. Turn on to highlight if there
    ↪ are floating gates.
801 #define FLOATING_WELL_CHECK // Default is on. Turn on to highlight if well
    ↪ does not connect to power or ground. The nwell of moscaps and
    ↪ nwell-resistor are excluded
802 //#define NW_RING // Turn on to enable NW ring to separate the node from
    ↪ BULK
```

El siguiente script utiliza el comando *sed* para reemplazar el nombre del topcell y la dirección de la carpeta en el runset necesario para ejecutar la verificación del *LVS*, para que esto sea posible, es necesario utilizar una versión modificada del *runset* con los nombres que el comando *sed* buscará. También se utiliza el comando *pwd* para obtener la carpeta de trabajo actual. Si se quisiera cambiar el nombre del topcell, este se cambia en la variable llamada cell_name.

Cuadro 11.3: Parte del script de reemplazo de texto en el *runset* de verificación *LVS*

```
filepath=$(pwd)
cell_name=chip_IO
sed -i "s:direccioncarpeta:$filepath:" archivos_importantes/LVS.4a
sed -i "s:topcell_name:$cell_name:" archivos_importantes/LVS.4a
```

Luego de estos cambios al runset es posible ejecutar la verificación de LVS utilizando el siguiente comando de IC Validator, donde el $filepath es la ubicación de la carpeta de trabajo y $cell_name es el nombre del topcell.

Cuadro 11.4: Comando de *IC Validator* utilizado para realizar verificación *LVS*

```
icv -i $filepath/EL_GRAN_JAGUAR.gds -c $cell_name -s
↪ $filepath/ICV/archivo.icv -sf ICV -vue
↪ $filepath/archivos_importantes/nuevo_LVS.4a >> $filepath/output.txt
```

Si el resultado de la verificación es exitoso, esto se mostrara con un PASS y CLEAN en el archivo .RESULTS como se muestra en la Figura No. 6. Si la verificación no se completa con éxito, se podrá leer un mensaje de FAIL y NOT CLEAN. Los archivos que se producen al realizar la verificación se pueden apreciar en la Figura No. 5, como se puede observar los archivos son bastante similares a los creados en el proceso de ERC y DRC pero contienen una carpeta llamada *MILKYWAY_XTR* y unos archivos llamados *STARCXT.mapping* y *STARCXT.runset_rep* . Estos archivos son necesarios para realizar la extracción de parásitos.

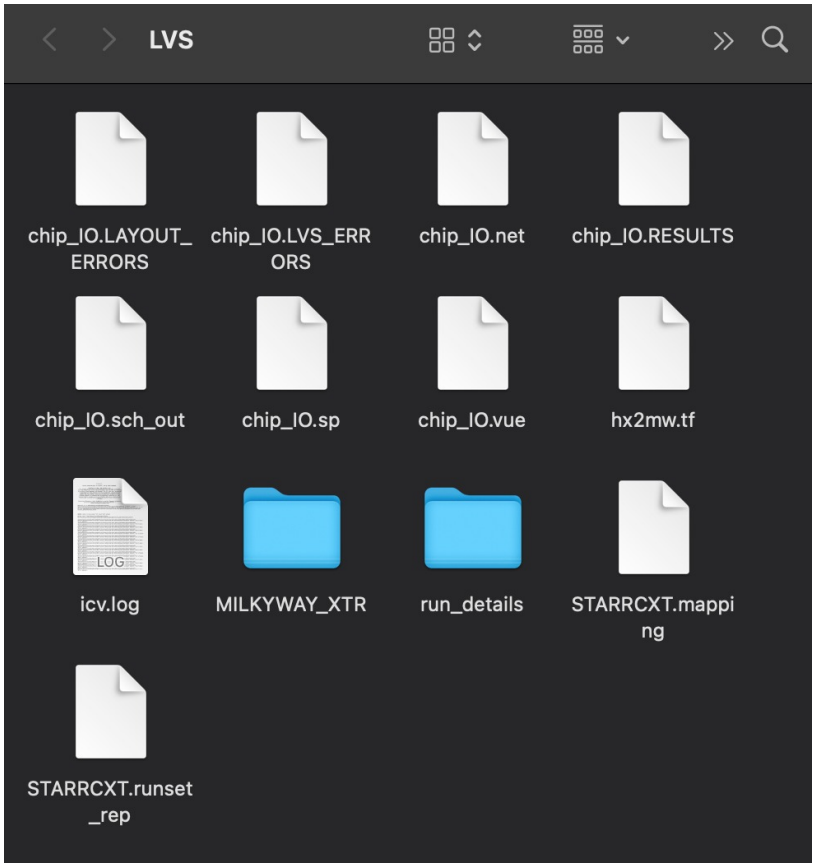


Figura 9: Archivos creados luego de ejecutar *LVS*

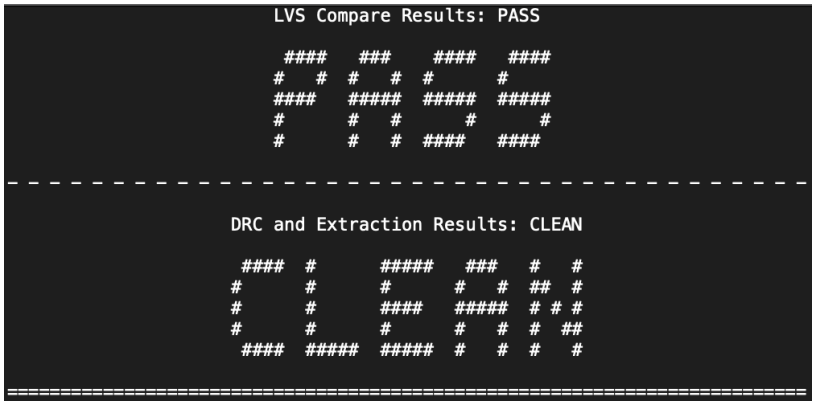


Figura 10: Resultado exitoso de *LVS*

CAPÍTULO 12

Creación automatizada de *blackboxes*

Para realizar las verificaciones de ERC y LVS es necesario tomar en cuenta los blackboxes que se están utilizando dentro del diseño, en este capítulo se menciona el funcionamiento del script automatizado que busca y agrega nuevos blackboxes en el *runset*. Un blackbox es un sistema que se puede dar a conocer en terminos de sus entradas y salidas. En nuestro caso, desconocemos el funcionamiento interno del blackbox pero tenemos una idea de como funcionan sus entradas y salidas. Los blackboxes juegan un papel importante en este proyecto ya que se desconocen muchos de los circuitos de TSMC que se generan durante las fases de síntesis. La representación de un blackbox se puede apreciar en la Figura 11.

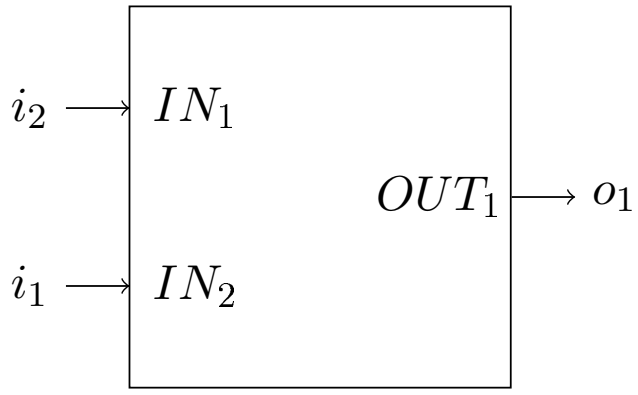


Figura 11: Ejemplo de un *blackbox*

Al no tener definidos los blackboxes los siguientes errores se mostrarán en un archivo con el nombre *nombre_del_topcell_lvs.log* dentro de la carpeta de *run_details* que se crea al realizar la prueba, estos errores en las verificaciones de ERC y LVS terminarán con un "ABORT".

Cuadro 12.1: Errores al ejecutar *ERC* o *LVS*

```
ERROR: Found schematic empty cell "AO31D1BWP7T" is defined in netlist but
↪  not declared as device in runset.
ERROR: Found schematic empty cell "OR3D1BWP7T" is defined in netlist but not
↪  declared as device in runset.
ERROR: Found schematic empty cell "OR4D1BWP7T" is defined in netlist but not
↪  declared as device in runset.
```

El script que genera los blackboxes utiliza el comando *grep* para buscar las palabras clave 'Found schematic empty cell' en un archivo que contiene todos los logs de la ejecución de LvS para verificar si existen nuevos blackboxes a generar, esto se guarda en una variable llamada *result*. Luego utilizando un *if* revisa si la variable contiene algún valor. Si la variable estuviese vacía, se despliega un mensaje en la terminal indicando que no se encontró ningún nuevo blackbox, esto no debería de suceder ya que se está ejecutando un runset sin ningún blackbox. Si la variable tiene algún valor guardado, se ejecuta el proceso para extraer los blackboxes del netlist ICV que se había generado anteriormente. Utilizando un bucle *for* se guarda el texto que debe tener un blackbox en una variable llamada `black_boxes`. Utilizando el comando *grep* dentro de un bucle *for* se realiza la extracción de cada pin de entrada y salida del blackbox. Se guarda cada pin de entrada y salida en una variable llamada `list`. Luego se añaden las comas y se guardan en una lista llamada `list2`. Al finalizar este bucle, se añaden los pines de entrada y salida junto con el texto '`remove_schematic_ports`' a la variable `black_boxes`. Por último, se añaden los blackboxes a un nuevo *runset* de LVS y ERC para no alterar el original. Al completar el proceso, se ejecuta ERC con el nuevo *runset* y se verifica que ya no existan nuevos blackboxes.

Cuadro 12.2: Script de fase de *blackboxes*

```
99  ##############################################################################
100 # Black Boxes
101 ##############################################################################
102 cat
    ↪  $filepath/archivos_importantes/LVS.4a>$filepath/archivos_importantes/nuevo_LVS.4a
103 result=$(grep 'Found schematic empty cell'
    ↪  $filepath/ERC/run_details/*_lvs.log | awk '{print $6}' | tr -d \")
104
105 item=$(grep 'Found schematic empty cell' $filepath/ERC/run_details/*_lvs.log
    ↪  | awk '{print $6}' | tr -d \")
106
107
108 if [ -n "$result" ]; then
109   echo -e "\n\n\nSE ENCONTRARON NUEVOS BLACK BOXES\n\n\n" >>
      ↪  $filepath/output.txt
110   echo -e "\n\n\nSE ENCONTRARON NUEVOS BLACK BOXES\n\n\n"
111   for objeto in ${item[@]}
112   do
113     black_boxes+=$(echo "lvs_black_box_options(\n")
114     black_boxes+=$(echo "        equiv_cells ={{schematic_cell =
        ↪  \"$objeto\", layout_cell = \"$objeto\"}},\n")
```

```
115
116
117     for i in $(grep -m1 -A 3 "=$objeto" $filepath/archivo.icv | tr -d "\n\r"
        ↪  | grep -m 1 -oP '(?<={pin).*?(?=}})' | awk NR==1)
118     do
119        list+=($(echo \"$i\" | sed 's/\(n\).*\(=\)/\1\2/' | sed 's/\<n\>//' |
           ↪  sed 's/.*=/"/g' |  sed 's/"{",/ /' | sed 's/}}",/"/'))
120     done
121     printf -v x "%s," "${list[@]}"
122     lista2=${x%,}
123     black_boxes+=$(echo "        remove_schematic_ports = {${lista2[@]}}\n")
124     black_boxes+=$(echo ");\n")
125
126     #echo "${list[@]}"
127     list=()
128     lista2=()
129   done
130
131
132 cat $filepath/archivos_importantes/ERC.4a | sed "s#\//BLACKBOXES
    ↪  AQUI#${black_boxes[@]}#g"  > $filepath/archivos_importantes/nuevo_ERC.4a
133 cat $filepath/archivos_importantes/LVS.4a | sed "s#\//BLACKBOXES
    ↪  AQUI#${black_boxes[@]}#g"  > $filepath/archivos_importantes/nuevo_LVS.4a
134
135   echo -e ${black_boxes[@]} >> $filepath/output.txt
136   echo -e "\n\n\nEJECUTANDO ERC NUEVAMENTE\n\n\n" >> $filepath/output.txt
137   echo -e ${black_boxes[@]}
138   echo -e "\n\n\nEJECUTANDO ERC NUEVAMENTE\n\n\n"
139   rm -rf ERC
140   mkdir ERC
141   cd ERC
142   icv -i $filepath/archivo.gds -c $cell_name -s $filepath/archivo.icv -sf
      ↪  ICV -vue $filepath/archivos_importantes/nuevo_ERC.4a >>
      ↪  $filepath/output.txt
143   cd ..
144 else
145   echo -e "\n\n\nNO SE ENCONTRARON NUEVOS BLACK BOXES\n\n\n" >>
      ↪  $filepath/output.txt
146   echo -e "\n\n\nNO SE ENCONTRARON NUEVOS BLACK BOXES\n\n\n"
147 fi
```

CAPÍTULO 13

Proceso de verificación de antena

La Verifiación de Antena se lleva a cabo en dos partes, la primera parte toma lugar durante el proceso de síntesis física en ICC2 y luego con IC Validator. Durante la primera parte se definen las propiedades de antena de la cell / celda y sus reglas. En la segunda parte se lleva a cabo la verificación utilizando IC Validator. En este trabajo solamente se toma en cuenta la segunda parte debido a que se quiere automatizar tan solo a las *Verificaciones Físicas*, todo el proceso de diseño adicional se lleva a cabo en las fases de síntesis física y síntesis lógica.

El comando que se muestra en el Cuadro 13.1 es el comando que se ejecuta para realizar la verificación física de antena. Al igual que la mayoría de las pruebas que han realizado, se requiere de un archivo GDS y del *runset* de la verificación para realizar la prueba. En este caso el runset de la verificación de antena se llama `ICVLM18_LM16_LM152_6M.215a_pre041518` y se encuentra en la carpeta *archivos_importantes*.

Cuadro 13.1: Comando de *IC Validator* utilizado para ejecutar verificación de antena

```
icv -i $filepath/EL_GRAN_JAGUAR.gds -c $cell_name -sf ICV -vue
↪  $filepath/archivos_importantes/ICVLM18_LM16_LM152_6M.215a_pre041518 >>
↪  $filepath/output.txt
```

Al finalizar la verificación de antena se producen los siguientes archivos dentro de una carpeta llamada *Antenna*. Como se puede observar se tiene un archivo .RESULTS que nos indica los resultados de la verificación, si la prueba se realizó con éxito, este devuelve "CLEAN" de lo contrario se produce un "ABORT" o "NOT CLEAN". El resultado correcto se puede observar en la Figura No. 13.

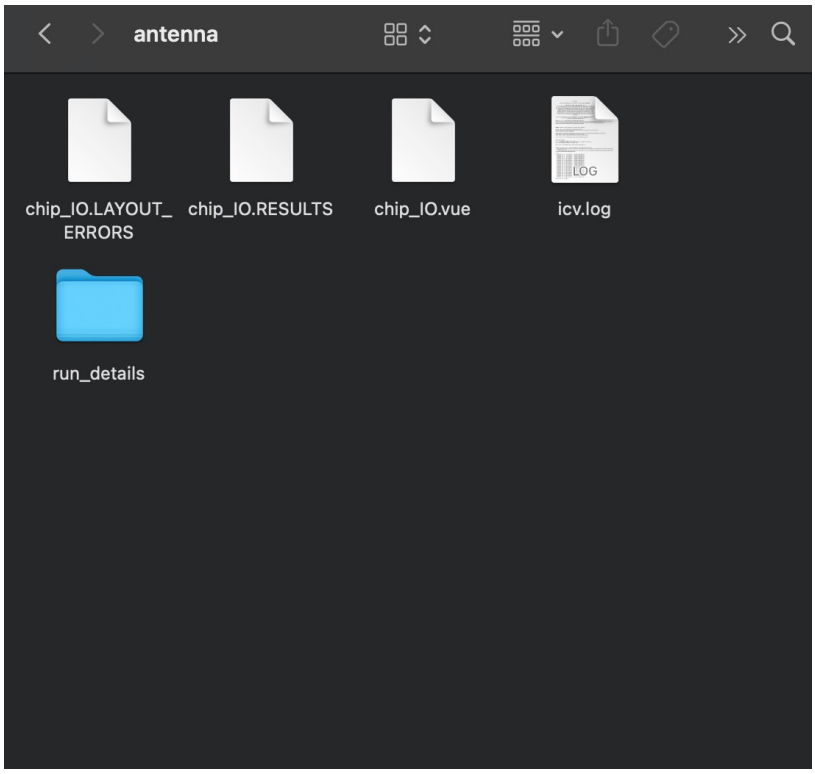


Figura 12: Archivos creados luego de ejecutar *Verificación de Antena*

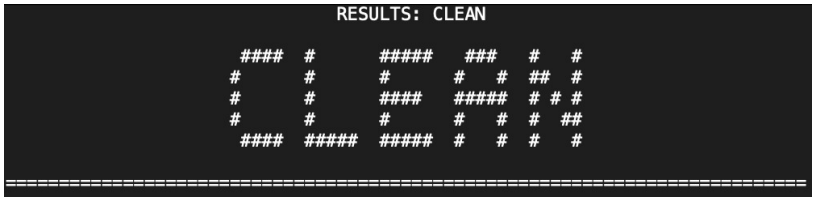


Figura 13: Resultado exitoso de *Verificación de Antena*

CAPÍTULO 14

Pruebas de script de *blackboxes*

Para verificar el funcionamiento adecuado del script realizado encargado de buscar y agregar blackboxes al runset original, se realizaron algunas pruebas con algunos circuitos sencillos. Entre estos circuitos se encuentran los siguientes circuitos NOT, XOR, full adder, contador de 4 bits y una ALU. Estos circuitos fueron los mismos utilizados el año pasado en el trabajos de graduación de [14] y [13] para asegurar que estos cumplen con todas las verificaciones y se tienen todos los archivos requeridos para realizar ERC y LVS. En estas pruebas se utilizan solamente el archivo ICV y el archivo GDS.

14.1. Compuerta NOT

Se utilizó una compuerta NOT como un circuito de complejidad baja para verificar el funcionamiento del script de creación de blackboxes. Para realizar esta prueba se copiaron y renombraron los archivos de formato GDS e ICV encontrados en la carpeta compartida. Luego se realizó una prueba con el netlist original para asegurar que ambas verificaciones se completan con éxito, esta prueba se completó con éxito. Luego se volvió a ejecutar el script utilizando el runset vacío y los resultados se muestran en las figuras 14 y 15. Los blackboxes encontrados por el script se muestran en el Cuadro 19.3 en el capítulo de anexos.

Figura 14: Resultados de la prueba *ERC* de una compuerta *NOT*

Figura 15: Resultados de la prueba *LVS* de una compuerta *NOT*

Como se puede observar en las figuras, ambas verificaciones físicas fueron completadas con éxito con el runset que no contenía ningún blackbox. En las figuras también se muestra el comando de terminal que se utilizó para ejecutar las verificaciones junto con el nombre de su topcell.

14.2. Compuerta XOR

Se utilizó la compuerta XOR como una compuerta de mayor complejidad que la compuerta NOT, pero como un circuito de complejidad baja. Como se puede observar en el Cuadro 19.4, que se encuentra capítulo de anexos, la compuerta XOR tiene dos blackboxes más, pero sigue siendo un circuito sencillo. Las blackboxes de este circuito y el anterior se pudieron hacer fácilmente sin necesidad del script, pero son un buen indicador del funcionamiento

del script debido a que es más fácil encontrar un error si existiera alguno. Para realizar la prueba, se copiaron y renombraron los archivos nuevamente y se cambió el nombre del topcell en el script. En las figuras 16 y 17 se pueden observar los resultados de la prueba.

Figura 16: Resultados de la prueba *ERC* de una compuerta *XOR*

Figura 17: Resultados de la prueba *LVS* de una compuerta *XOR*

En las figuras se puede observar que ambas pruebas fueron completadas con éxito para el topcell *Xor_IO*.

14.3. Circuito full adder

Se utilizó un full adder como un circuito de complejidad mediana, en el Cuadro 19.5 del capítulo de anexos se pueden encontrar los 8 blackboxes que se generaron con el script. Nuevamente, para realizar la prueba se repitió el proceso anterior. Los resultados se muestran en las figuras 18 y 19.

Figura 18: Resultados de la prueba *ERC* de un *Full Adder*

Figura 19: Resultados de la prueba *LVS* de un *Full Adder*

En las figuras se muestra el nombre del topcell que se utilizó al crear este circuito junto con los resultados exitosos de las verificaciones físicas.

14.4. Circuito contador de 4 bits

El contador de 4 bits se utilizó como un circuito de mediana dificultad. Este circuito es diferente al resto debido a que es un circuito secuencial, por lo que tendrá una señal de reloj. Se repite el proceso preestablecido y los resultados se encuentran en las figuras 20 y 21. Los blackboxes resultantes se muestran en el Cuadro 19.6 del capítulo de anexos.

Figura 20: Resultados de la prueba *ERC* de un *Contador de 4 bits*

Figura 21: Resultados de la prueba *LVS* de un *Contador de 4 bits*

En ambos casos los resultados de las pruebas fueron exitosas y se encontraron 14 nuevos blackboxes.

14.5. Circuito ALU

Se selecciono a una ALU como un circuito de mayor complejidad, aunque este aún no se acerca a la complejidad del Gran Jaguar. Los 29 blackboxes encontrados se encuentran en el Cuadro 19.7 del capítulo de anexos. En las figuras 22 y 23 se muestran los resultados de dicha prueba.

Figura 22: Resultados de la prueba ERC de una compuerta XOR

Figura 23: Resultados de la prueba *LVS* de una compuerta *XOR*

Se puede observar el nombre del topcell y las verificaciones ejecutadas con éxito en ambas imágenes.

CAPÍTULO 15

Integración de scripts de automatización

Para realizar la interfaz gráfica para el usuario, cada miembro del grupo de integración añadió a la carpeta compartida su script junto con cualquier archivo que se necesite para ejecutar su script con éxito. Luego cada miembro añadió su script a un código Python para ser ejecutado con la interfaz gráfica. Este script se muestra en el Cuadro 19.1 del capítulo de anexos. En la Figura 24 se muestra la interfaz gráfica.

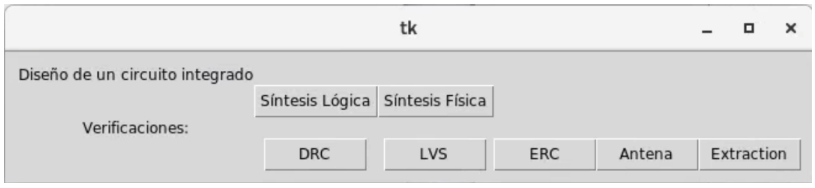


Figura 24: Interfaz gráfica de integración

Como se puede observar, se tiene un botón para cada fase de diseño automatizada. El código que se muestra en anexos manda a llamar un script de formato BASH cuando se presiona un botón. Luego este script manda a llamar el script principal que realiza la fase de diseño dentro de su respectiva carpeta. Este script manda a llamar los archivos que necesita de las carpetas de los otros miembros del grupo y realiza el proceso asignado.

CAPÍTULO 16

Conclusiones

- Se implementó un método para automatizar el proceso de *Verificaciones Físicas*, con el fin de facilitar lo más posible el proceso lento y tedioso.
- Se demostró el funcionamiento de las aplicaciones de *Synopsys* junto con sus limitaciones y requisitos.
- La comunicación con otros miembros del equipo permitió realizar todo el flujo de las *Verificaciones Físicas* con éxito.
- Se lograron ejecutar las verificaciones de *ERC* y *LVS* por medio de la creación automatizada de blackboxes.
- Se generaron errores para luego ser corregidos con el script de blackboxes en las verificaciones de *ERC* y *LVS*.
- Se logró integrar los trabajos de los integrantes del grupo de automatización.

CAPÍTULO 17

Recomendaciones

1. Utilizar las páginas *man* de *Sistemas Operativos Linux* para aprender la sintaxis y opciones de los comandos de la terminal.
2. Realizar pruebas al script con circuitos más complejos para eliminar cualquier *bug* que podría existir.
3. Mantener el trabajo en equipo para identificar rápidamente cualquier cambio que se le deba realizar al script de automatización.
4. Leer la documentación de Synopsys de las herramientas utilizadas para tener un buen entendimiento de la capacidad de la aplicación.
5. Replicar el trabajo hasta la actualidad sin la fase de automatización para entender el funcionamiento de los scripts.
6. Crear una instalación virtual de cualquier *Sistema Operativo Linux*, preferiblemente CentOS o RHEL, para trabajar sin las limitaciones de la red.

CAPÍTULO 18

Bibliografía

[1] M. Gianfagna. "What is Moore's Law?" (2022), dirección: `https://www.synopsys.com/glossary/what-is-moores-law.html`. (Encontrado: 24.08.2022).

[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.*" (2014). (Encontrado: 20.03.2022).

[3] S. H. Rubio. "*Definición del Flujo de Diseño para Fabricación de un Chip con Tecnología VLSI CMOS.*" (2019). (Encontrado: 20.03.2022).

[4] L. A. Nájera. "*Implementación de circuitos sintetizados a nivel netlist a partir de un diseño en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado.*" (2019). (Encontrado: 20.03.2022).

[5] J. N. Ruano. "*Definición del flujo en la herramienta VCS para la simulación de HDLs en la Fabricación de un Chip con Tecnología Nanométrica CMOS.*" (2020). (Encontrado: 20.03.2022).

[6] M. Sibrian. "*Verificación de reglas de diseño (DRC) para el desarrollo de un flujo funcional de un circuito integrado con tecnología nanométrica.*" (2020). (Encontrado: 20.03.2022).

[7] J. R. Girón. "*Etapa de verificación física de Diseño en Silicio vs. Esquemático (LVS) en el flujo de diseño para un chip a nanoescala", addendum = "(Encontrado: 20.03.2022).*" (2020).

[8] M. G. 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.*" (2020). (Encontrado: 20.03.2022).

[9] C. A. Cruz. "*Ejecución y utilización de un flujo de diseño para el desarrollo de un chip con tecnología nanométrica: Extracción de componentes parásitos y simulaciones en HSPICE.*" (2020). (Encontrado: 20.03.2022).

[10] K. S. Cardona. "*Mejoramiento del proceso de síntesis lógica llevada a cabo para la elaboración de un circuito integrado a escala nanométrica.*" (2021). (Encontrado: 20.03.2022).

[11] E. O. Torres. "*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.*" (2022). (Encontrado: 20.03.2022).

[12] J. A. 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.*" (2021). (Encontrado: 20.03.2022).

[13] 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.*" (2021). (Encontrado: 20.03.2022).

[14] J. E. 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.*" (2021). (Encontrado: 20.03.2022).

[15] J. A. Ruiz. "*Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: ejecución de la fase de verificación física Layout vs Schematic (LVS).*" (2021). (Encontrado: 20.03.2022).

[16] L. E. Abadía. "*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.*" (2021). (Encontrado: 20.03.2022).

[17] S. Jiang Jie-Hong (Roland) y Devadas. "Logic Synthesis in a nutshell." (2009), dirección: `https://www.sciencedirect.com/topics/computer-science/logic-synthesisl`. (Encontrado: 20.03.2022).

[18] Synopsys. "What is Physical Synthesis?" (2022), dirección: `https://www.synopsys.com/glossary/what-is-physical-synthesis.html`. (Encontrado: 20.03.2022).

[19] L. to Code. "What is Bash Used For?" (2022), dirección: `https://www.codecademy.com/resources/blog/what-is-bash-used-for/`. (Encontrado: 20.03.2022).

[20] D. Garn. "Five ways to use redirect operators in Bash." (2021), dirección: `https://www.redhat.com/sysadmin/redirect-operators-bash`. (Encontrado: 30.05.2022).

[21] D. Mckay. "How to Work with Variables in Bash." (2019), dirección: `https://www.howtogeek.com/442332/how-to-work-with-variables-in-bash/`. (Encontrado: 30.05.2022).

[22] G. B. "Comando cat de Linux – Con ejemplos de uso." (2022), dirección: `https://www.hostinger.es/tutoriales/comando-cat-linux`. (Encontrado: 30.05.2022).

[23] Tecmint. "15 Practical Examples of 'echo' command in Linux." (2021), dirección: `https://www.tecmint.com/echo-command-in-linux/`. (Encontrado: 12.05.2022).

[24] Javatpoint. "Linux cd Command | Linux change directory." (2021), dirección: `https://www.javatpoint.com/linux-cd`. (Encontrado: 12.05.2022).

[25] A. Gupta. "rm command in Linux with examples." (2021), dirección: `https://www.geeksforgeeks.org/rm-command-linux-examples/`. (Encontrado: 12.05.2022).

[26] G. Jevtic. "How to Use mkdir Command to Make or Create a Linux Directory." (2019), dirección: `https://phoenixnap.com/kb/create-directory-linux-mkdir-command`. (Encontrado: 12.05.2022).

[27] Schkn. "Bash If Else Syntax With Examples." (2021), dirección: `https://devconnected.com/bash-if-else-syntax-with-examples/`. (Encontrado: 12.05.2022).

[28] N. Lager y R. Gerardi. "Introduction to Linux Bash programming: 5 'for' loop tips." (2021), dirección: https://www.redhat.com/sysadmin/bash-scripting-loops. (Encontrado: 12.05.2022).

[29] D. Chobits. "Uso del comando SED en Linux y UNIX con ejemplos." (2019), dirección: https://www.ochobitshacenunbyte.com/2019/05/28/uso-del-comando-sed-en-linux-y-unix-con-ejemplos/. (Encontrado: 12.05.2022).

CAPÍTULO 19

Anexos

Cuadro 19.1: Codigo de interfaz gráfica

```
from tkinter import *
from tkinter import ttk
import os
import subprocess

def logica():
    subprocess.Popen(["./sintesislogica.bash"])

def fisica():
    subprocess.Popen(["./sintesisfisica.bash"])

def ERC():
    subprocess.Popen(["./erc.bash"])

def LVS():
    subprocess.Popen(["./lvs.bash"])

def DRC():
    subprocess.Popen(["./drc.bash"])

def antena():
    subprocess.Popen(["./antena.bash"])

def extraccion():
    subprocess.Popen(["./Extraction.bash"])

root = Tk()
frm = ttk.Frame(root, padding=10)
frm.grid()
```

```
30 ttk.Label(frm, text="Diseno de un circuito integrado").grid(column=0, row=0)
31 ttk.Button(frm, text="Sintesis Logica", command=logica).grid(column=1,
   ↪  row=1)
32 ttk.Button(frm, text="Sintesis Fisica", command=fisica).grid(column=2,
   ↪  row=1)
33 ttk.Label(frm, text="Verificaciones:").grid(column=0, row=2)
34 ttk.Button(frm, text="DRC", command=DRC).grid(column=1, row=3)
35 ttk.Button(frm, text="LVS", command=LVS).grid(column=2, row=3)
36 ttk.Button(frm, text="ERC", command=ERC).grid(column=3, row=3)
37 ttk.Button(frm, text="Antena", command=Antena).grid(column=4, row=3)
38 ttk.Button(frm, text="Extraction", command=extraccion).grid(column=5, row=3)
39
40 root.mainloop()
```

Cuadro 19.2: Script de automatización de Verificaciones Físicas

```
#!/bin/bash
###############################################################################################
# Creado por: Stefan Schwendener
# Nombre: run.bash
# Descripcion: automatizacion de las verificaciones fisicas
# Colocar el archivo EL_GRAN_JAGUAR.gds, out_final.v y out_final_io.v dentro de la carpeta principal
# La carpeta archivos_importantes debe estar en el mismo directorio que run.bash
###############################################################################################
filepath=$(pwd)
cell_name=chip_IO
ndm_name=EL_GRAN_JAGUAR
item=()
list=()
lista2=()
black_boxes=()
rm -f output.txt
touch output.txt
echo -e "\n\n\nEl directorio en que se encuentra es: $filepath\n\n\n" >> $filepath/output.txt
echo -e "\n\n\nEl directorio en que se encuentra es: $filepath\n\n\n"
###############################################################################################
# Eliminacion de remanentes
###############################################################################################
rm -rf ICV
rm -rf ERC
rm -rf LVS
rm -rf DRC
rm -rf Antenna
rm -rf SALIDAS
rm $filepath/archivos_importantes/nuevo_LVS.4a
```

```
rm $filepath/archivos_importantes/nuevo_ERC.4a
rm -f archivo.icv
echo -e "\n\n\nRemanentes eliminados\n\n\n" >> $filepath/output.txt
echo -e "\n\n\nRemanentes eliminados\n\n\n"
##############################################################################################
# Creacion de Folders y Archivos
##############################################################################################
mkdir ICV
mkdir ERC
mkdir LVS
mkdir DRC
mkdir Antenna
mkdir SALIDAS
chmod gou+wr ICV
chmod gou+wr ERC
chmod gou+wr LVS
chmod gou+wr DRC
chmod gou+wr Antenna
chmod gou+wr SALIDAS
touch output.txt
echo -e "\n\n\nFolders Creados\n\n\n" >> $filepath/output.txt
echo -e "\n\n\nFolders Creados\n\n\n"
##############################################################################################
# Modificacion de Ubicacion del Folder
##############################################################################################
sed -i "s:ruta_dir:$(pwd):" archivos_importantes/LVS.4a
sed -i "s:ruta_dir:$(pwd):" archivos_importantes/ERC.4a
sed -i "s:ruta_dir:$(pwd):" archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run/signoff_check_drc.rc
sed -i "s:ruta_dir:$(pwd):" archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run/signoff_check_drc.err

sed -i "s:topcell_name:$cell_name:" archivos_importantes/LVS.4a
sed -i "s:topcell_name:$cell_name:" archivos_importantes/ERC.4a
```

```
62
63 echo -e "\n\n\nModificacion de ubicacion de carpeta modificada\n\n\n" >> $filepath/output.txt
64 echo -e "\n\n\nModificacion de ubicacion de carpeta modificada\n\n\n"
65 ##########################################################################################################
66 # Lo que debemos anadir al archivo spice
67 ##########################################################################################################
68 touch ICV/inicial.sp
69 echo ".GLOBAL VDD VSS VDDPST VSSPST" >> ICV/inicial.sp
70 echo "*.EQUATION" >> ICV/inicial.sp
71 echo "*.SCALE METER" >> ICV/inicial.sp
72 echo "*.MEGA" >> ICV/inicial.sp
73 echo ".PARAM" >> ICV/inicial.sp
74 echo ".INCLUDE $(pwd)/archivos_importantes/source.added" >> ICV/inicial.sp
75 echo -e "\n\n\nSe creo lo que se anadira al archivo .sp\n\n\n" >> $filepath/output.txt
76 echo -e "\n\n\nSe creo lo que se anadira al archivo .sp\n\n\n"
77 ##########################################################################################################
78 # Generacion de Archivo ICV
79 ##########################################################################################################
80 cat $filepath/out_final_io.v $filepath/out_final.v>$filepath/ICV/headers.v
81 #cat $filepath/out_final.v>$filepath/ICV/headers.v
82 cd ICV
83 sed '/module\|endmodule\|input\|output\|inout/!d' headers.v >headers_mod.v
84 icv_nettran -verilog headers_mod.v -outType SPICE -outName libraries.sp
85 cat inicial.sp libraries.sp>libraries_f.sp
86 icv_nettran -verilog $filepath/out_final.v -sp libraries_f.sp -outType ICV -outName archivo.icv >>
   ↪  $filepath/output.txt
87 cd ..
88 cp ICV/archivo.icv $filepath
89 echo -e "\n\n\nCreacion del archivo .icv completada\n\n\n" >> $filepath/output.txt
90 echo -e "\n\n\nCreacion del archivo .icv completada\n\n\n"
91 ##########################################################################################################
92 # Prueba ERC
```

```
93  ##############################################################################################
94  cd ERC
95  icv -i $filepath/archivo.gds -c $cell_name -s $filepath/archivo.icv -sf ICV -vue
    ↪  $filepath/archivos_importantes/ERC.4a >> $filepath/output.txt
96  cd ..
97  echo -e "\n\n\nVerificacion ERC completada\n\n\n" >> $filepath/output.txt
98  echo -e "\n\n\nVerificacion ERC completada\n\n\n"
99  ##############################################################################################
100 # Black Boxes
101 ##############################################################################################
102 cat $filepath/archivos_importantes/LVS.4a>$filepath/archivos_importantes/nuevo_LVS.4a
103 result=$(grep 'Found schematic empty cell' $filepath/ERC/run_details/*_lvs.log | awk '{print $6}' | tr -d \")
104
105 item=$(grep 'Found schematic empty cell' $filepath/ERC/run_details/*_lvs.log | awk '{print $6}' | tr -d \")
106
107
108 if [ -n "$result" ]; then
109   echo -e "\n\n\nSE ENCONTRARON NUEVOS BLACK BOXES\n\n\n" >> $filepath/output.txt
110   echo -e "\n\n\nSE ENCONTRARON NUEVOS BLACK BOXES\n\n\n"
111   for objeto in ${item[@]}
112   do
113     black_boxes+=$(echo "lvs_black_box_options(\n")
114     black_boxes+=$(echo "        equiv_cells ={{schematic_cell = \"$objeto\", layout_cell = \"$objeto\"}},\n")
115
116
117     for i in $(grep -m1 -A 3 "=$objeto" $filepath/archivo.icv | tr -d "\n\r" | grep -m 1 -oP '(?<={pin).*?(?=}})'
      ↪  | awk NR==1)
118     do
119        list+=($(echo \"$i\" | sed 's/\(n\).*\(=\)/\1\2/' | sed 's/\<n\>//' | sed 's/.*=/"/g' |  sed 's/"{",/ /' |
        ↪  sed 's/}}",/"/'))
120     done
121     printf -v x "%s," "${list[@]}"
```

```
    lista2=${x%,}
    black_boxes+=$(echo "        remove_schematic_ports = {${lista2[@]}}\n")
    black_boxes+=$(echo ");\n")

    #echo "${list[@]}"
    list=()
    lista2=()
  done


cat $filepath/archivos_importantes/ERC.4a | sed "s#\//BLACKBOXES AQUI#${black_boxes[@]}#g"  >
↪  $filepath/archivos_importantes/nuevo_ERC.4a
cat $filepath/archivos_importantes/LVS.4a | sed "s#\//BLACKBOXES AQUI#${black_boxes[@]}#g"  >
↪  $filepath/archivos_importantes/nuevo_LVS.4a

  echo -e ${black_boxes[@]} >> $filepath/output.txt
  echo -e "\n\n\nEJECUTANDO ERC NUEVAMENTE\n\n\n" >> $filepath/output.txt
  echo -e ${black_boxes[@]}
  echo -e "\n\n\nEJECUTANDO ERC NUEVAMENTE\n\n\n"
  rm -rf ERC
  mkdir ERC
  cd ERC
  icv -i $filepath/archivo.gds -c $cell_name -s $filepath/archivo.icv -sf ICV -vue
  ↪  $filepath/archivos_importantes/nuevo_ERC.4a >> $filepath/output.txt
  cd ..
else
  echo -e "\n\n\nNO SE ENCONTRARON NUEVOS BLACK BOXES\n\n\n" >> $filepath/output.txt
  echo -e "\n\n\nNO SE ENCONTRARON NUEVOS BLACK BOXES\n\n\n"
fi
##############################################################################################
# Prueba LVS
##############################################################################################
```

```
cd LVS
echo -e "\n\n\nINICIANDO PRUEBA LVS\n\n\n"
icv -i $filepath/archivo.gds -c $cell_name -s $filepath/archivo.icv -sf ICV -vue
    $filepath/archivos_importantes/nuevo_LVS.4a >> $filepath/output.txt
echo -e "\n\n\nVERIFICACION LVS COMPLETA\n\n\n"
cd ..
##############################################################################################
# Prueba DRC
##############################################################################################
cd DRC
echo -e "\n\n\nINICIANDO PRUEBA DRC\nESTA PRUEBA PODRIA TARDAR Y NO MOSTRARA NADA EN LA TERMINAL\n\nPORFAVOR
    ESPERE A QUE TERMINE\n\n\n"
echo -e "\n\n\nINICIANDO PRUEBA DRC\n\n\n" >> $filepath/output.txt
icv -i $filepath/archivo.gds -c $cell_name -vue $filepath/archivos_importantes/DRC.215a_pre041518 >>
    $filepath/output.txt
#icv -icc2 -f NDM -i $ndm_name.ndm -p $filepath -c $cell_name -clf $filepath/archivos_importantes/slnFile.txt
    -icc_density_blockage -icc2_error_categories -I
    $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run -icc2_error_browser INST
    -icc2_error_cell signoff_check_drc.err -clf
    $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run/rule_pattern.clf -rc
    $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run/signoff_check_drc.rc -I
    $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run
    $filepath/archivos_importantes/DRC.215a_pre041518 >> $filepath/output.txt
cd ..
echo -e "\n\n\nVerificacion DRC completada\n\n\n" >> $filepath/output.txt
echo -e "\n\n\nVerificacion DRC completada\n\n\n"
##############################################################################################
# Prueba de Antenna
##############################################################################################
cd Antenna
icv -i $filepath/archivo.gds -c $cell_name -sf ICV -vue
    $filepath/archivos_importantes/ICVLM18_LM16_LM152_6M.215a_pre041518 >> $filepath/output.txt
```

```
cd ..
echo -e "\n\n\nVerificacion de antenna completada\n\n\n" >> $filepath/output.txt
echo -e "\n\n\nVerificacion de antenna completada\n\n\n"
#################################################################################################
# Restaurar Direcciones de texto
#################################################################################################
sed -i "s:$(pwd):ruta_dir:" $filepath/archivos_importantes/LVS.4a
sed -i "s:$(pwd):ruta_dir:" $filepath/archivos_importantes/ERC.4a
sed -i "s:$(pwd):ruta_dir:"
↪  $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run/signoff_check_drc.rc
sed -i "s:$(pwd):ruta_dir:"
↪  $filepath/archivos_importantes/signoff_fix_drc_run/signoff_check_drc_run/signoff_check_drc.err


sed -i "s:$cell_name:topcell_name:" $filepath/archivos_importantes/LVS.4a
sed -i "s:$cell_name:topcell_name:" $filepath/archivos_importantes/ERC.4a
#################################################################################################
# Copia de Archivos Importantes
#################################################################################################
cp -f $filepath/ERC/$cell_name.RESULTS $filepath/SALIDAS/ERC.RESULTS
cp -f $filepath/LVS/$cell_name.RESULTS $filepath/SALIDAS/LVS.RESULTS
cp -f $filepath/DRC/$cell_name.RESULTS $filepath/SALIDAS/DRC.RESULTS
cp -f $filepath/Antenna/$cell_name.RESULTS $filepath/SALIDAS/Antenna.RESULTS
cp -f $filepath/output.txt $filepath/SALIDAS/TERMINAL.txt

echo -e "\n\n\nSe copiaron los resultados a /SALIDAS\n\n\n" >> $filepath/output.txt
echo -e "\n\n\nLos resultados estan en /SALIDAS\n\n\n"
```

Cuadro 19.3: Blackboxes generados por el script para una compuerta NOT

```
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PVSS1CDG", layout_cell =
        ↪  "PVSS1CDG"}},
        remove_schematic_ports = {"VSS"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "TIEHBWP7T", layout_cell =
        ↪  "TIEHBWP7T"}},
        remove_schematic_ports = {"Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PDDW0204SCDG", layout_cell =
        ↪  "PDDW0204SCDG"}},
        remove_schematic_ports = {"I","DS","OEN","PAD","C","PE","IE"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "TIELBWP7T", layout_cell =
        ↪  "TIELBWP7T"}},
        remove_schematic_ports = {"ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PVDD1CDG", layout_cell =
        ↪  "PVDD1CDG"}},
        remove_schematic_ports = {"VDD"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "CKNDOBWP7T", layout_cell =
        ↪  "CKNDOBWP7T"}},
        remove_schematic_ports = {"I","ZN"}
);
```

Cuadro 19.4: Blackboxes generados por el script para una compuerta XOR

```
vs_black_box_options(
        equiv_cells ={{schematic_cell = "PVSS1CDG", layout_cell =
        ↪  "PVSS1CDG"}},
        remove_schematic_ports = {"VSS"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PDDW0204SCDG", layout_cell =
        ↪  "PDDW0204SCDG"}},
        remove_schematic_ports = {"I","DS","OEN","PAD","C","PE","IE"}
);
lvs_black_box_options(
```

```
        equiv_cells ={{schematic_cell = "PVDD1CDG", layout_cell =
        ↪  "PVDD1CDG"}},
        remove_schematic_ports = {"VDD"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "TIELBWP7T", layout_cell =
        ↪  "TIELBWP7T"}},
        remove_schematic_ports = {"ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "TIEHBWP7T", layout_cell =
        ↪  "TIEHBWP7T"}},
        remove_schematic_ports = {"Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "CKND0BWP7T", layout_cell =
        ↪  "CKND0BWP7T"}},
        remove_schematic_ports = {"I","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "AN2D0BWP7T", layout_cell =
        ↪  "AN2D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "OR2D0BWP7T", layout_cell =
        ↪  "OR2D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","Z"}
);
```

Cuadro 19.5: Blackboxes generados por el script para una circuito *Full Adder*

```
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PVSS1CDG", layout_cell =
        ↪  "PVSS1CDG"}},
        remove_schematic_ports = {"VSS"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PDDW0204SCDG", layout_cell =
        ↪  "PDDW0204SCDG"}},
        remove_schematic_ports = {"I","DS","OEN","PAD","C","PE","IE"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PVDD1CDG", layout_cell =
        ↪  "PVDD1CDG"}},
        remove_schematic_ports = {"VDD"}
);
lvs_black_box_options(
```

```
        equiv_cells ={{schematic_cell = "IOA21D0BWP7T", layout_cell =
        ↪  "IOA21D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "XOR3D0BWP7T", layout_cell =
        ↪  "XOR3D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","A3","Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "OAI21D0BWP7T", layout_cell =
        ↪  "OAI21D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "MAOI222D1BWP7T", layout_cell =
        ↪  "MAOI222D1BWP7T"}},
        remove_schematic_ports = {"A","B","C","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "INVD1BWP7T", layout_cell =
        ↪  "INVD1BWP7T"}},
        remove_schematic_ports = {"I","ZN"}
);
```

Cuadro 19.6: Blackboxes generados por el script para una circuito *Contador de 4 bits*

```
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PVDD1CDG", layout_cell =
        ↪  "PVDD1CDG"}},
        remove_schematic_ports = {"VDD"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "TIELBWP7T", layout_cell =
        ↪  "TIELBWP7T"}},
        remove_schematic_ports = {"ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PVSS1CDG", layout_cell =
        ↪  "PVSS1CDG"}},
        remove_schematic_ports = {"VSS"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "TIEHBWP7T", layout_cell =
        ↪  "TIEHBWP7T"}},
        remove_schematic_ports = {"Z"}
);
```

```
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PDDW0204SCDG", layout_cell =
        ↪  "PDDW0204SCDG"}},
        remove_schematic_ports = {"I","DS","OEN","PAD","C","PE","IE"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "ND2D1BWP7T", layout_cell =
        ↪  "ND2D1BWP7T"}},
        remove_schematic_ports = {"A1","A2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "NR2D1BWP7T", layout_cell =
        ↪  "NR2D1BWP7T"}},
        remove_schematic_ports = {"A1","A2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "CKXOR2D1BWP7T", layout_cell =
        ↪  "CKXOR2D1BWP7T"}},
        remove_schematic_ports = {"A1","A2","Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "INVD1BWP7T", layout_cell =
        ↪  "INVD1BWP7T"}},
        remove_schematic_ports = {"I","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "AOI32D1BWP7T", layout_cell =
        ↪  "AOI32D1BWP7T"}},
        remove_schematic_ports = {"A1","A2","A3","B1","B2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "MOAI22D0BWP7T", layout_cell =
        ↪  "MOAI22D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B1","B2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "DFKCNQD1BWP7T", layout_cell =
        ↪  "DFKCNQD1BWP7T"}},
        remove_schematic_ports = {"D","CP","CN","Q"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "XNR2D1BWP7T", layout_cell =
        ↪  "XNR2D1BWP7T"}},
        remove_schematic_ports = {"A1","A2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "DFQD1BWP7T", layout_cell =
        ↪  "DFQD1BWP7T"}},
```

```
        remove_schematic_ports = {"D","CP","Q"}
);
```

Cuadro 19.7: Blackboxes generados por el script para una circuito *ALU*

```
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PVDD1CDG", layout_cell =
        ↪  "PVDD1CDG"}},
        remove_schematic_ports = {"VDD"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PDDW0204SCDG", layout_cell =
        ↪  "PDDW0204SCDG"}},
        remove_schematic_ports = {"I","DS","OEN","PAD","C","PE","IE"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "PVSS1CDG", layout_cell =
        ↪  "PVSS1CDG"}},
        remove_schematic_ports = {"VSS"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "CKMUX2D0BWP7T", layout_cell =
        ↪  "CKMUX2D0BWP7T"}},
        remove_schematic_ports = {"I0","I1","S","Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "OAI31D0BWP7T", layout_cell =
        ↪  "OAI31D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","A3","B","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "MAOI22D0BWP7T", layout_cell =
        ↪  "MAOI22D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B1","B2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "OAI21D0BWP7T", layout_cell =
        ↪  "OAI21D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "CKND0BWP7T", layout_cell =
        ↪  "CKND0BWP7T"}},
        remove_schematic_ports = {"I","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "MUX2ND0BWP7T", layout_cell =
        ↪  "MUX2ND0BWP7T"}},
```

```
        remove_schematic_ports = {"IO","I1","S","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "IOA21D0BWP7T", layout_cell =
        ↪  "IOA21D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "INR2D0BWP7T", layout_cell =
        ↪  "INR2D0BWP7T"}},
        remove_schematic_ports = {"A1","B1","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "XNR2D1BWP7T", layout_cell =
        ↪  "XNR2D1BWP7T"}},
        remove_schematic_ports = {"A1","A2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "CKXOR2D0BWP7T", layout_cell =
        ↪  "CKXOR2D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "XOR3D0BWP7T", layout_cell =
        ↪  "XOR3D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","A3","Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "AOI22D0BWP7T", layout_cell =
        ↪  "AOI22D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B1","B2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "AOI31D0BWP7T", layout_cell =
        ↪  "AOI31D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","A3","B","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "IAO21D0BWP7T", layout_cell =
        ↪  "IAO21D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "NR2D0BWP7T", layout_cell =
        ↪  "NR2D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","ZN"}
);
lvs_black_box_options(
```

```
        equiv_cells ={{schematic_cell = "MOAI22D0BWP7T", layout_cell =
        ↪  "MOAI22D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B1","B2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "XNR3D0BWP7T", layout_cell =
        ↪  "XNR3D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","A3","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "ND3D0BWP7T", layout_cell =
        ↪  "ND3D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","A3","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "CKMUX2D1BWP7T", layout_cell =
        ↪  "CKMUX2D1BWP7T"}},
        remove_schematic_ports = {"I0","I1","S","Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "OAI22D0BWP7T", layout_cell =
        ↪  "OAI22D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B1","B2","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "OAI211D0BWP7T", layout_cell =
        ↪  "OAI211D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B","C","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "AN2D0BWP7T", layout_cell =
        ↪  "AN2D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","Z"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "NR3D0BWP7T", layout_cell =
        ↪  "NR3D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","A3","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "ND4D0BWP7T", layout_cell =
        ↪  "ND4D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","A3","A4","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "CKND2D0BWP7T", layout_cell =
        ↪  "CKND2D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","ZN"}
```

```
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "OAI211D1BWP7T", layout_cell =
        ↪  "OAI211D1BWP7T"}},
        remove_schematic_ports = {"A1","A2","B","C","ZN"}
);
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "AOI221D0BWP7T", layout_cell =
        ↪  "AOI221D0BWP7T"}},
        remove_schematic_ports = {"A1","A2","B1","B2","C","ZN"}
);
```

CAPÍTULO 20

Glosario

BASH: Interfaz de usuario de linea de comando de Bourne Shell mejorada, BASH significa "Bourne Again Shell". v, 6–8, 10, 14, 18, 22, 48

blackbox: Dispositivo de entradas y salidas cuyo funcionamiento interno es desconocido o de carácter despreciable. v, vii, ix, 15, 32, 33, 37, 39, 41, 43, 45, 49, 63–65, 67

cell / celda: Bloque compuesto de transistores interconectados que cumplen una tarea lógica ya sea secuencial o combinacional. . 35

GDS: En ingles las siglas pueden significar: Graphic Design System, Graphic Data Stream o Geometric Data Stream. Este es un formato de archivo utilizado en el intercambio del diseño de un *layout* de un circuito. 18, 24, 35, 37

HDL: Es un lenguaje descritor de hardware, sus siglas significan "Hardware Descriptor Language" en ingles.. 3, 22

ICV: Formato de netlist de IC Validator.. v, viii, 18, 22, 24, 33, 37

IMEC: Interuniversity Microelectronics Centre. 18

nanoescala: Escala en el orden de 10^{-9}. 4

netlist: Descripción de un circuito electrónica en la cual se muestran las componentes del circuito junto con sus nodos e interconexiones.. viii, 2, 22, 33

Shell: Interprete de línea de comandos de Sistemas Operativos basados en Unix. 6, 10