UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Etapa de verificación física de *Diseño en Silicio vs. Esquemático* (LVS) en el flujo de diseño para un chip a nanoescala

Trabajo de graduación presentado por Juan Ricardo Girón Rubio para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2020

Etapa de verificación física de *Diseño en Silicio vs. Esquemático* (LVS) en el flujo de diseño para un chip a nanoescala

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Etapa de verificación física de *Diseño en Silicio vs. Esquemático* (LVS) en el flujo de diseño para un chip a nanoescala

Trabajo de graduación presentado por Juan Ricardo Girón Rubio para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2020

Vo.Bo.:

(f) _____

Ing. Luis Nájera

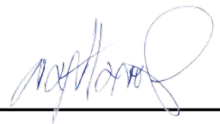
Tribunal Examinador:

(f) _____

Ing. Luis Nájera

(f) _____

MSc. Carlos Esquit



(f) _____

Ing. Jonathan de los Santos

Fecha de aprobación: Guatemala, 20 de enero de 2021.

Prefacio

Agradezco principalmente a mi familia por brindarme la oportunidad de obtener una educación de calidad y darme su apoyo incondicional.

Agradezco también al Ing. Luis Nájera, asesor de mi trabajo de graduación por su apoyo y asesoría en el mismo, a M. Sc. Carlos Esquit por promover esta investigación y brindarnos apoyo con su conocimiento en el campo de la nanoelectrónica. También agradezco al Ing. Jonathan de los Santos por brindarnos su ayuda con la instalación y actualización de herramientas que hicieron posible este proyecto.

Índice

Lista de figuras

Lista de cuadros

Resumen

Este trabajo establece una base sólida de uno de los módulos de verificación física en el flujo de diseño de un chip a nano escala, *Layout Versus Schematic* (LVS).

El proceso de LVS se encarga de verificar la integridad del *layout* de un circuito integrado. El flujo de LVS tiene dos procesos básicos. El primer proceso es el de extracción, en donde se genera un *netlist* extrayendo dispositivos y sus interconexiones de una base de datos de un *layout*. Tras llevar a cabo una síntesis lógica se obtiene un *netlist* del esquemático original. Este *netlist* es traducido por NetTran a un formato de la herramienta de IC Validator de Synopsys para poder ser sometido al proceso de comparación, siendo este el segundo proceso del flujo de LVS.

En el proceso de comparación se llevan a cabo modificaciones globales en los *netlists*, se generan puntos de equivalencia (celdas equivalentes) y posteriormente se comparan. Este toma como punto de comparación este par de celdas equivalentes, compuesto por una celda del *netlist* del *layout* y otra celda del *netlist* del esquemático. Por último, la herramienta de IC Validator genera un conjunto de archivos de resultados, los cuales pueden interpretarse para proceder a toma de decisiones y correción de errores.

Abstract

This work establishes a solid base of one of the physical verification modules in the design flow of a nanoscale chip, Layout Versus Schematic (LVS).

The LVS process is responsible of verifying the integrity of an integrated circuit's layout. The LVS flow has two basic processes. The first one is the extraction process, where a netlist is generated by extracting devices and their interconnections from a database of a layout. After performing a logical synthesis, the product is a netlist from the original circuit representation. This is translated by NetTran into a format of the Synopsys Validator IC tool in order to be subjected to the comparison process, which is the second process of the LVS flow.

In the comparison process, global modifications are made to the netlists, equivalence points (equivalent cells) are generated, and then compared. LVS comparison takes as a reference point this pair of equivalent cells, consisting of a cell of the schematic netlist and another cell of the layout netlist. Finally, the IC Validator tool generates a set of result files, which can be interpreted for decision-making and error correction.

CAPÍTULO 1

Introducción

Un circuito integrado es sometido a un flujo de diseño previo a ser fabricado. El flujo de diseño está dividido en módulos y cada uno de ellos cumple una función vital para que el circuito integrado esté libre de errores y sea posible su fabricación[1]. Parte de estos módulos cumplen la función de verificación física en donde se comprueba la integridad del diseño en silicio de un circuito [2]. Entre estas pruebas de integridad cabe mencionar a *Layout vs. Schematic* (LVS), *Design Rule Check* (DRC), *Electrical Rule Check* (ERC), *Antenna Rule Checking* y *Layout Parasitic Extraction* (LPE) [3] [4].

Probar la integridad del diseño en silicio de un circuito a nanoescala es un tema imprescindible para la funcionalidad correcta de este mismo [4]. Una analogía de estas pruebas es la construcción de un edificio, en donde un ingeniero maneja planos y estos mismos deben respetarse para que la estructura no colapse.

El presente trabajo tiene como objetivo el documentar y desarrollar la verificación física de *Layout vs. Schematic* (LVS). En este mismo se comprueba la integridad del diseño en silicio (*layout*) de un circuito a nanoescala verificando su congruencia con la representación original del circuito en lenguaje descriptivo de hardware (*schematic*).

CAPÍTULO 2

Antecedentes

El curso de estudio de Nanoelectrónica se introdujo a la Universidad del Valle de Guatemala en el año 2013. El Ing. Esquit introdujo e impartió el curso de Introducción a Sistemas de Diseño *VLSI* por primera vez en la universidad. Este curso abordó teoría básica sobre nanoelectrónica. A inicios, se utilizó la herramienta de *Electric VLSI Design System*. Dicha herramienta es gratuita y en esta es posible llevar a cabo procesos básicos de *VLSI*.

En el año 2014 se estableció un acuerdo con Synopsys, una de las compañías más fuertes en el campo de la nanoelectrónica a nivel mundial. Con este acuerdo se tuvo acceso al *University Program*, elaborado por Synopsys, en donde se brindan herramientas y licencias de *software* que permitan el diseño y verificación de un circuito a nanoescala. Gracias a este acuerdo se involucró *VLSI* en el primer trabajo de graduación el cual consistió en el diseño de un sumador de 32 bits con tecnología de 28 nanómetros. Este trabajo puede encontrarse en [5].

En el año 2015 se introdujeron al pénsum de la carrera de ingeniería electrónica los cursos Nanoelectrónica 1 y 2. En estos cursos se tuvo un mayor alcance gracias a las herramientas de Synopsys.

Posteriormente en el año 2019 un grupo de etudiantes enfocó su trabajo de graduación en la elaboración de un flujo de diseño para un chip a escala nanométrica. Este trabajo estableció la base fundamental para el diseño de este tipo de circuitos, ya que al tener un flujo de diseño, se lleva un proceso mucho más organizado y es aplicable a cualquier chip que cumpla cualquier tarea en específico. Estos trabajos pueden encontrarse en [6] [7].

CAPÍTULO 3

Justificación

El diseño de un circuito a nanoescala debe cumplir con requerimientos que son establecidos por parte del fabricante para poder ser sometido a fabricación. El diseño de este mismo pasa por un flujo conocido como *Design Flow* (Flujo de Diseño), en donde pasa por varias etapas que logran el diseño correcto y verificación de integridad del circuito. Grandes compañías de fabricación de circuitos integrados como Intel dividen su trabajo en módulos. Cada módulo está adaptado a una parte del flujo de diseño y grupos de ingenieros en diseño se experimentan en cada parte que les corresponde, tratando con *software* y herramientas increíbles. La integración de estos módulos permite la fabricación de un circuito integrado en silicio sin errores.

Este trabajo tiene el fin de establecer una base sólida para uno de los módulos de verificación física del flujo de diseño, *Layout Versus Schematic* (LVS). Con este se pretende documentar el proceso de verificación de la integridad de la representación física de un circuito integrado (*layout*) y que pueda ser fabricado. Con el fin de despertar el interés en el campo de la nanoelectrónica en Guatemala y abrir nuevas puertas en este campo de estudio.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Definir y documentar en su totalidad el proceso de LVS para la fabricación de un chip a nanoescala.

4.2. Objetivos específicos

- Generar el análisis de LVS de un circuito nanométrico como parte de un flujo de diseño para verificar la integridad del *layout*.
- Obtener el *netlist* del esquemático en el formato adecuado mediante la herramienta de NetTran para poder llevar a cabo la fase de comparación en el flujo del proceso de LVS.
- Establecer un manual del proceso de LVS paso a paso con la herramienta de IC Validator mediante videotutoriales y con manuales escritos para que pueda ser útil de referencia en futuros proyectos.

CAPÍTULO 5

Alcance

Este proyecto busca documentar y detallar el proceso para llevar a cabo la verificación física de LVS de tres maneras distintas con herramientas de Synopsys. Además, se busca seguir una línea de trabajo (*Workflow*) como parte de la metodología con los siguientes puntos:

1. Generar archivo RTL.
2. Obtener *netlist* a nivel compuerta mediante síntesis lógica.
3. Obtener CDL *netlist* de esquemático mediante NetTran *Tool*.
4. Obtener archivo GDS a partir del *layout* del circuito mediante síntesis física.
5. Correr prueba de LVS mediante comandos, VUE *Tool* y Custom Compiler.
6. Interpretación de archivos de salida de LVS.

Cada uno de los puntos de la línea de trabajo se describirá en este trabajo. El proceso se documentará para tres circuitos, una compuerta not, un circuito *Full Adder* y un circuito *Ripple Carry Adder*. Para cada circuito se llevará a cabo LVS de las tres maneras mencionadas en el Punto #5 de la línea de trabajo descrita anteriormente. Las librerías con las que se cuentan presentan ciertas limitaciones a la hora de llevar a cabo LVS debido a que no se cuenta con la descripción completa de celdas. Por esta razón, se realizará un *Black Box* LVS, en donde la comparación de *layout* y esquemático se hace en base a la representación en cajas negras de las celdas que los componen.

Al correr LVS para un circuito varios archivos de salida son generados. Estos archivos contienen información importante a interpretar. Los archivos importantes serán detallados e interpretados para las pruebas que se harán.

Los archivos de entrada para LVS se describirán y se presentará una guía de cómo obtener cada uno de ellos. En este caso, un archivo GDS y un CDL *netlist* de esquemático. Así mismo, se presentarán videotutoriales que describen dichos procesos.

El proceso documentado podrá adaptarse a cualquier circuito, siendo este uno de los objetivos del proyecto general, el establecer un flujo de diseño completo y adaptable.

CAPÍTULO 6

Marco teórico

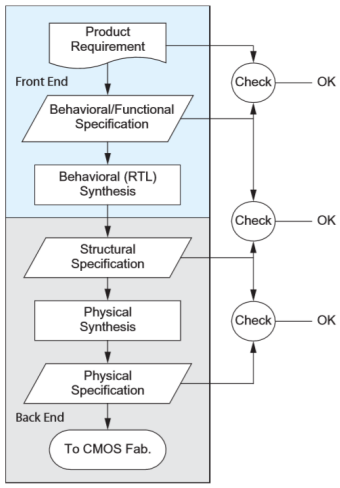


Figura 1: Design Flow

6.1. Flujo de diseño

El Flujo de Diseño o *Design Flow* se refiere a un proceso que permite a los diseñadores llevar de las especificaciones de un chip a la fabricación de este mismo fuera de errores. El diseño comienza evaluando los requerimientos del sistema. Posteriormente establece una descripción del comportamiento lógico del sistema (*Behavioral level*) y procede a una descripción estructural (*Structural level*) antes de ser fabricado. El flujo de diseño se divide en dos etapas: *Front End* y *Back End* asociadas al *behavioral level* y *structural level* respectivamente. En la etapa de *front end* se lleva a cabo una síntesis lógica en donde los diseños se capturan a un nivel *Register Transfer Level* (RTL) mediante HDL. Luego, esta descripción lógica del sistema pasa por una síntesis física obteniendo una descripción física del sistema (*layout*). [1] En la Figura #1 se puede observar un diagrama generalizado del flujo de diseño.

6.1.1. *Front end*

Front end se refiere básicamente a la etapa de diseño en la cual se plantea la función específica del chip en lenguaje descriptivo. Esta parte del flujo de diseño puede observarse en la Figura #4.

El proceso puede resumirse en los siguientes pasos.

1. Verilog/RTL: elaboración del circuito en verilog de forma *structural* o *behavioral*. A este circuito se le conoce como RTL.

2. Síntesis lógica: se utiliza la plataforma Design Vision para la conversión del RTL en un *netlist* de compuertas lógicas.

3. *Netlist*: circuito generado después de la síntesis lógica. Se presenta de manera *structural* y con componentes referenciados en librerías.

4. Verificación: etapa final del *front end flow*, en donde se verifica la funcionalidad de la síntesis del circuito en plataformas como Formality, VCS o bien utilizando una *Nanoboard*.

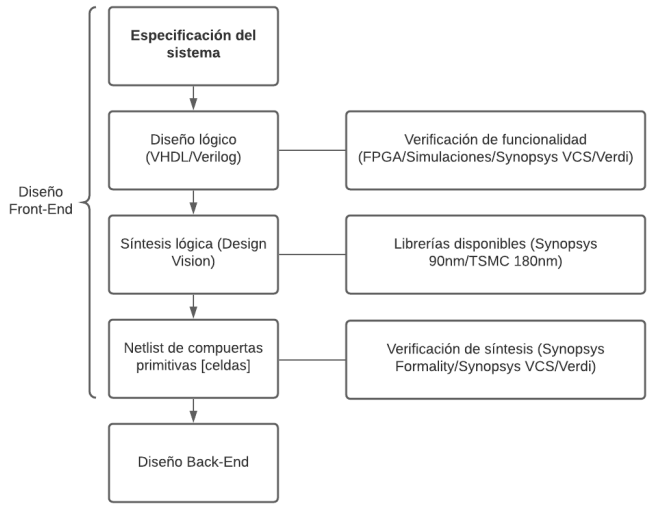


Figura 2: *Front End Design Flow*

6.1.2. *Back end*

Back end se refiere específicamente a la implementación física del circuito. En esta etapa de diseño se transforma el circuito en lenguaje descriptivo o RTL en un diseño físico (*layout*), compuesto por *frames* o celdas. Esta parte del flujo de diseño puede verse en la Figura #3.

El proceso puede resumirse en los siguientes pasos:

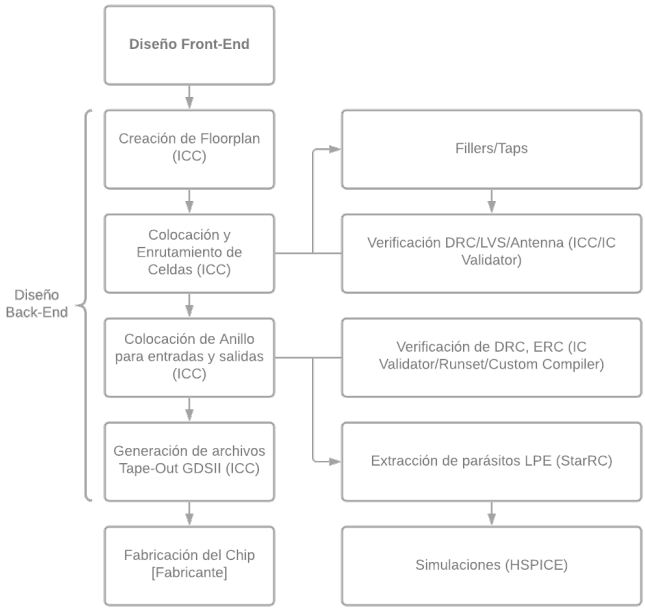


Figura 3: *Back End Design Flow*

1. Síntesis física: basándose en un *netlist* totalmente verificado se obtiene una descripción física del mismo, conocido como *layout*. El *layout* contiene una descripción de la topología del circuito la cual ya es manufacturable.
2. *Placement*: diseño de *floorplan*, en donde se establecen las celdas estratégicamente en una planta.
3. *Routing*: teniendo ya las celdas establecidas estratégicamente se procede a interconectar estas mismas (*routing*).
4. *Layout Versus Schematic* (LVS): verificar la equivalencia de un *layout* con el *netlist* de esquemático.
5. *Parasitic Extraction*: obtención de las propiedades capacitivas y resistivas en cada nodo del *layout*.

6.2. Verificación física

Luego de obtener una representación física del circuito, debe comprobarse la integridad de la misma. Debe verificarse el funcionamiento correcto de la parte eléctrica y de la parte lógica del circuito. Algunos problemas que presenta el circuito pueden ser tolerables, pero en otros casos el *layout* necesita algunos cambios y estos no deben ser muchos para no introducir más problemas. En esta etapa del flujo de diseño los cambios en el *layout* provienen de las siguientes verificaciones:

1. *Design Rule Check* (DRC): verifica que el *layout* no viole ninguna de las restricciones o reglas de diseño impuestas por la tecnología.

2. *Layout vs. Schematic* (LVS): verifica la integridad del diseño. Del *layout*, se obtiene un *netlist* y se compara con el *netlist* de esquemático obtenido por la síntesis lógica. [2]

3. *Parasitic Extraction*: obtiene los parámetros eléctricos de los elementos del *layout* a partir de sus representaciones geométricas. Junto con el *netlist*, estos parámetros verifican las características eléctricas del circuito. [2]

4. *Antenna Rule Checking*: "*previene los efectos de antena, que pueden dañar los gates de los transistores durante los pasos de manufactura mediante la acumulación de excesos de carga en los alambres de metal que no están conectados a los canales PN de los transistores.*"[2]

5. *Electrical Rule Checking* (ERC): verifica que las conexiones de *power* y *ground* sean correctas, los tiempos de transiciones de señales y las cargas capacitivas y los *fanouts* estén limitados apropiadamente. [2]

6.2.1. IC Validator

IC Validator es una herramienta de Synopsys que se utiliza para la validación de la integridad de un circuito basándose en la comparación del diseño en silicio del circuito y el esquemático del mismo. [8]

6.2.2. *Netlist*

Un *netlist* consiste en una representación real del diseño de un circuito electrónico. Se conforma por las instancias de los elementos básicos del circuito, sus interconexiones (*Nets*) y ciertos atributos.

Un RTL establece una descripción del circuito mediante lenguaje descriptivo (HDL). Un *netlist* contiene elementos referenciados a librerías, "*es un paso más cerca a la implementación real del circuito en silicio*". [9]

6.2.3. NetTran

NetTran es una herramienta de traducción de *netlists*. El *netlist* del esquemático se compara con el *netlist* del *layout* en el flujo de LVS. Para poder llevar a cabo esta comparación, el *netlist* del esquemático debe traducirse a un formato de IC Validator. Esto puede verse

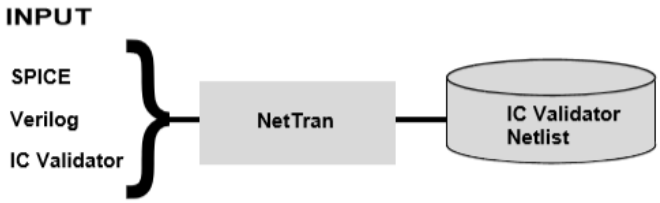


Figura 4: *Netlist translation*

en la Figura #5, antes de iniciar el proceso de comparación los *netlists* deben estar en el formato de IC Validator. Este formato contiene una descripción jerárquica de los dispositivos e interconexiones del esquemático.

Al tener el *netlist* del esquemático en el formato correcto, el proceso de LVS puede llevarse a cabo.

Los *netlists* de SPICE y Verilog pueden traducirse al formato de IC Validator. La unidad geométrica de propiedades de dispositivos en IC Validator es el *Micrón*, para SPICE es el metro."*Los netlists en formato SPICE difieren del formato IC Validator en que en el formato de SPICE se establecen referencias implícitas.*"[8] El orden de los puertos ya está establecido, en cambio en el formato de IC Validator la referencia es explícita ya que se establecen nombres de puertos y el orden es indiferente.

En el caso de formato Verilog, NetTran solo lo soporta de forma *structural*. Un *netlist* en formato de Verilog utiliza módulos para establecer celdas. En este formato se requiere establecer las entradas y salidas del circuito antes de utilizarse. El formato de IC Validator no lo necesita. [8]

6.2.4. *Layout*

El diseño del *layout* es una representación gráfica y esquemática de un circuito integrado en donde se describe la ubicación exacta de componentes para su fabricación. Existen varias reglas de diseño que establecen medidas y separaciones límite de componentes para poder ser empacados durante el proceso de manufacturación. Estas medidas normalmente se especifican en micrones en el área industrial. Esto complica bastante la migración de procesos ya que no en todos los casos las reglas se especifican de la misma manera. [1]

6.2.5. LVS

En el procesamiento de semiconductores se establece una etapa en la que el diseño de un circuito se refleja en un *layout*. Existen herramientas de verificación que permiten asegurar la integridad del diseño del circuito. El *layout* consiste en una representación física del circuito que contiene información importante a considerar para la fabricación correcta del circuito en silicio.

Layout Versus Schematic conocido por sus siglas del inglés como LVS, es una de las fases de verificación física que conforma el flujo de diseño de un chip a nanoescala. Durante la fase de síntesis lógica del flujo de diseño de un circuito a nanoescala se genera un *netlist* de esquemático. LVS es un proceso que se encarga de validar la congruencia y equivalencia del *netlist* del *layout* con el *netlist* obtenido en la síntesis lógica.[9] Herramientas de *software* se encargan de la validación de equivalencia entre ambos *netlists*, una de ellas es IC Validator por parte de Synopsys. La herramienta genera reportes y archivos donde se muestran los errores en caso de haberlos.

6.3. LVS *Flow*

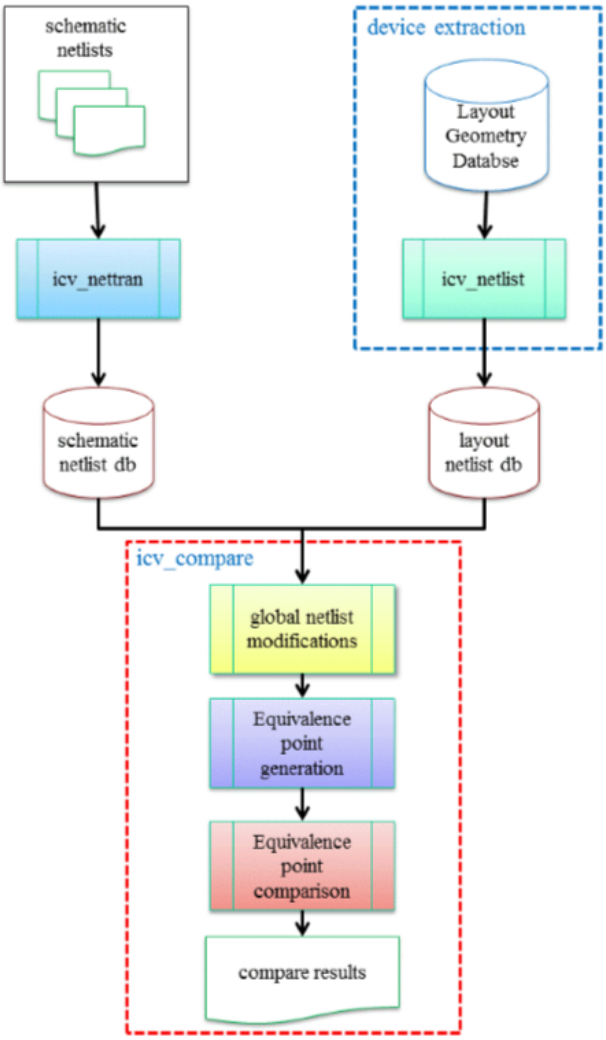


Figura 5: LVS *Flow*

El proceso de LVS pasa por un flujo. Dos procesos básicos conforman a este mismo. El primero es la extracción de *netlist* del *layout*, en donde la herramienta de IC Validator analiza las capas en la base de datos del *layout* y extrae todos los dispositivos y *nets*. En paralelo se traduce el *netlist* del esquemático a un formato de IC Validator. El segundo proceso es el de comparación. En este proceso se lleva a cabo la comparación de ambos *netlists*, el de *layout* y el de esquemático, que este segundo se traduce previamente a un formato de IC Validator mediante NetTran. [8] Este flujo se describe en la Figura #5. El proceso es el siguiente:

1. Extracción de *netlist* del *layout*. De la misma manera que se obtuvo un *netlist* en la síntesis lógica, se obtiene uno que describe al *layout*. "*La forma más simple del circuito extraído debe ser en forma de un netlist, que está en un formato para un simulador o un programa de análisis.*"[10]

2. Se llevan a cabo modificaciones globales en los *netlists*, esto con el fin de acelerar el

proceso de comparación.

3. Se establecen puntos de equivalencia. La herramienta de IC validator establece pares que tienen lógica similar. Estos pares pueden establecerse también directamente en el *Runset*.

4. Por último se lleva a cabo el proceso de comparación.

6.3.1. Modificaciones globales del *netlist*

Dependiendo de la complejidad del circuito, la representación a nivel CMOS de este mismo puede contener elementos en serie, paralelo o puede contener conexiones que pueden simplificarse. Para llevar a cabo una comparación eficiente y rápida de pares equivalentes en el flujo de LVS, la herramienta de IC Validator lleva a cabo ciertas modificaciones al circuito en donde la representación del mismo se simplifica al máximo. Además, la herramienta lleva a cabo filtrado y combinación de dispositivos. [8]

6.3.2. Filtrado de dispositivos

La herramienta de IC Validator lleva a cabo un filtrado de dispositivos en donde se remueven todos aquellos que no necesitan compararse. [8]

6.3.3. Combinación de dispositivos

La combinación de dispositivos reduce al máximo la configuración de los mismos. Este proceso de combinación toma en cuenta dispositivos en serie y en paralelo. Es un proceso cíclico e iterativo que alterna combinación en paralelo y combinación en serie. [8] El proceso no toma en cuenta los *nets* de *power* y *ground*, y no termina hasta simplificar la configuración de los dispositivos al máximo.

En la Figura #6 puede observarse una configuración en paralelo. Puede observarse que los pines están interconectados, por esto mismo se genera un dispositivo equivalente.

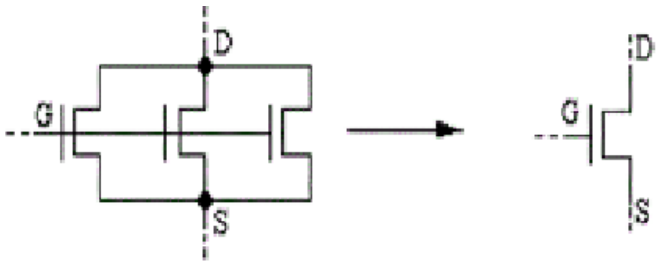


Figura 6: Combinación en paralelo

En la Figura #7 puede observarse una configuración en serie. Además, los pines de *drain* y *source* están conectados secuencialmente, esto implica una conexión en serie. Sin embargo,

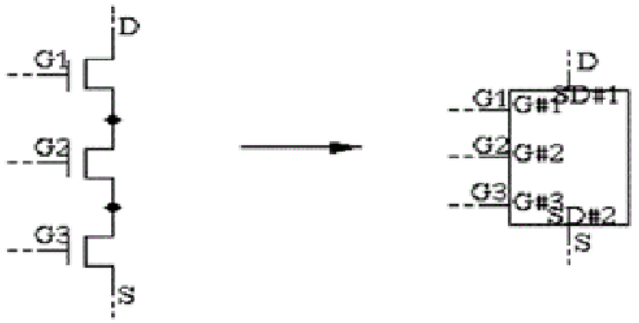


Figura 7: Combinación en serie

estas conexiones no pueden tener otra conexión. Se genera un dispositivo con multiples pines de *gate* (cantidad de dispositivos en la cadena).

6.3.4. Generación de puntos de equivalencia

La generación de estos puntos de equivalencia se basa en pares de celdas que cuentan con lógica potencialmente similar. El par consta de una celda del esquemático y otra del *layout*. A este par se le conoce como celdas equivalentes.

Las celdas equivalentes pueden establecerse directamente en el *runset* o bien son generados por la herramienta de IC Validator. Sin embargo, mientras mejor se establezcan estas celdas equivalentes el proceso de comparación será más rápido. [8]

6.3.5. Comparación de puntos equivalentes

Teniendo ya establecidos los puntos de equivalencia (celdas equivalentes) la herramienta de IC Validator lleva a cabo una comparación jerárquica, empezando desde abajo y terminando hasta arriba.

6.3.6. Resultados de comparación

Al finalizar la comparación de los *netlists*, la herramienta de IC Validator genera varios archivos. Se genera un resumen del proceso de comparación. [8] Este resumen contiene:

1. Resultados finales, pasa la prueva de LVS o no (*PASS/FAIL*).
2. Reporta el número de celdas equivalentes que pasaron la prueba y las que no (*passed/failed*).
3. Mensajes de compilación.
4. Diagnósticos.

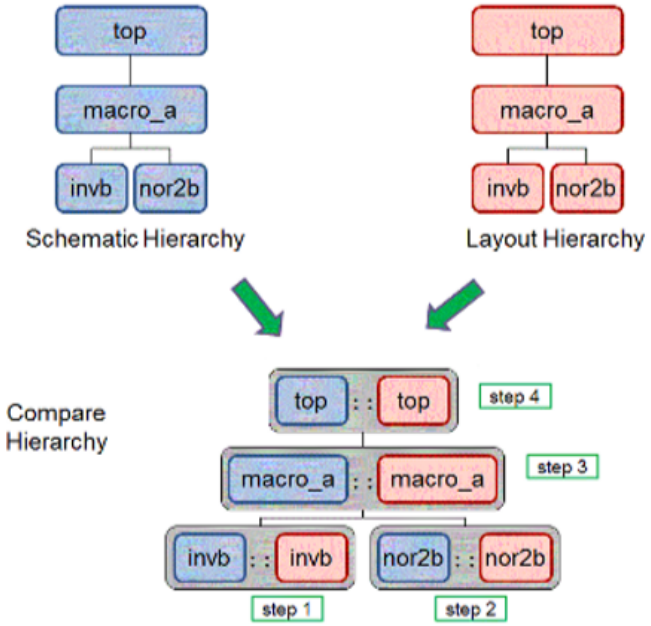


Figura 8: Comparación

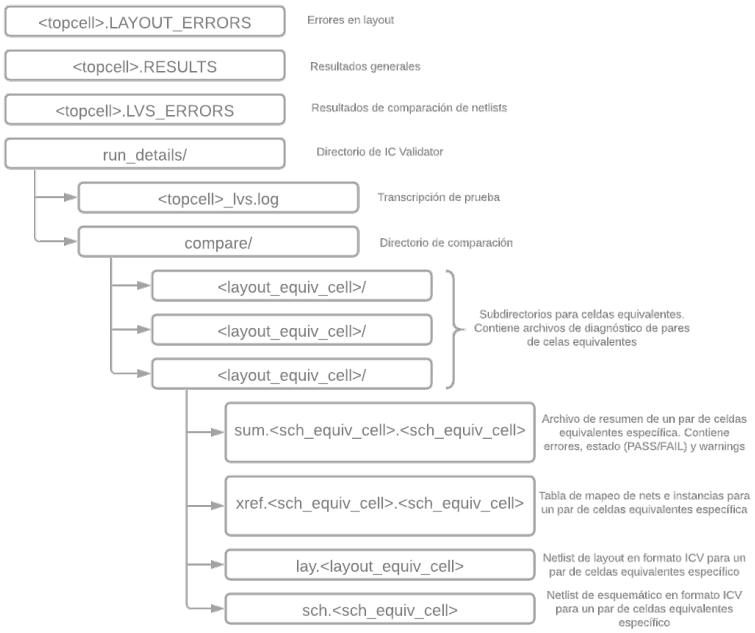


Figura 9: Archivos de salida proceso de LVS

En el Cuadro #1 pueden observarse los mensajes que se pueden obtener en los resultados.

La carpeta de *run details* contiene un subdirectorio por cada par de celdas equivalentes como puede observarse en la Figura #9. Cada subdirectorio de estos contiene cuatro archivos:

Categoría	**Mensaje**
Net mismatches	N *net(s)* potencialmente cortocicuitadas en el *layout* N *net(s)* potencialmente abiertas en el *layout* N errores de conexiones incorrectas N *nets* faltantes
Instance mismatches	N dispositivo(s) faltantes en el *layout* N dispositivo(s) extras en el *layout*
Property errors	N dispositivo(s) tienen discrepancia en propiedades

Cuadro 1: Mensajes en reportes

1. sum: el resumen de la comparación para el par de celdas equivalentes.
2. xref: tabla de mapeo de los *nets* e instancias equivalentes en cada par de celdas equivalentes.
3. lay: *netlist* del *layout* en formato de IC Validator para el par de celdas equivalentes.
4. sch: *netlist* del esquemático en formato de IC Validator para el par de celdas equivalentes.

6.4. CDL

Circuit Design Language mejor conocido por sus siglas del inglés CDL. Este es un tipo de *netlist* que describe a un circuito. Es utilizado en el proceso de *Layout vs. Schematic* (LVS) y representa al lado del esquemático. Este es un formato de *netlist* similar a SPICE. Se diferencia con SPICE en que un `.GLOBAL` declara *power*, *ground* y también el relój. Además, contiene celdas vacías representadas por instancias de subcircuitos. [11]

6.5. GDS

Graphic Data System mejor conocido por sus siglas del ingles GDS. Este es un archivo generado por herramientas de CAD para circuitos integrados. Este mismo contiene información del *layout*. Es un formato binario y no legible que contiene figuras geométricas, *labels* y otra información del *layout* de forma jerárquica. El formato es GDSII y normalmente el archivo se utiliza para compartir el diseño en silicio del circuito. En LVS este mismo representa a la parte del *layout* durante la comparación. La herramienta de IC Validator extrae un *netlist* del archivo GDS del diseño y este mismo es el que se somete a comparación al correr LVS.

6.6. Custom Compiler

Custom Compiler es una herramienta de Synopsys que permite la simulación, análisis y diseño de circuitos integrados. Esta es una herramienta que permite llevar a cabo la verifi-

cación de *Layout vs. Schematic* (LVS) mediante interfaz gráfica. Esta herramienta presenta los resultados de una manera bastante amigable y organizada.[12]

6.7. PDK

La librería *Process Design Kit*, o mejor conocida por sus siglas del inglés como PDK, es un conjunto de archivos que se utiliza en la industria de semiconductores para poder modelar procesos de fabricación en herramientas de diseño dedicadas a circuitos integrados. Esta librería es creada por el fabricante con variaciones de tecnología. La librería PDK se entrega a sus clientes para que estos mismos puedan utilizarla durante el flujo de diseño. Prácticamente se utiliza para diseñar, dibujar, simular y verificar los diseños antes de enviarlo al fabricante. Mientras más precisa sea la librería, mayor será la posibilidad de tener un diseño libre de errores antes de ser sometido a fabricación.

CAPÍTULO 7

LVS *Workflow*

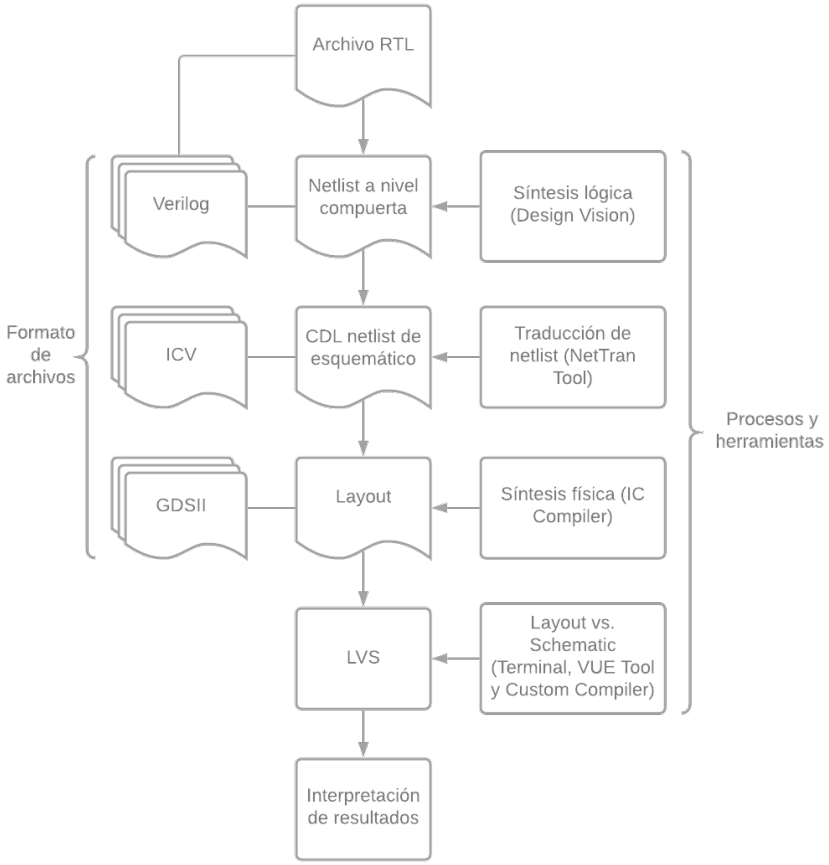


Figura 10: LVS Workflow

En la Figura #10 se puede observar la línea de trabajo para poder lograr la verificación física de LVS. Esto es muy importante tenerlo claro ya que en el proceso se van generando archivos necesarios para poder llevar a cabo la verificación y se utilizan diferentes herramientas para generarlos.

Se puede observar que del lado izquierdo se presentan los formatos de los archivos y del lado derecho los procesos necesarios y las herramientas determinadas para poder obtener dichos archivos. La línea de trabajo comienza con un archivo RTL. Por ahora, debe fijarse la atención en este archivo y seguir la línea en donde el proceso puede interpretarse como:

1. Archivo RTL: este es el archivo primitivo que contiene el circuito original, se utiliza Verilog.
2. Netlist a nivel compuerta: este archivo es una de las salidas de la síntesis lógica en el flujo de diseño y se obtiene un netlist con referencias a librerías, en este caso el formato es en Verilog.
3. Netlist a nivel transistor: las verificaciones físicas se llevan a cabo a nivel transistor por lo que el netlist a nivel compuerta no es suficiente. Para obtenerlo se utiliza la herramienta de NetTran.
4. Layout: es la representación en silicio del circuito, se obtiene mediante la síntesis física en el flujo de diseño y se genera un archivo en formato GDSII.
5. LVS: en este punto se pueden llevar a cabo verificaciones físicas tales como LVS, ERC y LPE.

Esta imagen representa el proceso a seguir para poder llevar a cabo la verificación física de LVS. Teniendo ya claro el proceso en los siguientes capítulos se describe detalladamente cada punto de la línea de trabajo como también se presentan pruebas importantes y que detallan el uso de las herramientas.

Por cada uno de los capítulos siguientes se describe una de las fases en la línea de trabajo. En cada fase se generan ejemplos los cuales seguirán la línea mientras se pase de capítulo. Esto para mantener el orden del proceso y que en cada fase se detalle de manera correcta y ordenada cada punto importante.

CAPÍTULO 8

Archivo RTL (Fase 1)

El inicio de la línea de trabajo es con un archivo RTL. Este se refiere a la elaboración del circuito en verilog de forma estructural o behavioral. En esta fase simplemente se genera el RTL para el circuito al que se le realizará la verificación de LVS.

Se tomaron tres circuitos como base. Un ejemplo de nivel básico en donde se aplica todo el proceso a una compuerta Not, por otro lado dos ejemplos de nivel medio, un circuito *Full Adder* y un *Ripple Carry Adder*.

8.1. Ejemplo de nivel básico: compuerta Not

```
module Not(A, Y) ;
input A;
output Y;
assign Y = ~A;
endmodule
```

8.2. Ejemplo de nivel medio: *Full Adder*

```
module fulladder(X1, X2, Cin, s, Cout);
input X1, X2, Cin;
output s, Cout;
wire a1, a2, a3;

    xor u1(a1,X1,X2);
    and u2(a2,X1,X2);
```

```
    and u3(a3,a1,Cin);
    or u4(Cout,a2,a3);
    xor u5(S,a1,Cin);

endmodule
```

8.3. Ejemplo de nivel medio: *Ripple Carry Adder*

```
module RCA(X, Y, S, Co);
input [3:0] X, Y;
output [3:0] S;
output Co;
wire w1, w2, w3;

    fulladder u1(X[0], Y[0], 1'b0, S[0], w1);
    fulladder u2(X[1], Y[1], w1, S[1], w2);
    fulladder u3(X[2], Y[2], w2, S[2], w3);
    fulladder u1(X[3], Y[3], w3, S[3], Co);

endmodule

module fulladder(X1, X2, Cin, s, Cout);
input X1, X2, Cin;
output s, Cout;
wire a1, a2, a3;

    xor u1(a1,X1,X2);
    and u2(a2,X1,X2);
    and u3(a3,a1,Cin);
    or u4(Cout,a2,a3);
    xor u5(S,a1,Cin);

endmodule
```

CAPÍTULO 9

Netlist a nivel compuerta (Fase 2)

El netlist a nivel compuerta es uno de los archivos importantes a generar. Empezando con un archivo RTL, se genera un *netlist* con referencias a librerías mediante la síntesis lógica del circuito. Es sumamente importante tener en cuenta que este archivo no es el archivo de entrada en el flujo del proceso de LVS. Esto porque la representación del circuito está a nivel compuerta y el *runset* con el que se lleva a cabo la verificación está a nivel transistor. Por esta razón el *netlist* obtenido mediante síntesis lógica no es suficiente para llevar a cabo la verificación. El *netlist* que se obtiene tras la síntesis lógica pasa por una conversión mediante la herramienta de NetTran. Primero para obtener el nivel de representación necesario o compatible al *runset* y segundo para obtenerlo en el formato necesario para poder someterlo a comparación en la herramienta de IC Validator. Este proceso de conversión se detallará más adelante, por ahora es necesario concentrarse en obtener los archivos necesarios a partir de la síntesis lógica.

Este trabajo no está dedicado a la fase de síntesis lógica en el flujo de diseño de un chip a escala nanométrica. Por esta razón, el proceso no se detallará pero puede revisarse a detalle y detenidamente en [6] en caso de querer obtener más información.

Como se mencionó anteriormente, el flujo de LVS toma como archivos de entrada dos *netlists* que serán el punto de partida, uno del esquemático y otro del layout. El *netlist* del esquemático debe estar en forma estructural y este se compone de elementos referenciados en librerías. Se solicita de forma estructural ya que la herramienta de NetTran solo soporta este formato.

9.1. Prueba de nivel básico: Compuerta Not

A partir de la sístesis lógica del circuito se obtienen 3 archivos:

1. (.sdc)
2. (.ddc)
3. (.v)

Los archivos que serán útiles para LVS son el (.v) que este es el circuito sintetizado y el (.sdc) que nos servirá más adelante en la síntesis física. El destino y nombre de estos archivos se establece en el *runset* de *Design Vision* para la síntesis lógica.

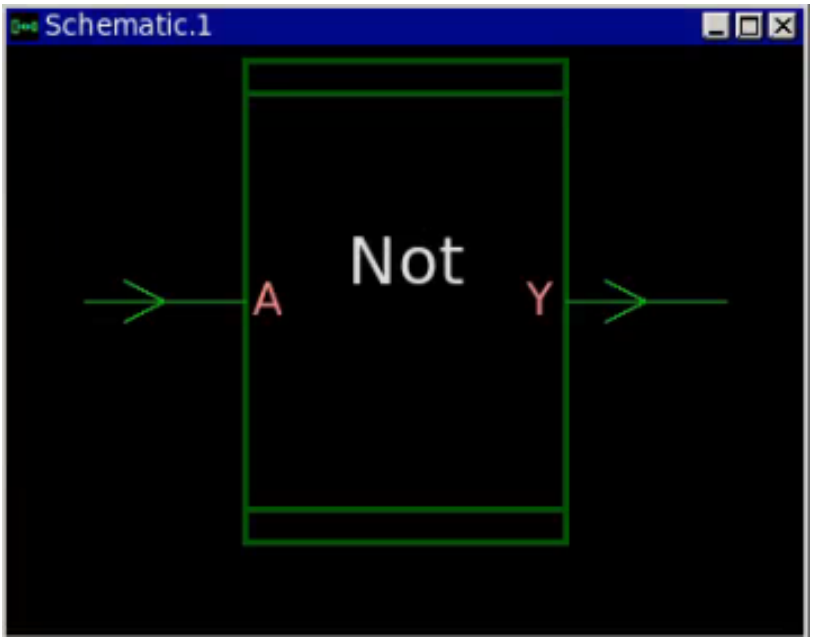


Figura 11: Representación en caja negra de compuerta Not en *Design Vision*

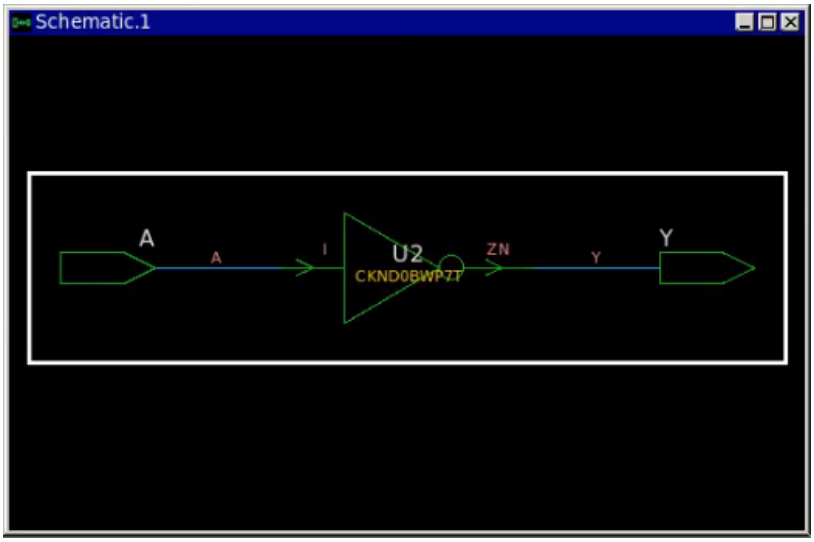


Figura 12: Vista en esquemático de compuerta Not en *Design Vision*

Puede observarse en la Figura #11 y Figura #12 el esquemático y la representación en caja negra de la compuerta Not. Para esto simplemente en *Design Vision* seleccionamos la

Figura 13: Archivos generados con síntesis lógica en *Design Vision* para compuerta not

vista de esquemático (*Schematic View*). Además en la Figura #13 se observan los archivos generados en el proceso de la síntesis lógica.

9.2. Prueba de nivel medio: Full Adder

Figura 14: Representación en caja negra de *Full Adder* en *Design Vision*

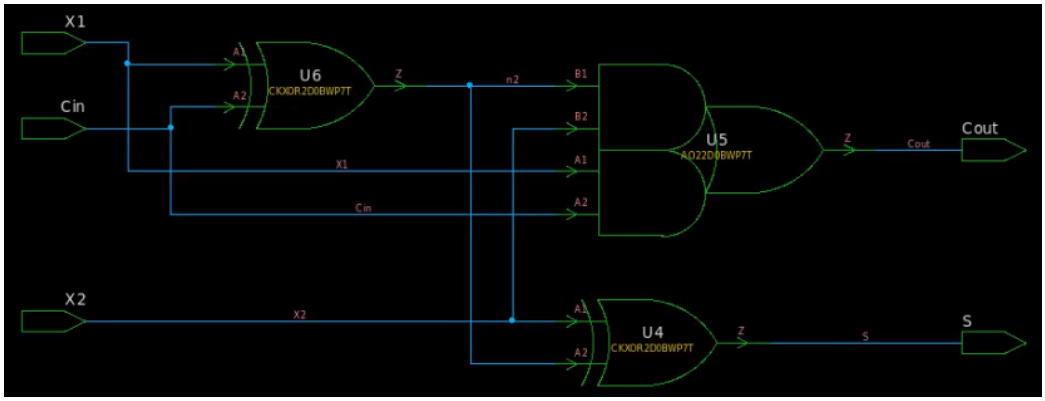


Figura 15: Vista en esquemático de *Full Adder* en *Design Vision*

En la Figura #14 se muestra la representación en caja negra para un *Full Adder*. Además puede observarse también en la Figura #15 el esquemático del circuito con los elementos referenciados en librerías. Por último en la Figura #16 pueden observarse los archivos generados con *Design Vision*.

Figura 16: Archivos generados con síntesis lógica en *Design Vision* para circuito *Full Adder*

9.3. Prueba de nivel medio: Ripple Carry Adder

Figura 17: Representación en caja negra de *Ripple Carry Adder* en *Design Vision*

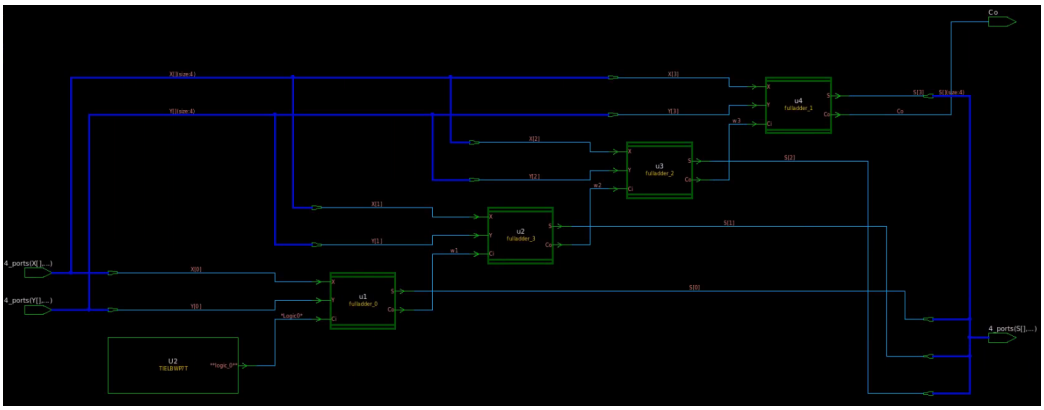


Figura 18: Vista en esquemático de *Ripple Carry Adder* en *Design Vision*

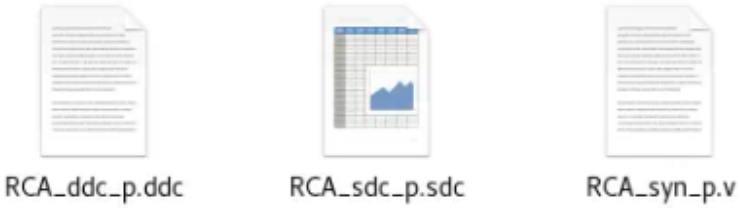


Figura 19: Archivos generados con síntesis lógica en *Design Vision* para circuito *Ripple Carry Adder*

En la Figura #17 se muestra la representación en caja negra para un *Ripple Carry Adder*. Además puede observarse también en la Figura #18 el esquemático del circuito con los elementos referenciados en librerías. Por último en la Figura #19 pueden observarse los archivos generados con *Design Vision*.

CAPÍTULO 10

CDL *netlist* de esquemático (Fase 3)

10.1. NetTran

NetTran es la herramienta que permite traducir diferentes formatos de archivos. La herramienta de IC Validator necesita que ambos *netlists* (*layout* y esquemático) estén en formato ICV para que puedan someterse a comparación. Este formato contiene una descripción jerárquica de los dispositivos e interconexiones del esquemático.

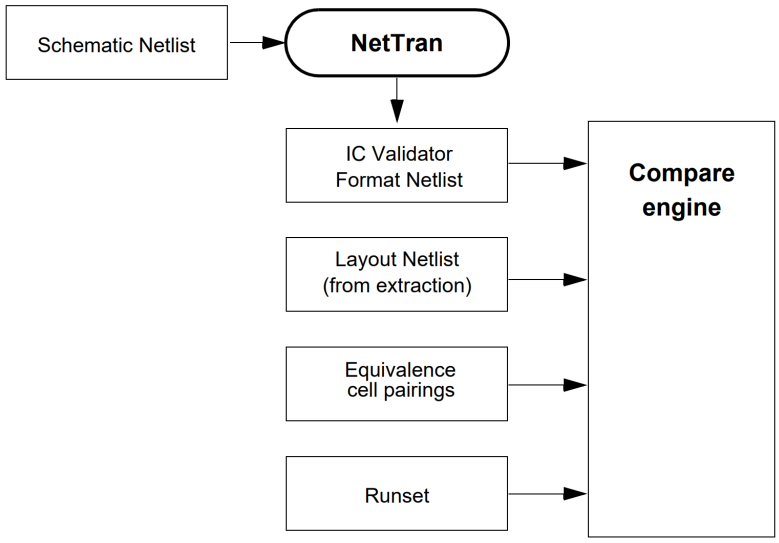


Figura 20: Proceso de comparación

Puede observarse en la Figura #20 que el *netlist* del esquemático es traducido previo a ser sometido a comparación. En el caso del *netlist* del *layout* esta traducción no se lleva a

cabo manualmente ya que el mismo *runset* hace esta traducción al generar el *netlist* en el proceso de extracción en el flujo de proceso de LVS. Cuando ambos *netlists* se encuentran en el formato ICV, pueden someterse a comparación con un *runset*.

Netlists en formato SPICE y Verilog pueden traducirse con NetTran. El formato SPICE se diferencia del ICV en que en SPICE se utiliza la referencia implícita en donde el orden de puertos es fijo. Por otro lado, en el formato ICV, la referencia es explicita ya que el nombre de los puertos está establecido y por esta razón, pueden existir en cualquier orden.

Es importante tener en cuenta que la herramienta de NetTran solo soporta los Verilog en forma *structural* y no *behavioral*. Un *netlist* de Verilog utiliza módulos para la definición de celdas. También es muy importante saber que se requiere que cada puerto y cada *net* se defina previamente como *input*, *output*, *inout*, o *wire*.

Esta herramienta es bastante sencilla de utilizar. Para traducir archivos simplemente se corre un comando con distintos parámetros que nos permiten llevar a cabo la traducción.

10.1.1. Traducción de Verilog

A continuación se presenta un ejemplo de traducción de *netlist* en Verilog a formato ICV. Para poder hacer uso de la herramienta de NetTran basta con abrir una ventana de comandos y escribir el siguiente comando adaptando los parámetros a lo que se desea obtener.

Para traducir un *netlist* de Verilog a ICV debe adaptarse el siguiente comando:

```
icv_nettran -verilog netlist_name.v -outType ICV -outName output_name.icv
```

Para traducir un *netlist* de Verilog a SPICE adapte el siguiente comando:

```
icv_nettran -verilog netlist_name.v -outType SPICE -outName output_name.sp
```

Se recomienda que para la traducción de *netlists* se cree una nueva carpeta y se ejecute una terminal desde la misma para llevar a cabo la traducción. Esto ya que por *default* la herramienta genera el archivo de salida y lo guarda en `\home\administrador\` del sistema.

Compuerta Not (Verilog a ICV)

A continuación se muestra el *netlist* a nivel compuerta obtenido mediante síntesis lógica con librerías de TSMC para una compuerta Not. Para este caso la traducción se hace de un archivo Verilog a un formato ICV.

```
module Not ( A, Y );
  input A;
  output Y;


  CKND0BWP7T U1 (  .I(A), .ZN(Y) );
endmodule
```

```
module Not_IO ( A, Y );
  input A;
  output Y;
  wire   A_w, Y_w, n1, n2;
  tri   A;
  tri   Y;

  Not Com ( .A(A_w), .Y(Y_w) );
  PDDW0204SCDG EN1 ( .I(n1), .OEN(n2), .IE(n2), .PAD(A), .DS(n1),
      .PE(n1), .C(
        A_w) );
  PDDW0204SCDG SAL1 ( .I(Y_w), .OEN(n1), .IE(n1), .PAD(Y), .DS(n1)
      , .PE(n1) );
  TIELBWP7T U6 ( .ZN(n1) );
  TIEHBWP7T U7 ( .Z(n2) );
endmodule
```

A continuación puede observarse el *netlist* en formato ICV.

```
{net_global }
{cell Not
{port A Y}
{inst U1=CKND0BWP7T {TYPE CELL}
{pin A=I Y=ZN}}
}

{cell Not_IO
{port A Y}
{inst Com=Not {TYPE CELL}
{pin A_w=A Y_w=Y}}
{inst EN1=PDDW0204SCDG {TYPE CELL}
{pin n1=I n2=OEN n2=IE A=PAD n1=DS n1=PE
 A_w=C}}
{inst SAL1=PDDW0204SCDG {TYPE CELL}
{pin Y_w=I n1=OEN n1=IE Y=PAD n1=DS n1=PE}}
{inst U6=TIELBWP7T {TYPE CELL}
{pin n1=ZN}}
{inst U7=TIEHBWP7T {TYPE CELL}
{pin n2=Z}}
}

}
```

Puede observarse que el archivo ICV es una descripción jerárquica del circuito. Las instancias de los módulos quedan como celdas (*cell* en código). También puede observarse la instancia de los dispositivos referenciados en librerías, y a cada una le prosigue la declaración de sus *pins*. Prácticamente es el mismo archivo pero este formato es el que nos permitirá someter el *netlist* a comparación. Este *netlist* no es el que se utilizará para LVS, esta tra-

ducción se llevó a cabo solo para comprender un poco más acerca de este formato ICV que se ha mencionado varias veces. Más adelante se detallará acerca del *netlist* de esquemático que será una entrada en el flujo del proceso de LVS.

Compuerta Not (Verilog a SPICE)

A continuación se muestra el *netlist* a nivel compuerta obtenido mediante síntesis lógica con librerías de TSMC para una compuerta Not. Para este caso la traducción se hace de un archivo Verilog a un formato SPICE.

```
module Not ( A, Y );
  input A;
  output Y;


  CKND0BWP7T U1 ( .I(A), .ZN(Y) );
endmodule

module Not_IO ( A, Y );
  input A;
  output Y;
  wire  A_w, Y_w, n1, n2;
  tri   A;
  tri   Y;

  Not Com ( .A(A_w), .Y(Y_w) );
  PDDW0204SCDG EN1 ( .I(n1), .OEN(n2), .IE(n2), .PAD(A), .DS(n1),
      .PE(n1), .C(
        A_w) );
  PDDW0204SCDG SAL1 ( .I(Y_w), .OEN(n1), .IE(n1), .PAD(Y), .DS(n1)
      , .PE(n1) );
  TIELBWP7T U6 ( .ZN(n1) );
  TIEHBWP7T U7 ( .Z(n2) );
endmodule
```

A continuación puede observarse el *netlist* en formato SPICE.

```
.SUBCKT Not A Y
XU1 A Y CKND0BWP7T
.ENDS

.SUBCKT Not_IO A Y
XCom A_w Y_w Not
XEN1 n1 n2 n2 A n1 n1 A_w PDDW0204SCDG
XSAL1 Y_w n1 n1 Y n1 n1 PDDW0204SCDG
XU6 n1 TIELBWP7T
XU7 n2 TIEHBWP7T
.ENDS
```

Puede observarse que el tamaño del archivo se reduce considerablemente en este formato. Las instancias se mantienen y así mismo los puertos de cada una. Las celdas en este caso se declaran con SUBCKT que hace referencia a un *subcircuit*.

10.2. *Netlist* de esquemático

En esta sección se detalla el proceso de cómo obtener el *netlist* de esquemático correcto para poder llevar a cabo la verificación de LVS. Este archivo sí es el que se utilizará como entrada en el flujo de LVS. En la literatura hacen referencia a este *netlist* como CDL *netlist*.

Si se observa la Figura #5 en un lado del flujo de LVS tenemos la parte del esquemático. Puede observarse en esta misma imagen que la entrada por parte del esquemático es un *netlist*. Este *netlist* es el que se obtuvo mediante la síntesis lógica y este mismo está a nivel compuerta.

El fabricante no puede brindarnos un CDL *netlist* a nivel transistor debido a que el acuerdo que se tiene con la universidad no lo cubre. Por esto mismo se tiene que crear directamente el CDL *netlist* con las librerías que fueron brindadas. Estas librerías son la *Standard Cell Library* y la *I/O Cell Library*. Básicamente el CDL *netlist* contiene el circuito que se está trabajando y como este mismo hace referencia a estas librerías los subcircuitos de cada celda referenciada se incluyen en este mismo *netlist*. Entonces, básicamente la diferencia entre el netlist que se obtuvo mediante la síntesis lógica y el CDL *netlist* es que este último cuenta con los subcircuitos de cada celda referenciada.

Debido a que el fabricante no puede brindarnos estas librerías a nivel transistor, la verificación de LVS deberá llevarse a cabo mediante *Black Boxes*. Esto se refiere a que la comparación de *netlists* se hará en base a cajas negras que describen a cada celda. Por esto mismo, el contenido en cada celda es omitido y lo único que es sometido a comparación en LVS son las instancias de cada celda y sus puertos.

Para obtener el CDL *netlist* se necesitan los siguientes archivos:

1. *Netlist* en formato Verilog o SPICE del circuito.
2. *Netlist* en formato Verilog de *Standard Cell Library* (en caso se haya utilizado).
3. *Netlist* en formato Verilog de *I/O Cell Library*.

Para poder obtener el CDL *netlist* a partir de un *netlist* en formato Verilog deben seguirse los pasos que se presentan a continuación.

10.2.1. Paso 1: Extracción de *headers* en *netlists* de la *Standard Cell Library* y de la *I/O Cell Library*

Como primer punto es importante copiar estas dos librerías en dos archivos distintos (uno por librería), por ejemplo la *Standard Cell Library* se copia en un nuevo archivo **CORE.v**

y la *I/O Cell Library* se copia en un nuevo archivo **IO.v** para llevar a cabo el siguiente proceso.

En estas librerías se encuentran módulos como el descrito a continuación:

```
...
`celldefine
module AN4D9 ( I1 , I2 , Z );
  input I1 , I2 ;
  output Z;

  and          ( Z, I1 , I2 );

  specify
    (A1 => Z)=(1, 2);
    (A2 => Z)=(3, 4);
  endspecify

endmodule
`endcelldefine
...
```

El extraer los *headers* de estas librerías se refiere a eliminar todo lo que está definido exceptuando las definiciones de ***input***, ***output*** y también ***inout*** y la declaración del módulo, ***module***. También debe eliminar las definiciones de celdas (*celldefine*).

Para la celda presentada anteriormente la extracción de *headers* sería como se muestra a continuación:

```
...
module AN4D9 ( I1 , I2 , Z );
  input I1 , I2 ;
  output Z;
endmodule
...
```

Estas librerías cuentan con la definición de varios módulos, este proceso de extracción de *headers* es necesario para cada uno de ellos.

10.2.2. Paso 2: Concatenación de *headers*

Los archivos generados en el paso anterior (**CORE.v y IO.v**) contienen todos los *headers* de las librerías utilizadas. En este paso se juntarán ambos archivos para obtener uno solo con la definición de todos los *headers* de ambas librerías. Esta concatenación se lleva a cabo con el siguiente comando:

```
cat CORE.v IO.v> headers.v
```

El archivo que contiene todos los *headers* de ambas librerías es al que se le llama **hea-**

ders.v y este se utilizará más adelante.

10.2.3. Paso 3: Traducción de *headers* a CDL (SPICE)

Para traducir el archivo de **headers.v** obtenido anteriormente a formato CDL (SPICE) debe adaptarse al comando que se presenta a continuación:

```
icv_nettran -verilog netlist_name.v -outType SPICE -outName output_name.sp
```

Debe adaptarse el comando anterior con el archivo que contiene todos los *headers* de las celdas de ambas librerías mencionadas anteriormente. Acá se obtendrá el CDL con todas las referencias a las librerías. Este archivo puede llamarse **libraries.sp**.

Posteriormente, el archivo CDL (SPICE) al que se le llamó como **libraries.sp** debe contener lo siguiente al inicio del mismo:

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

El último archivo es brindado por TSMC. Este mismo contiene la declaración de los subcircuitos de los dispositivos como transistores, diodos, capacitores y resistencias definidos en el runset. Es sumamente importante incluirlo, al igual que lo mencionado anteriormente.

10.2.4. Paso 4: Unión de CDL de librerías con *netlist*

En el paso anterior se obtuvo el CDL de las librerías *I/O Cell Library* y *Standard Cell Library*. Ahora, es necesario juntar este CDL con el *netlist* original y convertirlo al mismo tiempo al formato ICV, que como ya se ha mencionado varias veces, es el formato que utiliza la herramienta de IC Validator para llevar a cabo la comparación de *netlists*.

```
icv_nettran -verilog netlist_sintesis_logica.v -sp libraries.sp -outType ICV
-outName CDL_netlist.icv
```

El archivo que se obtiene en este último comando es el CDL *netlist* o *netlist* de esquemático que se utilizará para LVS. Si en dado caso se presentan dudas del proceso chequear el *Application Note* de Imec titulado *HOW TO CREATE A CDL NETLIST FROM VERILOG* [13].

10.3. Generación de esquemático a partir de un netlist en formato CDL en Custom Compiler

Para generar el esquemático a partir de un *netlist* en formato CDL en la herramienta de Custom Compiler hay que dirigirse a la ventana de *Library Manager* y seleccionar `File -> Import -> Schematic from Netlist...` como se muestra en la Figura # 21 y deberá aparecer una ventana como la que se muestra en la Figura # 22.

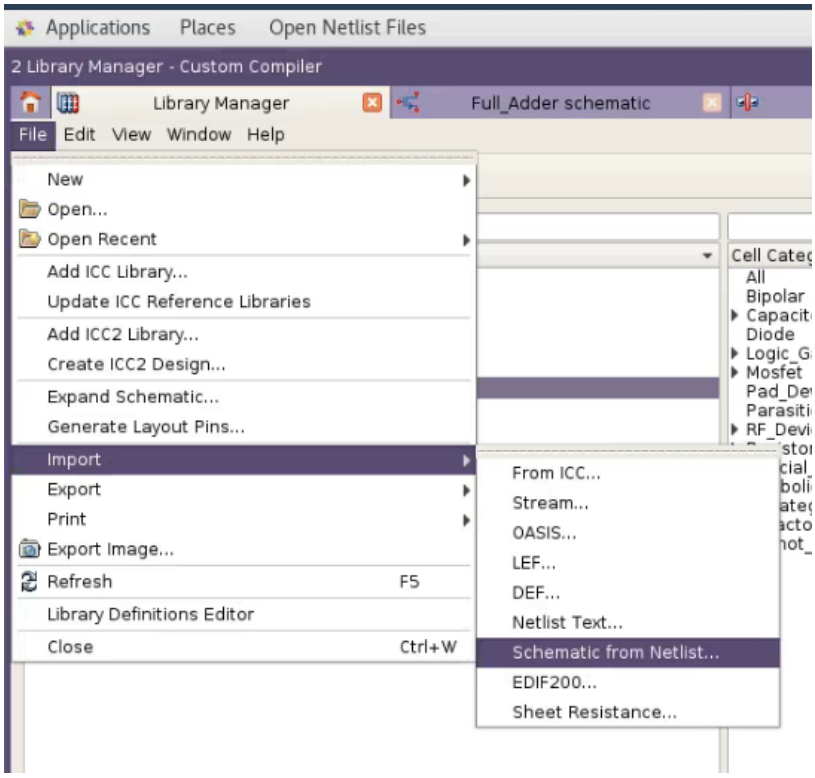


Figura 21: Opción Schematic from Netlist...

En el apartado de *Language* debe seleccionarse CDL. En el paso 3 de la sección anterior debe obtenerse el CDL en formato SPICE. Para esto solo debe cambiarse el formato de salida en el parámetro de *-outType* a SPICE. En el apartado de *Top Cell Name*, debe establecerse el nombre correcto de la celda a la que quiere añadirse el esquemático. Por último, en los apartados de *Library* y *View Name* deben seleccionarse la librería en donde se encuentra la celda y *schematic* respectivamente ya que se estará importando el esquemático.

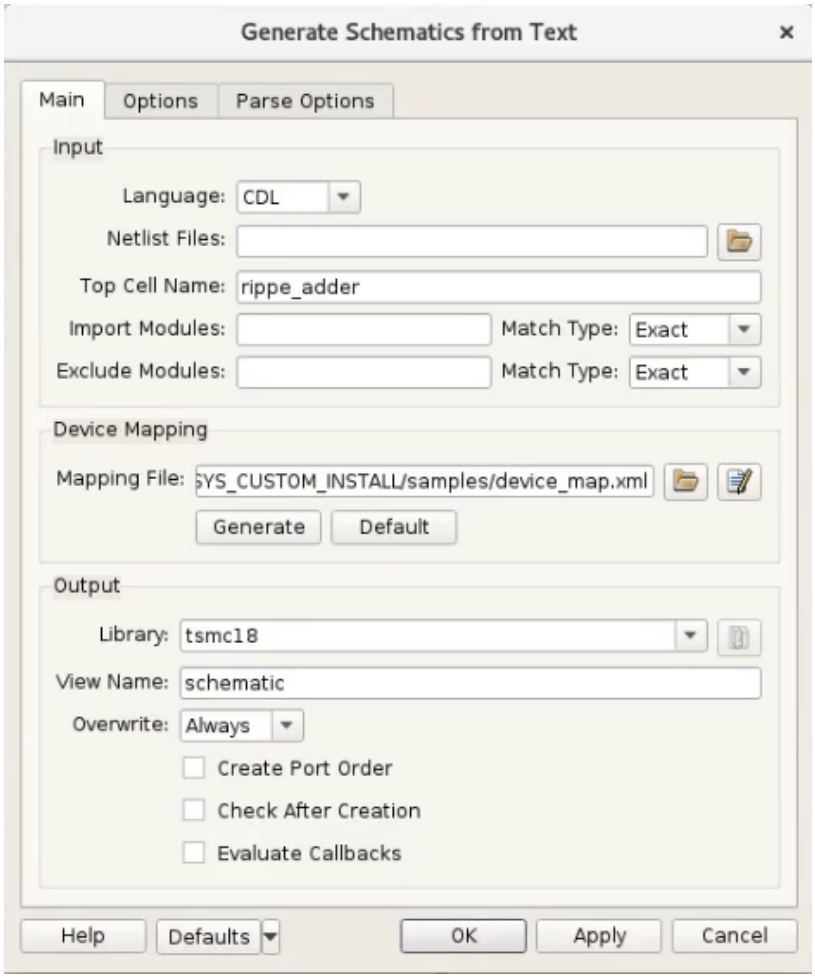


Figura 22: Ventana Schematic from Netlist...

10.4. Generación de esquemático a partir de una *view*

La vista de esquemático puede generarse a partir de otras vistas como el *netlist* en formato CDL. Para esto se utiliza la herramienta de Custom Compiler. En este caso se muestra cómo generar la vista de *schematic* a partir de la vista de CDL. Para poder llevar a cabo este proceso deben estar integradas las librerías necesarias en la herramienta que puedan representar al esquemático. Las librerías EDK y PDK que brinda Synopsys cuentan con diversos modelos de transistores y en el caso de TSMC como no se cuenta con la representación de las celdas a nivel transistor el esquemático se genera a nivel de caja negra.

En la Figura # 23 puede observarse cómo acceder a esta opción de crear una nueva vista a partir de otra. Al seleccionar esta opción debe aparecer una pantalla como la de la Figura # 24, en donde debe seleccionarse la librería y la celda con la que se está trabajando en los apartados de *Library* y *Cell*, ahora en el apartado de *View* en el lado de *Source CellView* debe seleccionarse cdl. Si esta opción de cdl no apareciera en la lista puede solo escribirse directamente en el espacio pero la vista de cdl debe estar en las opciones de *Views* en la ventana de *Library Manager*. Si esta opción no está, debera generarse el archivo cdl como

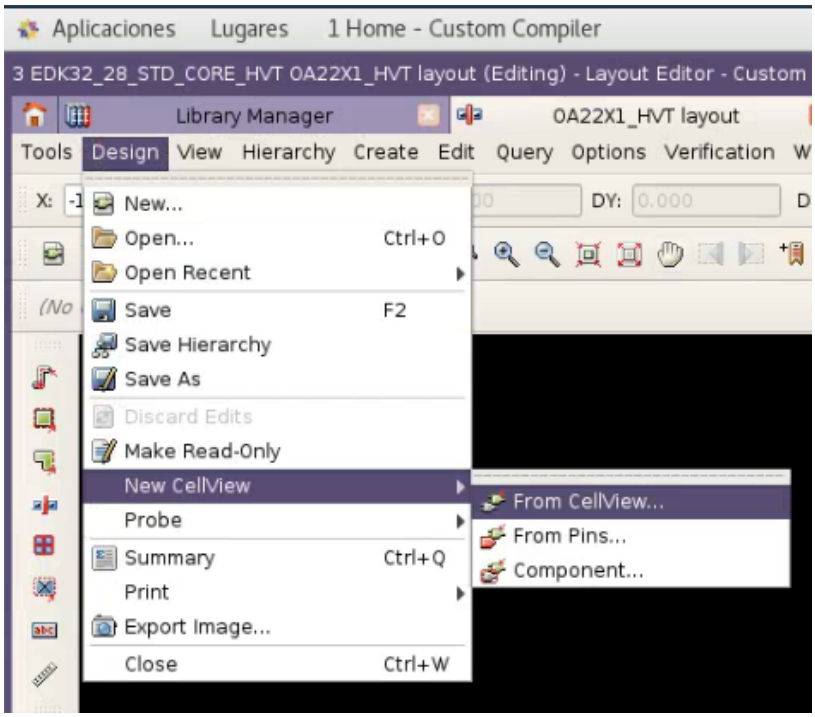


Figura 23: Opción *New CellView*

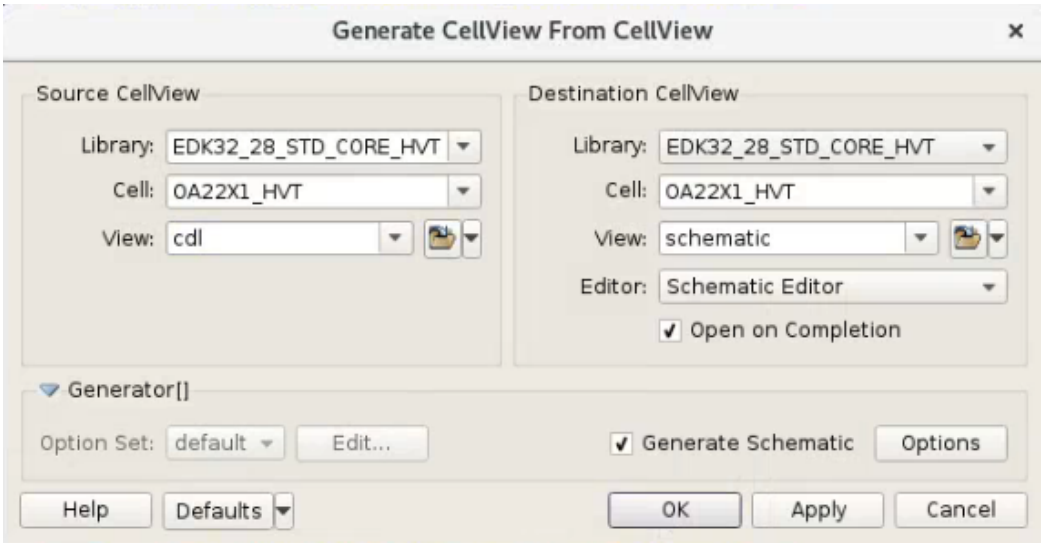


Figura 24: Ventana *New CellView*

se muestra en secciones anteriores de este trabajo. Posteriormente, tendiendo ya la vista de cdl, debe seleccionarse la vista de *schematic* en el apartado de *Destination CellView* y en *Editor* debe seleccionarse la opción de *Schematic Editor*. En el apartado de *Generator* debe seleccionar el *checkbox* que dice *Generate Schematic*. Al generar el esquemático deberá aparecer en las opciones de *Views* en la ventana de *Library Manager* y la celda con la que se está trabajando.

CAPÍTULO 11

Layout (Fase 4)

Para obtener el *layout* del circuito se lleva a cabo la síntesis física. Para esto se utiliza la herramienta de IC Compiler. Este es un proceso bastante elaborado por lo que en este trabajo escrito no se detallará. Si se desea obtener más información detallada acerca de la síntesis física puede chequear [6]. En este capitulo de igual manera se estará trabajando con los circuitos descritos en los capítulos anteriores.

En un bloque específico de código de la síntesis física se genera el archivo GDS que se ha mencionado bastante en capítulos anteriores. Este archivo GDS contiene el diseño de un circuito integrado que puede ser creado por varios programas de CAD. Este mismo contiene información del *layout* de un circuito, incluyendo sus *layers*, figuras geométricas y *labels*. El formato en el que se genera se conoce como GDSII.

El bloque en donde el GDS es generado en el *runset* de síntesis física es el siguiente:

```
...
set_write_stream_options -output_pin {text geometry} -
    keep_data_type
write_stream -lib_name MILKIWAY -format gds "CIRCUIT.gds"
...
```

En este código debe adaptarse el nombre de la *Milkyway library* con la que se está trabajando y nombrar el archivo de salida en donde dice *circuit*. Con esto se generará el archivo GDS necesario para LVS.

Otra manera de generar el archivo GDS es directamente desde la interfaz de IC Compiler. Para esto hay que dirigirse a `File -> Export -> Write Stream` como se observa en la Figura #25. Posteriormente aparecerá una ventana como la de la Figura #26 y debe completarse la información solicitada. En este caso en el apartado de *library name* debe seleccionarse la *milkyway library* con la que se está trabajando y el formato de salida como

GDSII. Debe seleccionarse la opción de *Selected cells* y también seleccionar todas las celdas del circuito. Por último en el apartado de *Stream file name to write* debe escribirse el nombre que se le dará al archivo GDS. Este archivo se guardará en el *run directory* seleccionado.

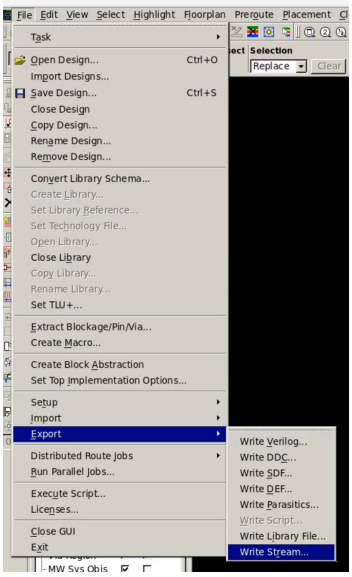


Figura 25: Generar archivo GDS desde IC Compiler

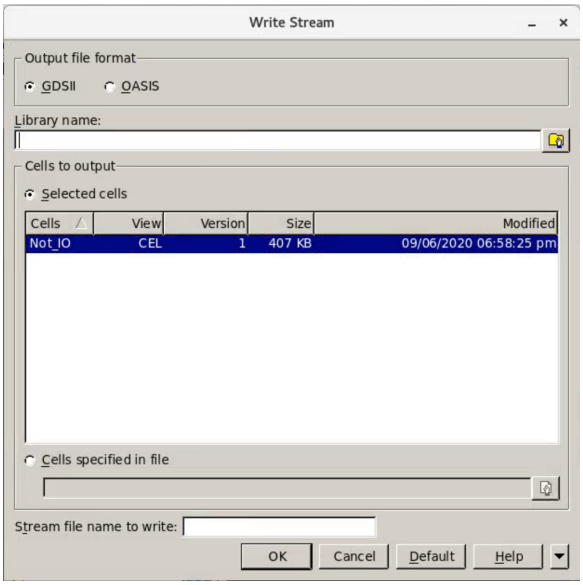


Figura 26: Ventana para generar archivo GDS desde IC Compiler

11.0.1. Layout compuerta Not

En la Figura #27 puede observarse el *layout* para una compuerta Not generado con la herramienta de IC Compiler.

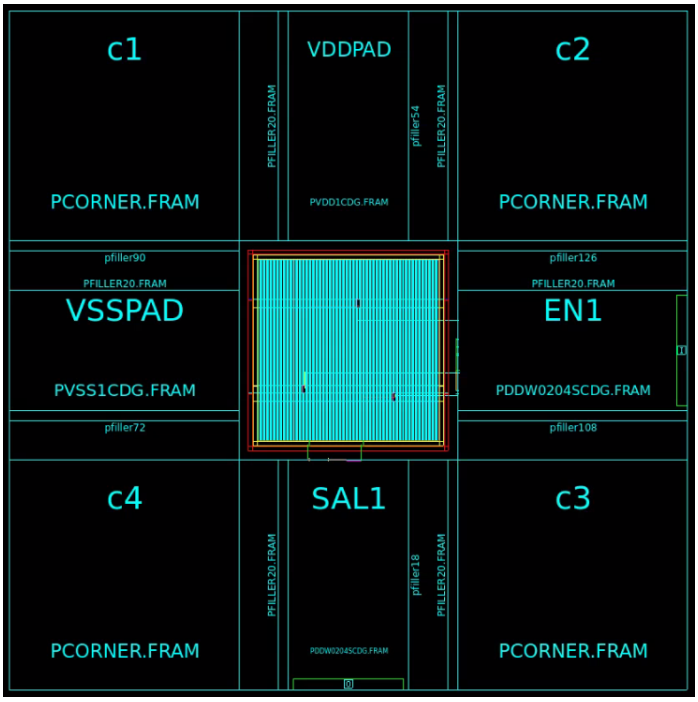


Figura 27: Layout compuerta Not

11.0.2. Layout *Full Adder*

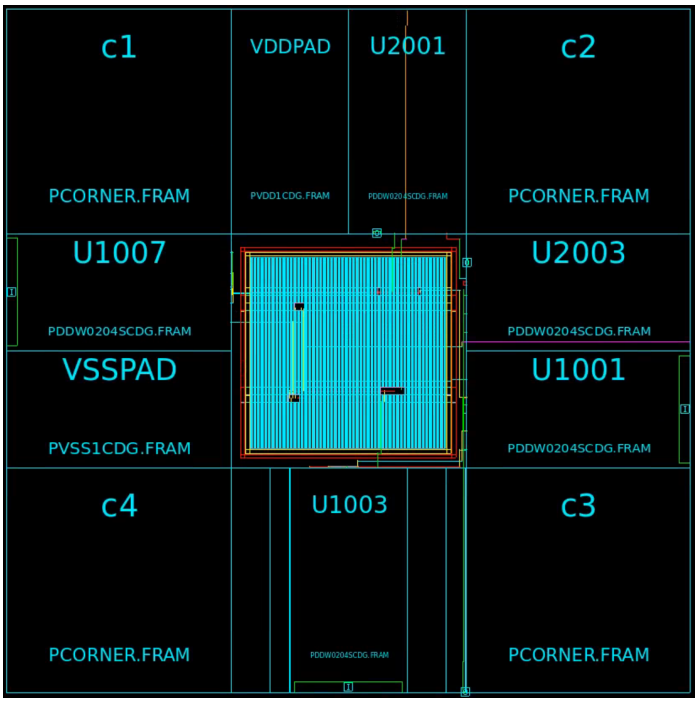


Figura 28: Layout *Full Adder*

En la Figura #28 puede observarse el *layout* para un *Full Adder* generado con la herramienta de IC Compiler.

11.0.3. Layout *Ripple Carry Adder*

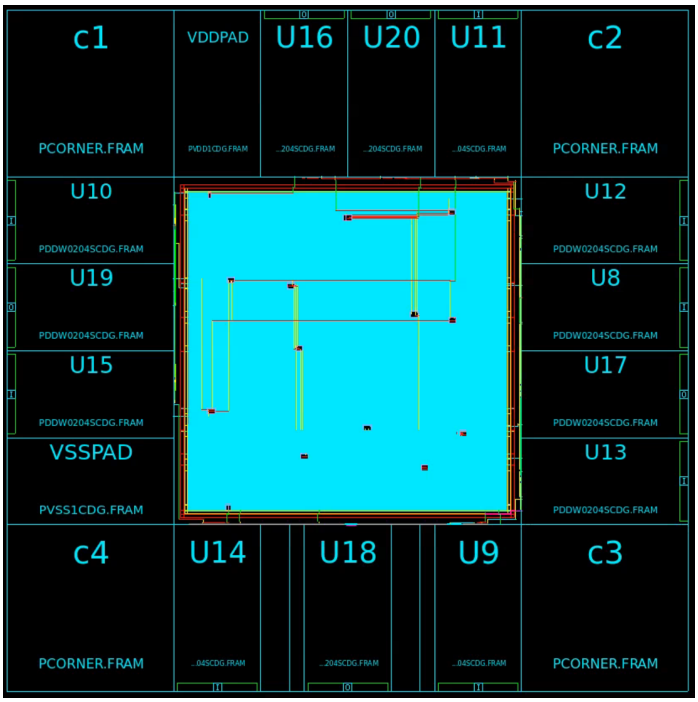


Figura 29: Layout *Ripple Carry Adder*

En la Figura #29 puede observarse el *layout* para un *Ripple Carry Adder* generado con la herramienta de IC Compiler.

11.0.4. Importación de *layout* en Custom Compiler

Custom Compiler es una de las herramientas de Synopsys que nos permite llevar a cabo verificaciones como LPE, LVS y DRC. Esta es una herramienta que cuenta con interfaz gráfica y para LVS se conecta con VUE Tool y utiliza IC Validator para correr el *runset*.

La ventaja de esta herramienta es que al generar los archivos de salida presenta una ventana en donde se muestran los errores para LVS y para ERC. Además, estos pueden verse directamente en el *layout* del circuito para ver exactamente en dónde está el problema. En esta sección se explicará cómo importar en Custom Compiler el *layout* del circuito desde IC Compiler. Esta herramienta trabaja con una librería que se conoce como PDK descrita a continuación.

Importación de PDK a Custom Compiler

Para poder importar la librería Custom Compiler diríjase a la ruta en donde está la librería PDK y justo en esta ruta ejecute una terminal. Al ejecutar la terminal escriba el siguiente comando: `custom_compiler` y presione la tecla *enter*. Al correr este comando deberá aparecer una ventana como la de la Figura #30, en esta misma seleccione *Library*

Manager y deberá aparecer una ventana como la de la Figura #31. Puede observarse en la esquina inferior derecha de estas ventanas que aparece un símbolo parecido al de una ventana de comandos, en esta misma aparecen los errores y mensajes de cada acción que uno hace en la herramienta.

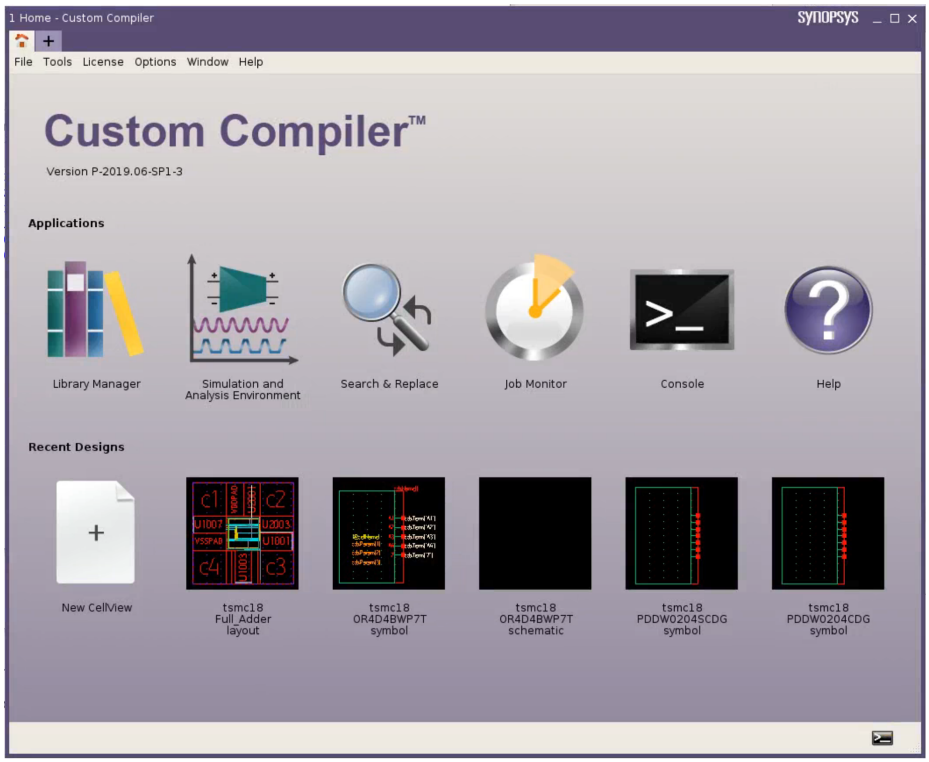


Figura 30: Iniciando Custom Compiler

En la Figura #30 pueden observarse cinco apartados:

1. *Libraries*: en este apartado se encuentran las librerías que se han importado, pueden ser librerías PDK o también aparecen librerías que pueden importarse desde IC Compiler.
2. *Cell Categories*: la herramienta divide a las celdas en categorías, como por ejemplo en resistencias, compuertas lógicas, capacitores, etc.
3. *Cells*: en este apartado pueden observarse las celdas por su nombre.
4. *Views*: la herramienta de Custom Compiler nos permite distintos tipos de vistas como *layout*, esquemático, símbolo, etc.
5. *Previews*: puede observarse una pequeña pantalla con un adelanto de la vista seleccionada.

Importación de librería en IC Compiler a Custom Compiler

Para la importación de esta librería es necesario que se haya hecho previamente la síntesis física del diseño. La librería que se importa es la *Milkyway Library* y esta debe tener contenido

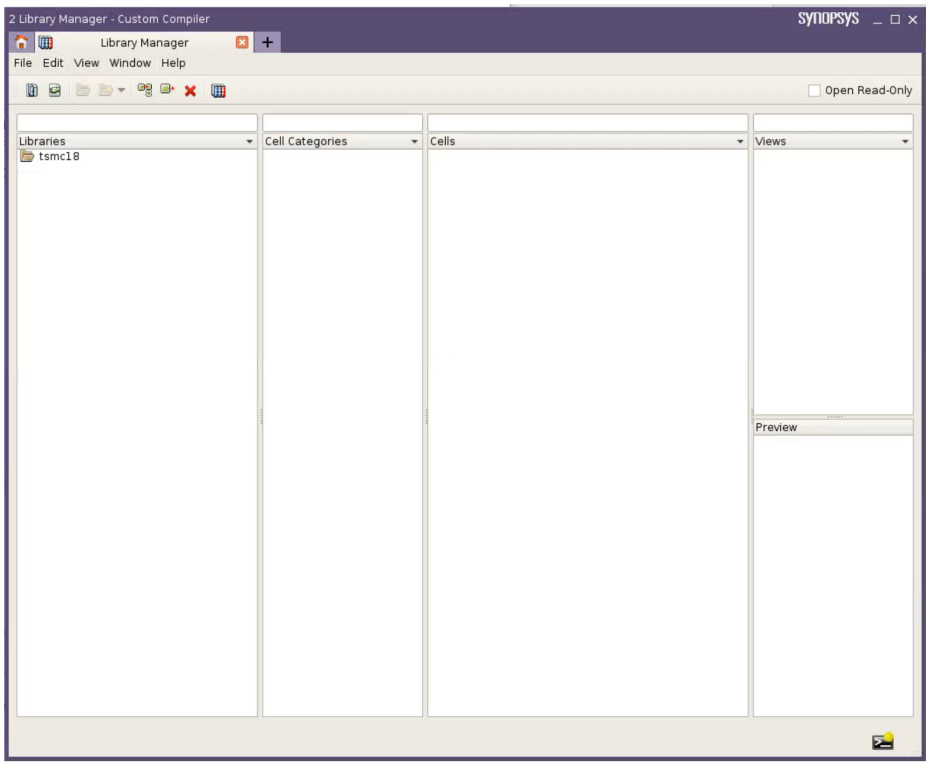


Figura 31: Custom Compiler

previo a ser importada.

Asumiendo que se realizó previamente la síntesis física hay que dirigirse a `File -> Add ICC Library` como se muestra en la Figura #32.

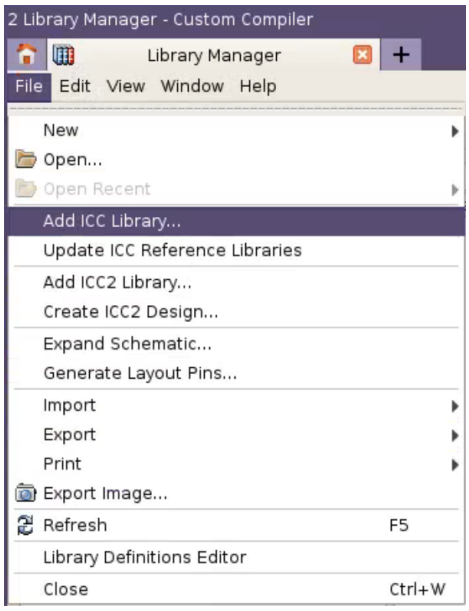


Figura 32: *ICC Library*

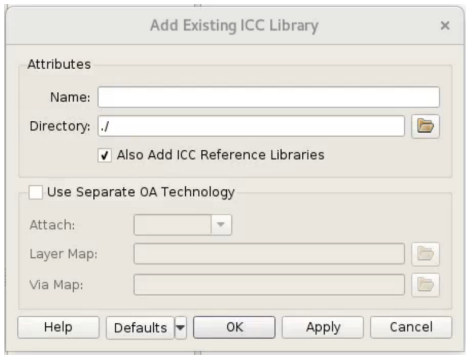


Figura 33: *Add ICC Library*

Al presionar esta opción deberá aparecer una ventana como la de la Figura #33. En la entrada de *Directory* debe buscarse la *Milkyway Library* con la que se trabajó en IC Compiler y en la entrada de *Name* debe ponerse un nombre (preferiblemente distintivo) a la librería que se importará. Al importarla debería aparecer en el apartado de *Libraries* descrito anteriormente.

Por otro lado está también la opción de importar la información de la *Milkyway Library* a una librería ya creada en Custom Compiler. Para esto hay que dirigirse a `File -> Import -> From ICC` como se observa en la Figura #34.

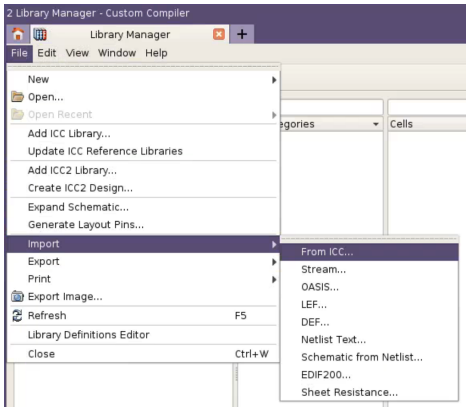


Figura 34: *Import from ICC*

Al seleccionar esta opción deberá aparecer una ventana como la de la Figura #35. En esta ventana deberá buscarse la *Milkyway Library* con la que trabajó en IC Compiler y deben ingresarse las celdas en específico que se desean importar. En la entrada de *OA Library* debe seleccionarse la librería en la que se desea importar las celdas de la *Milkyway Library* que se seleccionó.

En este caso debe ubicarse el apartado de *Libraries* y buscar la librería en la que se importó la información desde IC Compiler y verificar que aparezcan las celdas que se importaron.

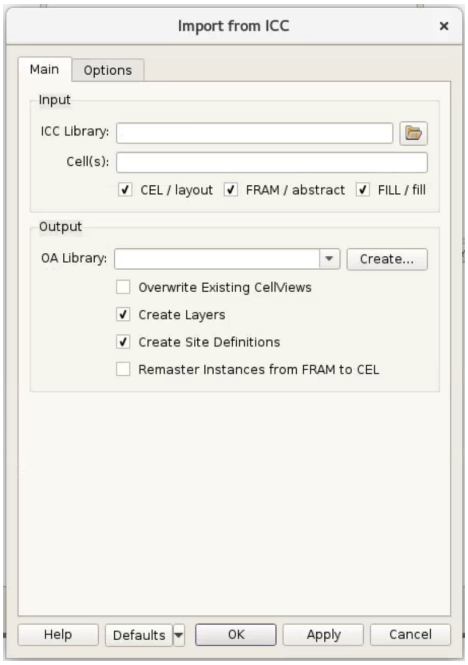


Figura 35: Importación de *Milkyway Library*

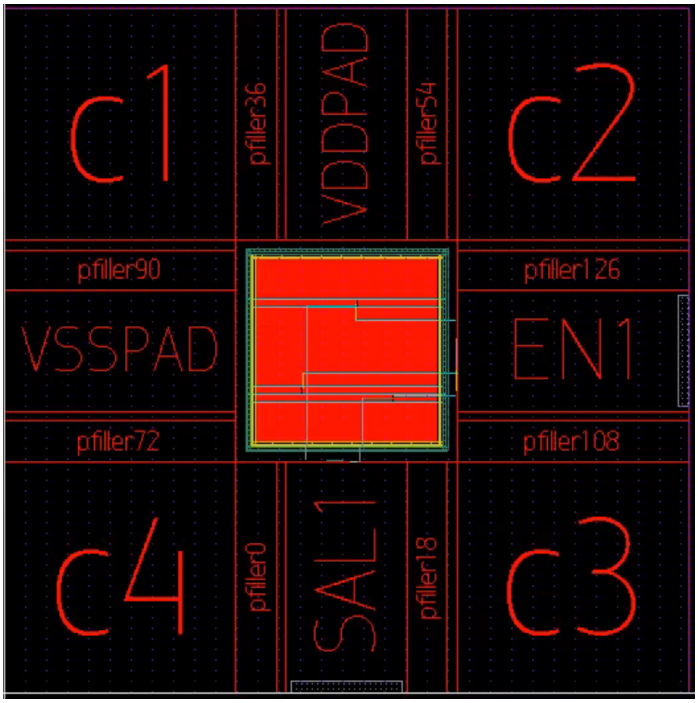


Figura 36: *Layout* compuerta Not en Custom Compiler

Layouts en Custom Compiler

Puede observarse en la Figura #36 el *layout* de la compuerta Not, en la Figura #37 el *layout* de un *Full Adder* y en la Figura #38 el *layout* de un *Ripple Carry Adder*. Todos estos

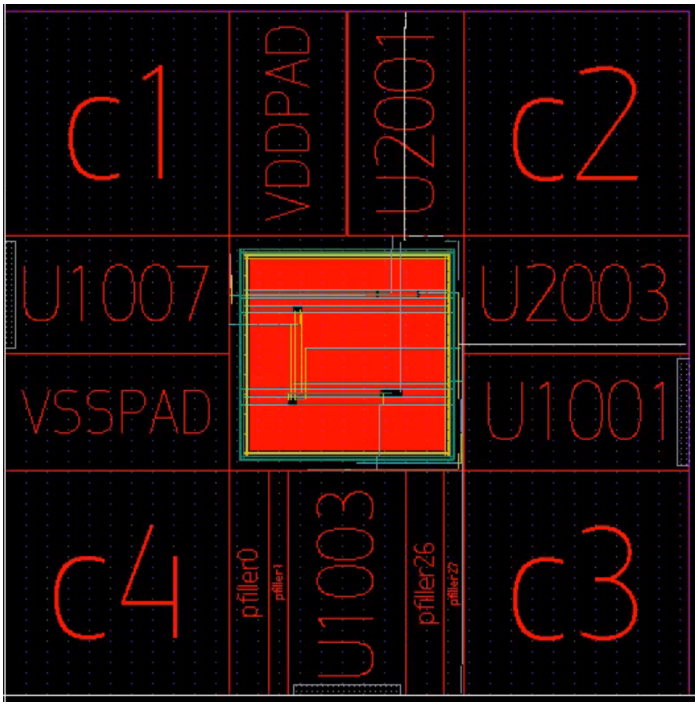


Figura 37: *Layout Full Adder* en Custom Compiler

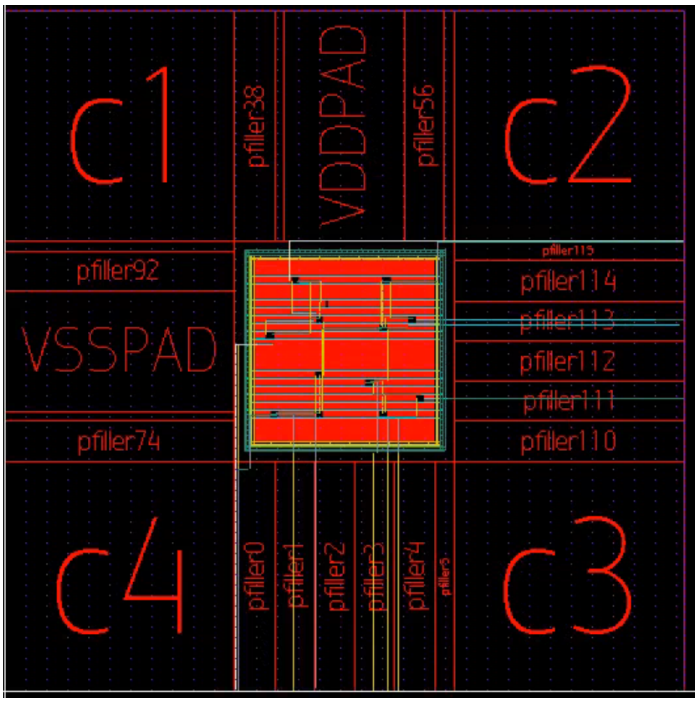


Figura 38: *Layout Ripple Carry Adder* en Custom Compiler

layouts fueron importados de IC Compiler.

11.0.5. Exportación de archivo GDS en Custom Compiler

El archivo GDS también se puede exportar directamente desde Custom Compiler. Para esto desde la ventana de *Library Manager* en Custom Compiler debe abrirse `File -> Export -> Stream` como se puede observar en la Figura #39 y deberá aparecer una ventana como la de la Figura #40.

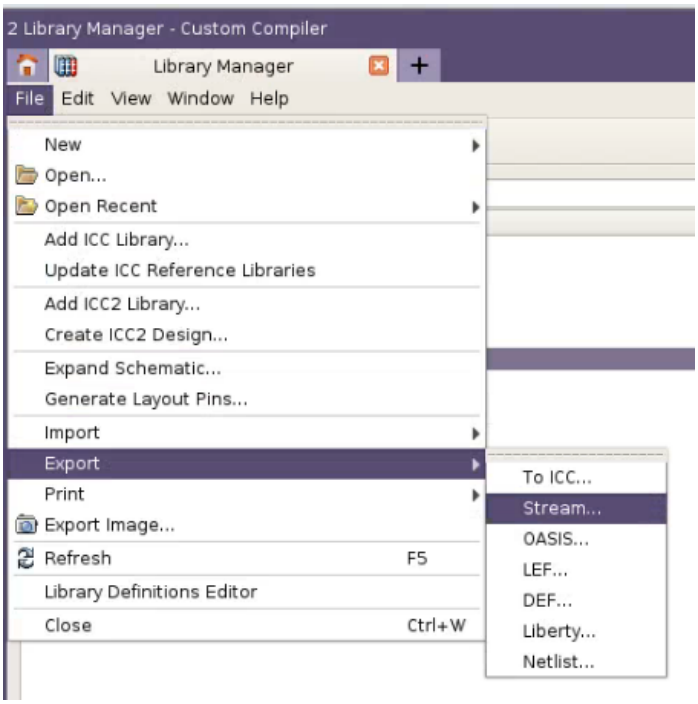


Figura 39: Exportar GDS desde Custom Compiler

En la ventana de la Figura #40 en el apartado de *Input* deben configurarse las opciones de *Library* en donde debe seleccionarse la librería con la que está trabajando en Custom Compiler. La segunda opción que dice *Top Cell* se refiere a la *top cell* del circuito con el que se está trabajando y por último en *View* se selecciona la vista de *layout* ya que de esta es que se obtiene el GDS necesario.

En la misma ventana de la Figura #40 hay una pestaña de *Options* en donde se pueden establecer algunas configuraciones para la exportación del archivo GDS, las que están seleccionadas por *default* deberían funcionar sin problema alguno.

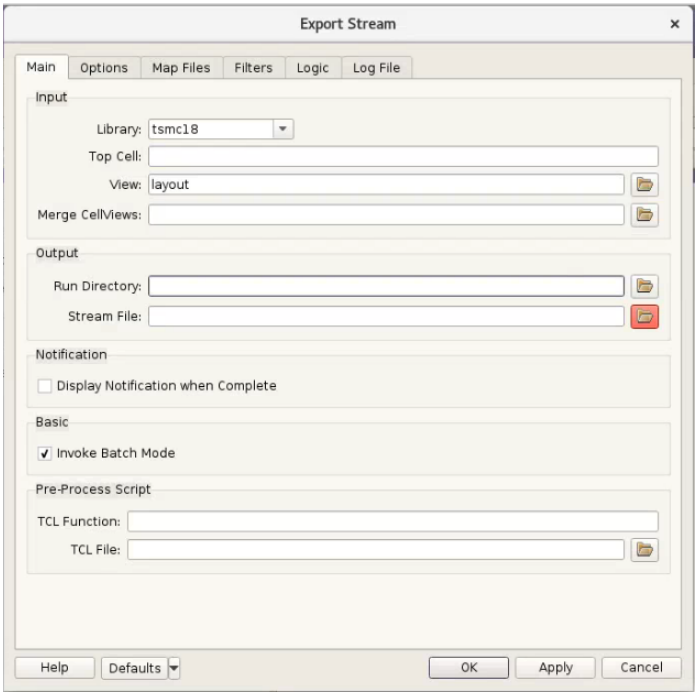


Figura 40: Ventana de configuración para exportar GDS desde Custom Compiler

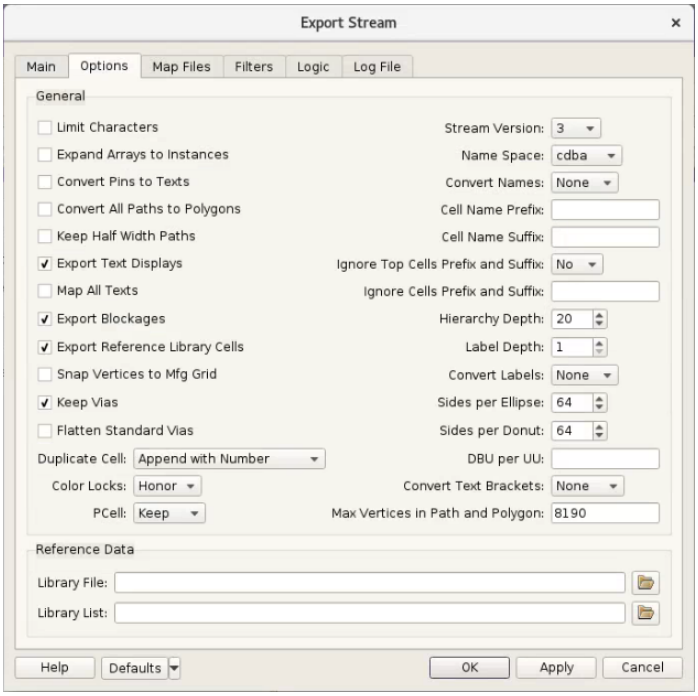


Figura 41: Opciones para exportar GDS desde Custom Compiler

CAPÍTULO 12

LVS (Fase 5)

En este capítulo se detallan las maneras de llevar a cabo la verificación física de LVS. Acá se explicarán tres maneras distintas de correr dicha prueba. De igual manera pueden verse los videotutoriales en caso haya alguna duda. Es importante mencionar que en este punto, se debe contar con los archivos siguientes:

1. *Layout* del circuito
2. Archivo GDS
3. CDL *netlist* de esquemático en formato ICV
4. *Runset* de IC Validator para LVS

Si aún no se cuenta con alguno de estos archivos mencionados anteriormente se puede chequear los capítulos anteriores en donde se detalla cómo obtener cada uno de ellos. Como se mencionó, existen varias maneras de correr LVS, cada una utiliza la herramienta de IC Validator pero las pruebas se pueden hacer en distintos entornos. A continuación se mencionan las tres maneras de correr LVS:

1. Mediante comandos
2. Mediante VUE *Tool*
3. Mediante Custom Compiler

12.1. *Black Box* LVS

Debido a que no se cuenta con la descripción completa de las celdas de las librerías *I/O Cell Library* y *Standard Cell Library* se utiliza el *Black Box* LVS. Este caso es prácticamente cuando no se cuenta con la vista de *layout* de las celdas o bien, la descripción completa de las celdas en el CDL o bien, la falta de ambos.

En este caso, para celdas específicas no todo el contenido es chequeado. La comparación se hace en base a la representación en cajas negras de las celdas, en donde solo se compara la instancia de las mismas y sus puertos.

Las *black boxes* deben configurarse directamente en el *runset* de LVS. Esta configuración se hace de la siguiente manera:

```
...
lvs_black_box_options(
        equiv_cells ={{schematic_cell = "NOMBRE_CELDA",
            layout_cell = "NOMBRE_CELDA"}},
        remove_schematic_ports = {"NOMBRE_PUERTO1", "
            NOMBRE_PUERTO2",...}
);
...
```

Es muy importante mencionar que en el *runset* de LVS que fue brindado por TSMC los dispositivos de las librerías *I/O Cell Library* y *Standard Cell Library* no están definidos. En esta sección del *runset* se definen estos mismos y de no ponerlo, mostrará un error. En la parte de `equiv_cells` deben establecerse las equivalencias para cada celda y la parte de `remove_schematic_ports` deben colocarse todos los puertos de cada celda. Esta última parte es necesaria porque se está trabajando con celdas incompletas, el *netlist* del *layout* tendrá menor cantidad de puertos en comparación al *netlist* de esquemático. El diseño no es lo suficientemente completo para obtener las interacciones jerárquicas necesarias para crear los puertos, entonces al comparar, los pines de las *black boxes* definidas dará error (*unmatched*). Para mayor información de dónde colocar esta configuración puede consultar en *IC Validator LVS User Guide* [8] y *IC Validator Reference Manual* [14]. También hay información valiosa en el *Application Note* de Imec titulado como *HOW TO RUN A BLACK BOX LVS* [15].

12.2. LVS mediante comandos

Correr LVS mediante comandos es la manera más sencilla pero a la vez podría decirse que es la manera más desordenada. Esto por la manera en que se muestran los resultados y el orden que hay que llevar para correr la prueba. Para esto deben seguirse los pasos que se describen a continuación:

12.2.1. Paso 1: preparación de archivos

Para correr esta prueba de manera ordenada se recomienda crear una carpeta que se llame LVS en donde se prefiera y dentro de esta carpeta crear una carpeta por cada circuito al que se le hará la verificación física de LVS. Dentro de la carpeta de cada circuito puede crearse otra carpeta para los archivos de salida, puede nombrarse como *Outputs*. Esto porque en cada prueba se generan varios archivos y carpetas de salida que contienen reportes y archivos importantes para interpretar. De esta manera será mucho más fácil el encontrar cada uno de ellos y quedarán registrados de manera ordenada.

Ahora, dentro de la carpeta de cada circuito puede crearse una nueva carpeta para los archivos de entrada. Acá pueden colocarse los archivos que se utilizarán para correr LVS. En este caso, el archivo GDS, el CDL *netlist* del esquemático en formato ICV. Esto se hace para poder correr la prueba de una manera ordenada y que sea mucho más sencillo el obtener los resultados.

12.2.2. Paso 2: *environment setup* en el *runset*

En el *runset* que brindó TSMC existe un apartado de *environment setup* en donde deben colocarse las rutas de los archivos de esquemático y *layout* con los que está trabajando. Este es un *runset* bastante largo por lo que puede buscar este apartado utilizando `ctrl + f`.

12.2.3. Paso 3: comando de LVS

Como se mencionó anteriormente, la herramienta que corre la prueba de LVS es IC Validator. Los comandos para correr dicha prueba son directamente de esta herramienta. A continuación se describe el comando:

```
icv -i ARCHIVO_GDS -c NOMBRE_TOPCELL -s CDL_NETLIST -sf ICV -vue RUNSET
```

IC Validator cuenta con varias opciones en sus comandos, estas mismas están descritas en el manual de *IC Validator User Guide* [16]. Por ahora hay que concentrarse en el comando descrito anteriormente, que contiene lo suficiente para correr la prueba. A continuación se describen los parámetros:

1. -i : nombre de la librería con la que se está trabajando, en este caso se utilizará GDSII. Entonces acá se coloca el archivo GDS (colocarlo con su ruta exacta).
2. -c : nombre de la *top cell* con la que se está trabajando.
3. -s : acá se coloca el CDL *netlist* en formato ICV (colocarlo con su ruta exacta).
4. -sf : formato de CDL *netlist* (debe ser ICV).
5. -vue : acá se coloca el *runset* de LVS (colocarlo con su ruta exacta).

Es muy importante que los archivos que se colocan en este comando sean los mismos que se establecieron en el *environment setup* del *runset*.

12.2.4. Paso 4: correr comando

Para correr el comando debe abrirse la carpeta de *outputs* que se creó en el paso 1. Dentro de esta carpeta debe abrirse una terminal y colocar el comando del paso anterior adaptado a los archivos con los que se está trabajando y presionar la tecla *enter*. La prueba empezará a ejecutarse en la terminal, se desplegará el porcentaje de los dos procesos importantes de LVS, el de extracción y el de comparación. Para que la prueba sea exitosa, ambos deben llegar al 100 % y deberá salir el siguiente mensaje: `IC Validator is done` en la misma terminal. Si este último mensaje se despliega, los archivos generados deberán aparecer en esta misma carpeta de *outputs*.

12.3. LVS mediante VUE *Tool*

Para correr LVS mediante la VUE *Tool* se hace uso de IC Compiler. Debe tenerse abierta la ventana en donde se generó el *layout* en la síntesis física y dirigirse a `Verification -> IC Validator VUE` como se muestra en la Figura #42.

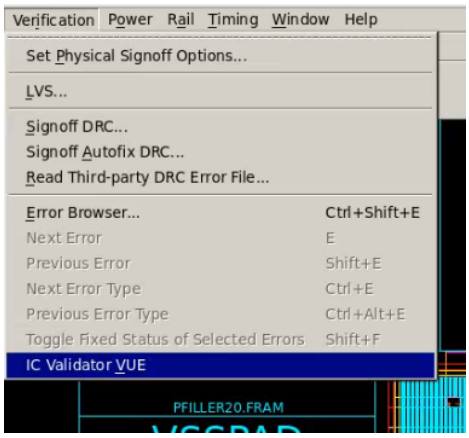


Figura 42: Abrir VUE *Tool*

Al seleccionar esta opción deberá abrirse una ventana como la de la Figura #43. En esta ventana en la entrada de *Runset File* debe buscarse el archivo del *runset* de LVS. En la entrada de *Run Directory* seleccionar en donde se generarán los archivos de salida de la prueba, se recomienda hacer lo mismo que en el paso 1 de preparación de archivos de la sección anterior.

Por otro lado, en el apartado que dice *Layout Input*, debe seleccionarse la opción de GDSII, posteriormente en la entrada de *Library* debe colocarse el archivo GDS que se generó y en la entrada de *Cell* el nombre de la *topcell* del circuito con el que está trabajando.

En esta misma ventana, del lado izquierdo debe seleccionarse la opción de LVS. Al seleccionarla deberá cambiar la ventana como a la que se muestra en la Figura #44. En esta misma debe colocarse en la entrada de *File*, el archivo CDL *netlist* en formato ICV. En la entrada de *Cell*, debe colocarse el nombre de la *topcell* del circuito con el que está trabajando y en el seleccionador que aparece del lado derecho, debe colocarse ICV.

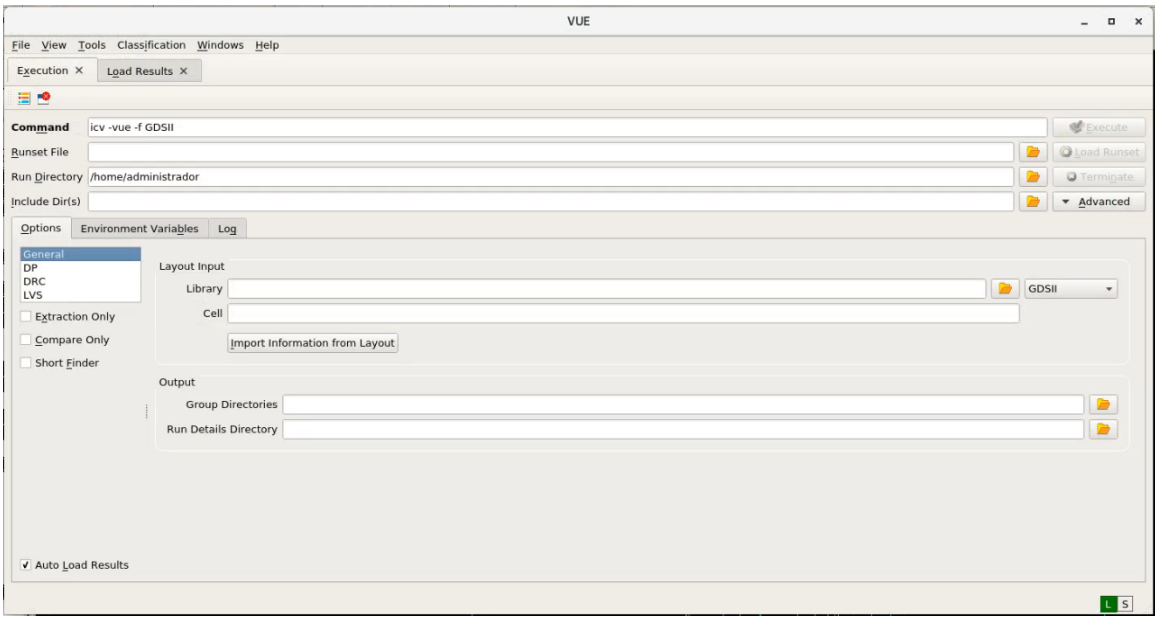


Figura 43: VUE *Tool*

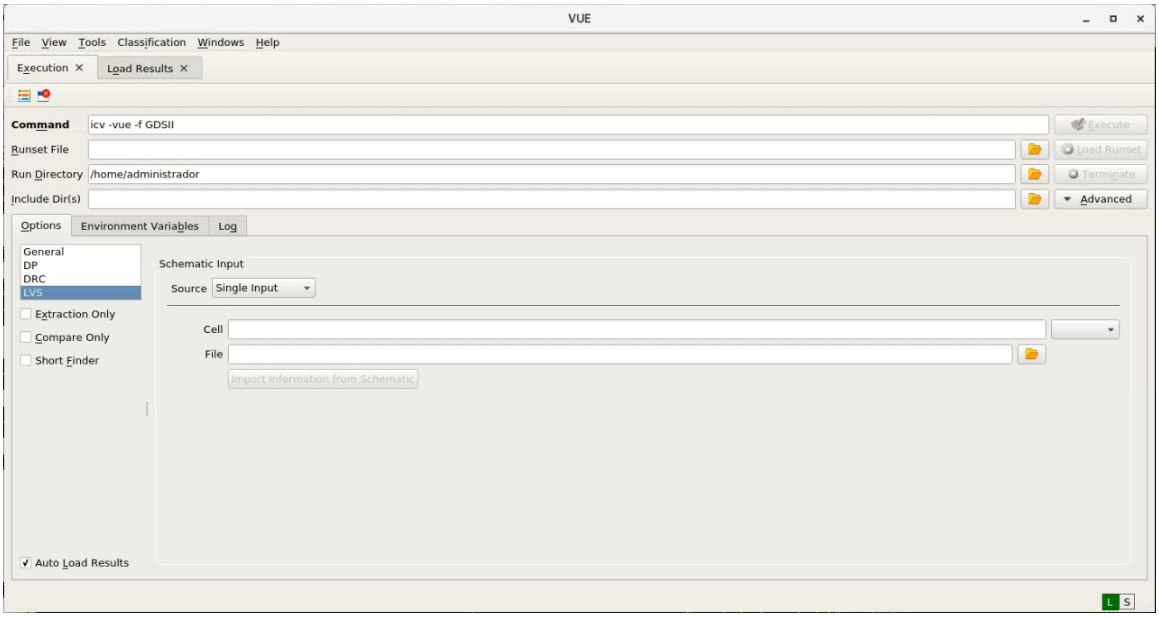


Figura 44: VUE *Tool* LVS

Después de haber configurado todo en esta ventana hay que verificar que los archivos a los que está haciendo referencia en el *runset* en el apartado de *environment setup* coincidan con los que se configuraron en esta misma pestaña. Al haber chequeado esto debe presionar el botón de *Execute* y la prueba empezará a correr. Los archivos de salida se generarán en el directorio seleccionado. De nuevo, se recomienda crear la carpeta de *outputs* para cada circuito con el que haga pruebas, esto porque es una manera mucha más ordenada de trabajar y se evitará el error humano al seleccionar los archivos correctos para interpretar resultados.

12.4. LVS mediante Custom Compiler

Para poder correr LVS en Custom Compiler debe haberse importado el *layout* de IC Compiler a Custom Compiler. De no haberlo hecho se puede revisar el capítulo anterior en donde se detalla este proceso. Asumiendo que la importación fue correcta, debe abrirse en Custom Compiler el *layout* del circuito con el que esté trabajando y seleccione `Verification -> LVS -> Setup and Run` como se muestra en la Figura #45.

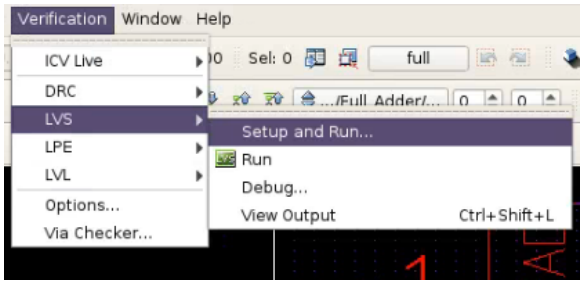


Figura 45: LVS en Custom Compiler

Al seleccionar esta opción deberá abrirse una ventana como la de la Figura #46. Acá debe seleccionarse en la entrada de *Run Dir* el directorio en donde se generarán los archivos de salida. De preferencia puede hacerse lo mismo de las opciones anteriores, crear una carpeta que se llame *outputs* en donde se generarán todos los archivos de salida de la prueba.

Posteriormente, en el apartado de *Layout* debe seleccionarse la opción de *OpenAccess*, esto ya que se trabajará con el *layout* que se importó del circuito. Posteriormente, en la entrada de *Library* debe seleccionarse la librería con la que se está trabajando en Custom Compiler. En la entrada de *Cell*, ingresar el nombre de la *top cell* del circuito con el que está trabajando y por último en la entrada de *view* seleccionar la opción de *layout*.

En el apartado de *Schematic/Config* debe seleccionarse la opción de *Netlist*. Esto ya que no se cuenta con un esquemático del circuito, la comparación se hará directamente con el CDL *netlist* en formato ICV que se generó. En la entrada de *Netlist*, ingresar el archivo de CDL *netlist* en formato ICV. En el seleccionador de formato debe seleccionarse la opción de ICV. En la entrada de *Cell*, ingresar el nombre de la *topcell* del circuito con el que está trabajando.

Por último, en el apartado de *Job Parameters*, en el seleccionador de *Tool* hay que escoger IC Validator. En la entrada de *Runset*, hay que buscar el archivo de *runset* de LVS. Para terminar, en el seleccionador de *Job Class*, debe escogerse la opción de LVS. Antes de presionar el botón de *OK*, hay que verificar que el archivo de esquemático en el *runset* coincida con el configurado en esta misma ventana. El archivo GDS en este caso no importa, pero el de esquemático sí. Al haber verificado esto, debe presionarse *OK* y deberá desplegarse una ventana de la VUE *Tool*, esto ya que la misma herramienta de Custom Compiler conecta con VUE *Tool* para desplegar sus resultados. Por esta razón, esta manera de correr LVS es la mejor ya que al abrirse la VUE *Tool*, los resultados se muestran de una manera más amigable e interactiva. Incluso, de haber un error, puede seleccionarse y lo marcará directamente en el *layout* del circuito.

Por ahora no se mostrarán los resultados de pruebas. En el siguiente capítulo se mostrarán

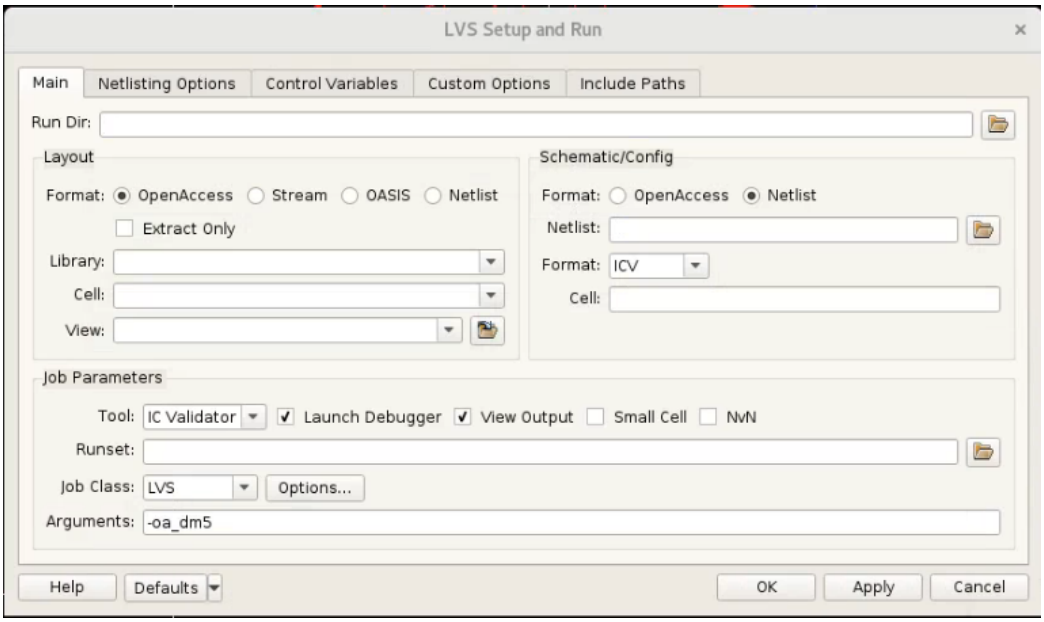


Figura 46: Configuración LVS en Custom Compiler

tres ejemplos de LVS con los circuitos trabajados en los capítulos anteriores. Para cada circuito se hará la prueba de las tres distintas maneras.

CAPÍTULO 13

Interpretación de resultados (Fase 6)

Se llevó a cabo la verificación física de LVS para los tres cicuitos con los que se ha trabajado en los demás capítulos y se hizo de las tres maneras para cada uno de ellos. A continuación se detallan los resultados para cada circuito de las tres maneras.

13.1. Descripción de archivos generados

En el caso de LVS la herramienta de IC Validator genera archivos de errores, *log* y archivos de reportes de equivalencias. Los archivos de errores lista los *nets* y dispositivos del *layout* que no son equivalentes con el esquemático. Los archivos de errores, resúmenes y el *netlist* del *layout* se guardan en el directorio de trabajo. También archivos adicionales son incluidos en este mismo directorio pero en la carpeta de *run_ details* los cuales son útiles para generar los archivos de errores.

Todos los archivos generados se identifican por la *top cell* del circuito con el que se está trabajando y la abreviación del archivo generado, es decir `TOP_CELL.ABREVIACIÓN`.

Se llevó a cabo la verificación física de LVS para los tres cicuitos con los que se ha trabajado en los demás capítulos y se hizo de las tres maneras para cada uno de ellos.

13.2. Archivo *RESULTS*

La herramienta de IC Validator crea un archivo de resultados el cual puede encontrar como `TOP_CELL.RESULTS` (debe adaptarse el nombre de la *top cell* del circuito con el que se está trabajando). Este archivo tiene los resultados de áreas clave de la prueba y también

tiene el comando utilizado para correr la prueba. Este es el punto de partida para analizar e interpretar los resultados de la prueba. Este archivo contiene muchas secciones, pero haremos énfasis en aquellas que nos ayudarán a interpretar los resultados de LVS. Acá también se presentan algunos resultados de DRC en unas de las secciones, si se desea más información de esto puede chequear el manual *IC Validator User Guide* [16].

Results Header

En el encabezado de resultados se encuentran dos secciones, una de DRC y otra de LVS. Puede observarse en la Figura #47 en donde nos indica para LVS si es (*FAIL/PASS*), es decir, si falla o pasa la prueba. En DRC marca si la prueba sale sin errores o con errores (*NOT CLEAN/CLEAN*).

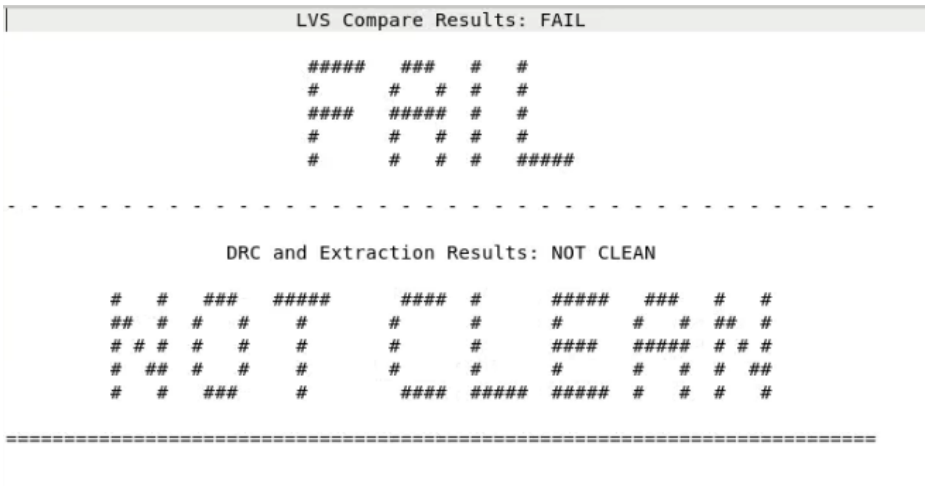


Figura 47: Encabezado de archivo de resultados

En caso que alguna de las pruebas (DRC/LVS) no se complete, muestra una línea que dice "*RESULTS: RUN ABORTED*".

Resumen de resultados

Esta parte se divide en las siguientes dos secciones:

1. Reporte de las estadísticas del proceso de comparación de LVS: puntos de equivalencia, número de *black box cells* correctas e incorrectas, número de equivalencias correctas e incorrectas y número de errores prioritarios.
2. Reporte y estadística de errores de la extracción de dispositivos: numero de reglas que fueron utilizadas, número de reglas no ejecutadas, número de reglas sin violaciones y el número total de violaciones.

Como puede observarse en la Figura #48, están las partes descritas anteriormente.

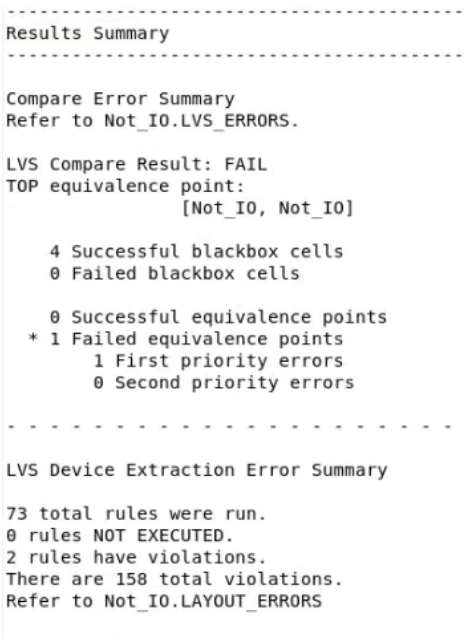


Figura 48: Resumen de resultados LVS

13.3. Archivo de errores en *layout*

La herramienta de IC Validator crea un archivo de errores el cual nombra como `TOP_CELL.LAYOUT_ERRORS` (debe adaptarse el nombre de la *top cell* del circuito con el que se está trabajando). Este archivo contiene resultados de la prueba, un resumen de los errores y una lista detallada de errores.

En el inicio del archivo se indica si el *layout* está correcto o con errores (*CLEAN/ERRORS*).

Posteriormente se muestra la sección de resumen de los errores, en donde se encuentran las funciones y los errores que estas tienen (en caso de tenerlos) de la parte de ERC.

Por último, está una sección detallada. Cada reporte de error contiene: una descripción del error que indica que el chequeo se está realizando, el nombre de estructura en donde el error se está dando, la posición en donde se dio el error y más información de este. Las coordenadas son dadas jerárquicamente, es decir, relativo a la estructura nombrada y solo una vez por cada instancia.

Estas secciones pueden observarse en la Figura #49.

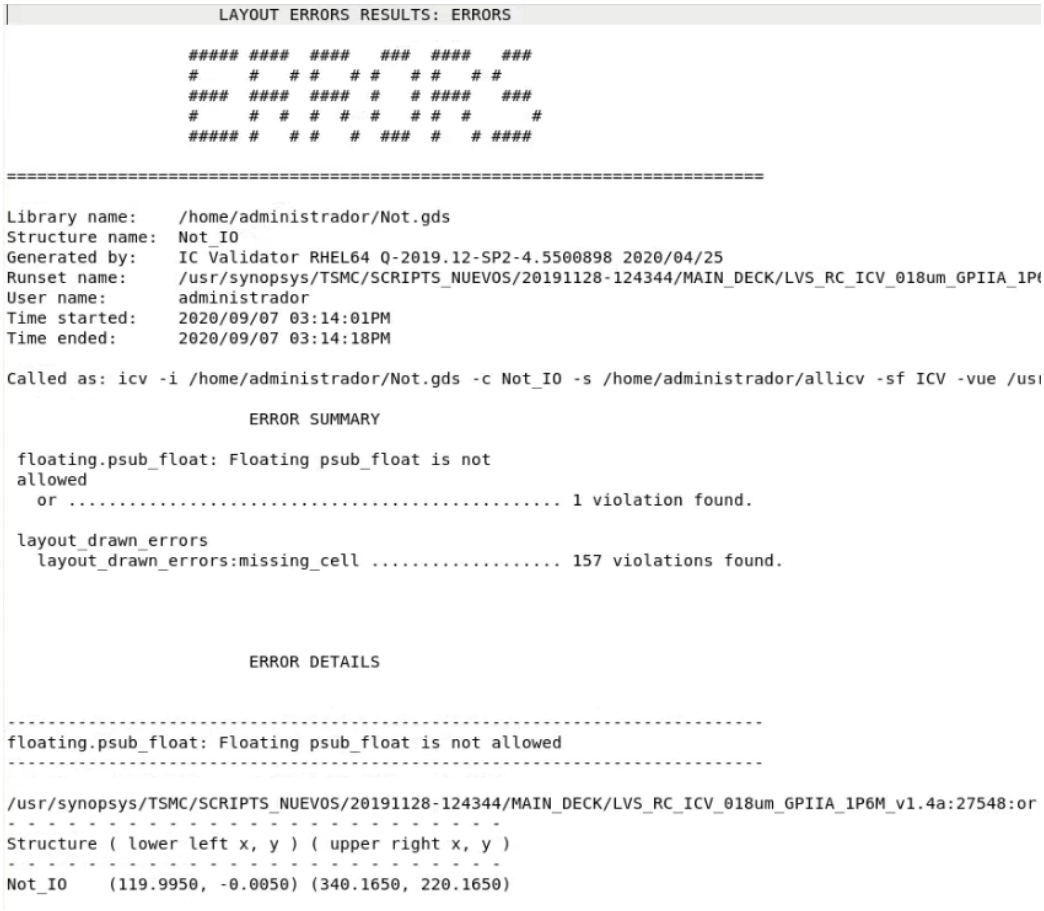


Figura 49: Errores en *layout*

13.4. Archivo de errores en LVS

La función de *compare()* genera el archivo de errores de LVS y la herramienta de IC Validator nombra este mismo como `TOP_CELL.LVS_ERRORS` (debe adaptarse el nombre de la *top cell* del circuito con el que se está trabajando). En este mismo se indica si la comparación entre *layout* y esquemático se logró correctamente. Este es uno de los archivos más importantes a interpretar, ya que este cuenta los errores propios de la comparación entre ambos elementos (*layout* y esquemático).

En este archivo también se muestran los puntos de equivalencia del circuito y muestra detalladamente si estos fallaron o no. Además, presenta diagnósticos que son diseñados para brindar asistencia para ayudar a solucionar el diseño. De igual manera en la parte superior muestra si la comparación fue exitosa o no (*PASS/FAIL*).

En la Figura #50 puede observarse un resumen detallado de las instancias encontradas tanto en *layout* como en esquemático y muestra si estas instancias hacen *match* o no. Por otro lado, muestra los puertos y *nets* encontrados.

Puede observarse en la Figura #51 el estado del proceso de comparación de la prueba. Por otro lado en la Figura #52 puede observarse un diagnóstico, en donde se indican faltas

```
> Not_IO != Not_IO (level = 0)

Error summary:

          5 Unmatched schematic instances
          7 Unmatched schematic nets
          0 Unmatched layout instance
          0 Unmatched layout net

          0 Matched instance
          0 Matched net

Port summary:

          2 Unmatched schematic ports
          0 Unmatched layout port
          0 Matched port


Post-compare summary (* = unmatched devices, nets or ports):

        Matched      Unmatched    Unmatched        Instance types
                     schematic    layout           [schematic, layout]
        ---------    ---------    ---------        --------------------
                0            1            0      * BLACK_BOX[CKND0BWP7T, CKND0BWP7T]
                0            2            0      * BLACK_BOX[PDDW0204SCDG, PDDW0204SCDG]
                0            1            0      * BLACK_BOX[TIEHBWP7T, TIEHBWP7T]
                0            1            0      * BLACK_BOX[TIELBWP7T, TIELBWP7T]
        ---------    ---------    ---------        --------------------
                0            5            0      * Total instances

                0            7            0      * Total nets

                0            2            0      * Total ports
```

Figura 50: Tabla de errores

```
Final comparison result:FAIL

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


TOP equivalence point:
               [Not_IO, Not_IO]

Comparison summary

    4 Successful blackbox cells
    0 Failed blackbox cells

    0 Successful equivalence points
  * 1 Failed equivalence points
        1 First priority errors
        0 Second priority errors

NOTE: THIS RUN USED BLACK BOXES - For sign-off LVS running with full equivalence points is recommended
```

Figura 51: Estado de comparación

de instancias o puertos que faltan tanto en *layout* como en esquemático.

```
Diagnostic summary:

    5 missing layout instances
    7 missing layout nets


DIAGNOSTIC: 5 missing layout instances

    The following schematic instances are missing in the layout netlist.

        Schematic instance (type)                    Layout instance
        -------------------------                    --------------------------
        XSAL1 (BLACK_BOX[PDDW0204SC                * Missing instance
          DG])
        XEN1 (BLACK_BOX[PDDW0204SCD                * Missing instance
          G])
        XU6 (BLACK_BOX[TIELBWP7T])                 * Missing instance
        XCom/XU1 (BLACK_BOX[CKND0BW                * Missing instance
          P7T])
        XU7 (BLACK_BOX[TIEHBWP7T])                 * Missing instance


DIAGNOSTIC: 7 missing layout nets

    The following schematic nets are missing in the layout netlist.

        Schematic net : connections                  Layout net
        ---------------------------                  ----------------------
        A_w : 2                                    * Missing net
        n1 : 8                                     * Missing net
        XSAL1/IE : 1                               * Missing net
        Y_w : 2                                    * Missing net
        A : 1                                      * Missing net
        Y : 1                                      * Missing net
        n2 : 3                                     * Missing net
```

Figura 52: Diagnóstico

13.5. LVS para compuerta not con comandos y VUE *Tool*

En la Figura #53 puede observarse que los resultados de comparación fueron exitosos. Se corrió un *Black Box* LVS y puede observarse que las 4 *black boxes* definidas pasaron la prueba, pues esto indica que todas las instancias de los dispositivos se encontraron tanto en *layout* como también en el esquemático. En este caso el punto de equivalencia fue la *top cell*, que para este circuito se le llamó como NOT_IO, siendo este un circuito bastante sencillo.

En la Figura #54 se muestra el archivo de errores en el *layout*, en este caso los errores que está marcando son directamente de ERC. Por parte de LVS no hay ningun error.

Por último, en la Figura #55 puede observarse lo que marca el archivo de resultados. En donde evidentemente pasa la prueba de LVS, pero existen errores en DRC. Es importante mencionar que estos mismos resultados son obtenidos mediante VUE *Tool*, por esto mismo la descripción es exactamente la misma.

```
--------------------------------------------------------------------------------
Final comparison result:PASS

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


TOP equivalence point:
               [Not_IO, Not_IO]

Comparison summary

    4 Successful blackbox cells
    0 Failed blackbox cells

    1 Successful equivalence points
    0 Failed equivalence points

Schematic and layout agree at all user-intended equivalent points involved
in compare.

NOTE: THIS RUN USED BLACK BOXES - For sign-off LVS running with full equivalence points is recommended
```

Figura 53: Resultados de comparación para compuerta not mediante comandos y VUE *Tool*

```
                    LAYOUT ERRORS RESULTS: ERRORS

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

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

Library name:    /home/administrador/Escritorio/LVS/Not/NotFiles/GDS/Not.gds
Structure name:  Not_IO
Generated by:    IC Validator RHEL64 Q-2019.12-SP2-4.5500898 2020/04/25
Runset name:     /usr/synopsys/TSMC/SCRIPTS_NUEVOS/20191128-124344/MAIN_DECK/LVS_RC_ICV_018um_GPIIA_1P6
User name:       administrador
Time started:    2020/09/11 11:26:55PM
Time ended:      2020/09/11 11:27:04PM

Called as: icv -i /home/administrador/Escritorio/LVS/Not/NotFiles/GDS/Not.gds -c Not_IO -s /home/admini
20191128-124344/MAIN_DECK/LVS_RC_ICV_018um_GPIIA_1P6M_v1.4a

                        ERROR SUMMARY

 floating.psub_float: Floating psub_float is not
 allowed
   or ............................................... 5 violations found.



                        ERROR DETAILS


--------------------------------------------------------------------------------
floating.psub_float: Floating psub_float is not allowed
--------------------------------------------------------------------------------

/usr/synopsys/TSMC/SCRIPTS_NUEVOS/20191128-124344/MAIN_DECK/LVS_RC_ICV_018um_GPIIA_1P6M_v1.4a:27557:or
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure    ( lower left x, y ) ( upper right x, y )
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Not_IO       (-0.0050, -0.0050)  (340.1650, 340.1650)
PDDW0204SCDG (0.0000, 0.0000)    (60.0000, 115.0000)
TIELBWP7T    (0.0000, -0.2350)   (1.6800, 4.1550)
TIEHBWP7T    (0.0000, -0.2350)   (1.6800, 4.1550)
CKND0BWP7T   (0.0000, -0.2350)   (1.6800, 4.1550)
```

Figura 54: Errores en *layout* para compuerta not mediante comandos y VUE *Tool*

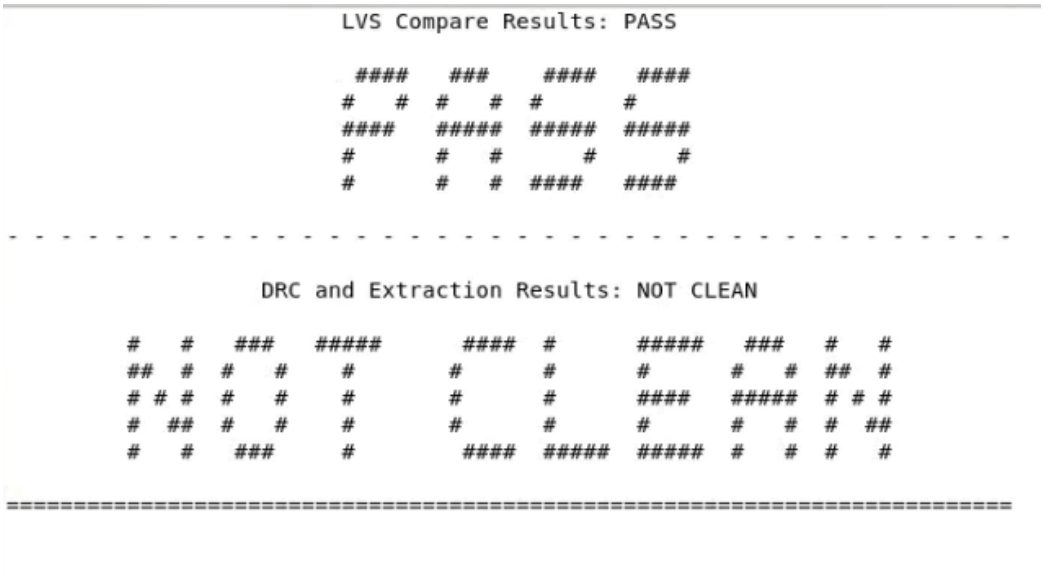


Figura 55: Resultados de LVS para compuerta not mediante comandos y VUE *Tool*

13.6. LVS para compuerta not mediante Custom Compiler

Custom Compiler es una de las herramientas de Synopsys más organizadas en cuanto a la presentación de los resultados de las pruebas. Aparte de tener una interfaz gráfica, muestra los resultados de una manera bastante amigable al usuario y los archivos generados son también presentados directamente en la herramienta después de correr la prueba.

En la Figura #56 puede observarse los resultados generales de la prueba para la compuerta not. En la parte superior se muestran los resultados para ERC. En la parte inferior, los resultados para LVS en donde puede verse que fueron evaluadas 4 *black boxes* y todas fueron correctas.

En la Figura #57 puede observarse la ventana de la VUE *Tool* que se abre al correr la prueba. Como se mencionó anteriormente, esta ventana muestra los resultados de la prueba de una manera bastante amigable y organizada. En el lado izquierdo de la ventana se observa los resultados de las celdas que hicieron *match* entre el *layout* y el esquemático. En el lado derecho de la ventana se observa una tabla que resume los resultados para las celdas. En este caso se encontraron 5 dispositivos y cada uno de ellos hizo *match* entre *layout* y esquemático. Cabe mencionar que en caso se presenten errores, esta misma ventana los presenta en la parte del lado izquierdo en donde se muestra *Unmatched*. Además, en caso de haber errores pueden verse directamente en el *layout*.

Por último, en la Figura #58 puede observarse la ventana que se muestra al correr la prueba. Acá mismo puede verse que en las pestañas están los archivos importantes generados. Por esto mismo correr LVS en Custom Compiler es la mejor manera de hacerlo, los resultados se muestran de una manera amigable y muy organizada.

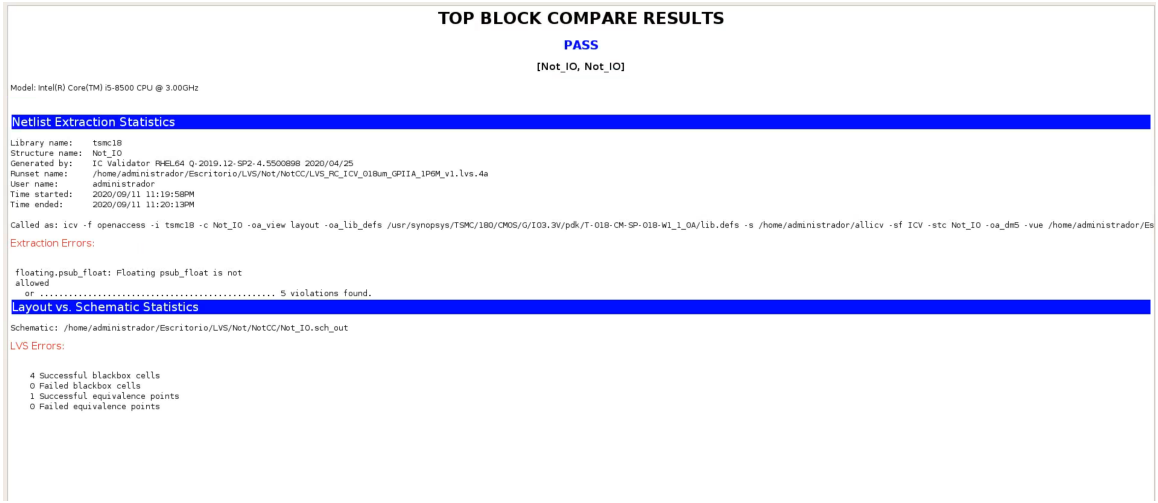


Figura 56: LVS para compuerta not mediante Custom Compiler

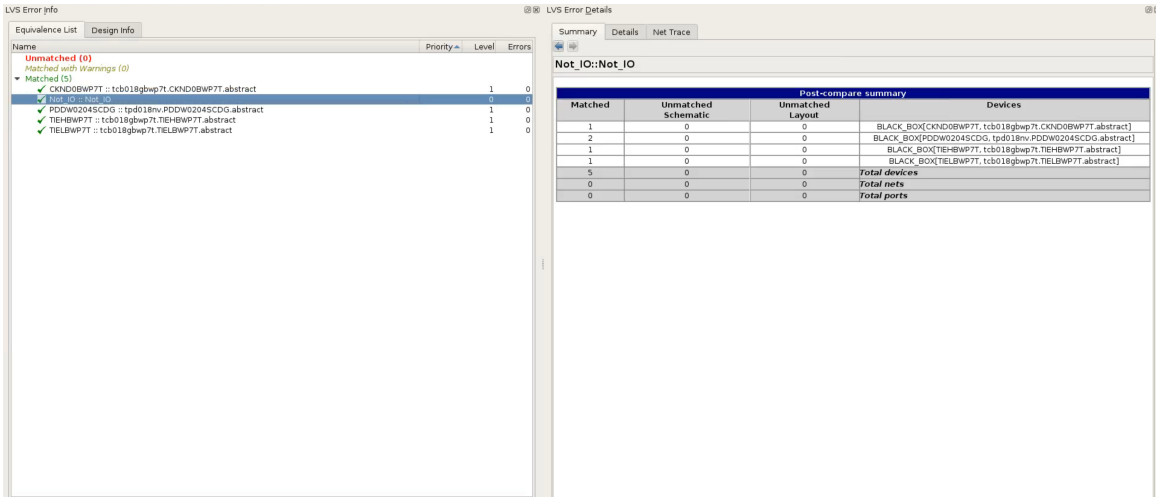


Figura 57: Resultados de LVS para compuerta not mediante Custom Compiler

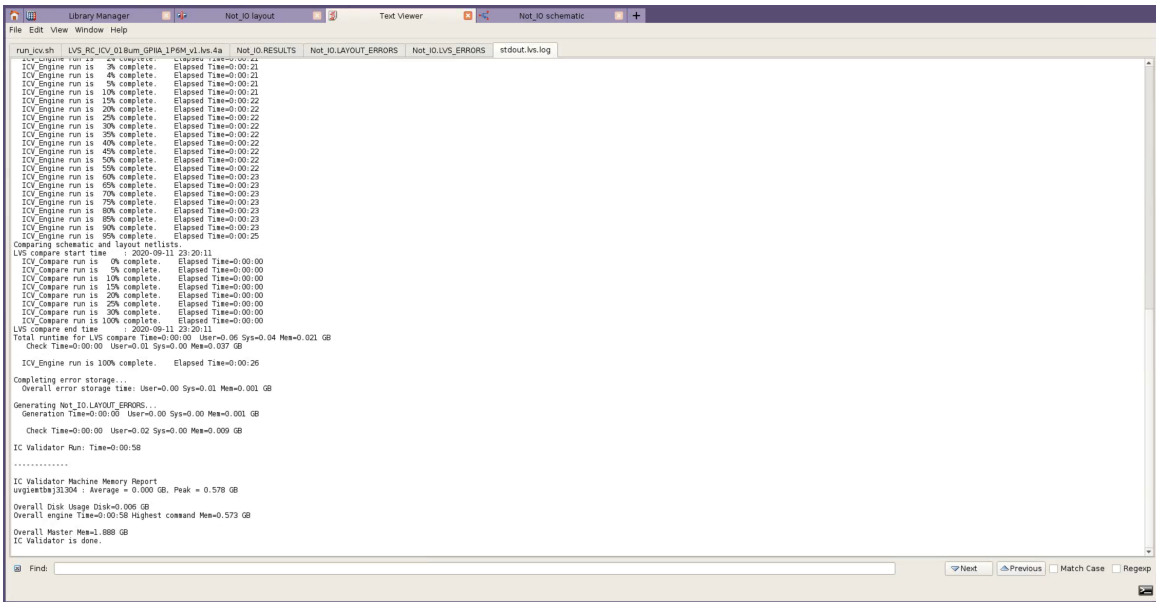


Figura 58: Archivos generados en Custom Compiler para compuerta not

13.7. LVS para *Full Adder* con comandos y VUE *Tool*

En la Figura #59 puede observarse que los resultados de comparación fueron exitosos. Se corrió un *Black Box* LVS y puede observarse que las 6 *black boxes* definidas pasaron la prueba, pues esto indica que todas las instancias de los dispositivos se encontraron tanto en *layout* como también en el esquemático. En este caso el punto de equivalencia fue la *top cell*, que para este circuito se le llamó como Full_Adder, siendo este un circuito con mayor complejidad que una sola compuerta not.

En la Figura #60 se muestra el archivo de errores en el *layout*, en este caso los errores que está marcando son directamente de ERC. Por parte de LVS no hay ningun error.

Por último, en la Figura #61 puede observarse lo que marca el archivo de resultados. En donde evidentemente pasa la prueba de LVS, pero existen errores en DRC. Es importante mencionar que estos mismos resultados son obtenidos mediante VUE *Tool*, por esto mismo la descripción es exactamente la misma.

```
Final comparison result:PASS

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


TOP equivalence point:
                [Full_Adder, Full_Adder]

Comparison summary

    6 Successful blackbox cells
    0 Failed blackbox cells

    1 Successful equivalence points
    0 Failed equivalence points

Schematic and layout agree at all user-intended equivalent points involved
in compare.

NOTE: THIS RUN USED BLACK BOXES - For sign-off LVS running with full equivalence points is recommended
```

Figura 59: Resultados de comparación para *Full Adder* mediante comandos y VUE *Tool*

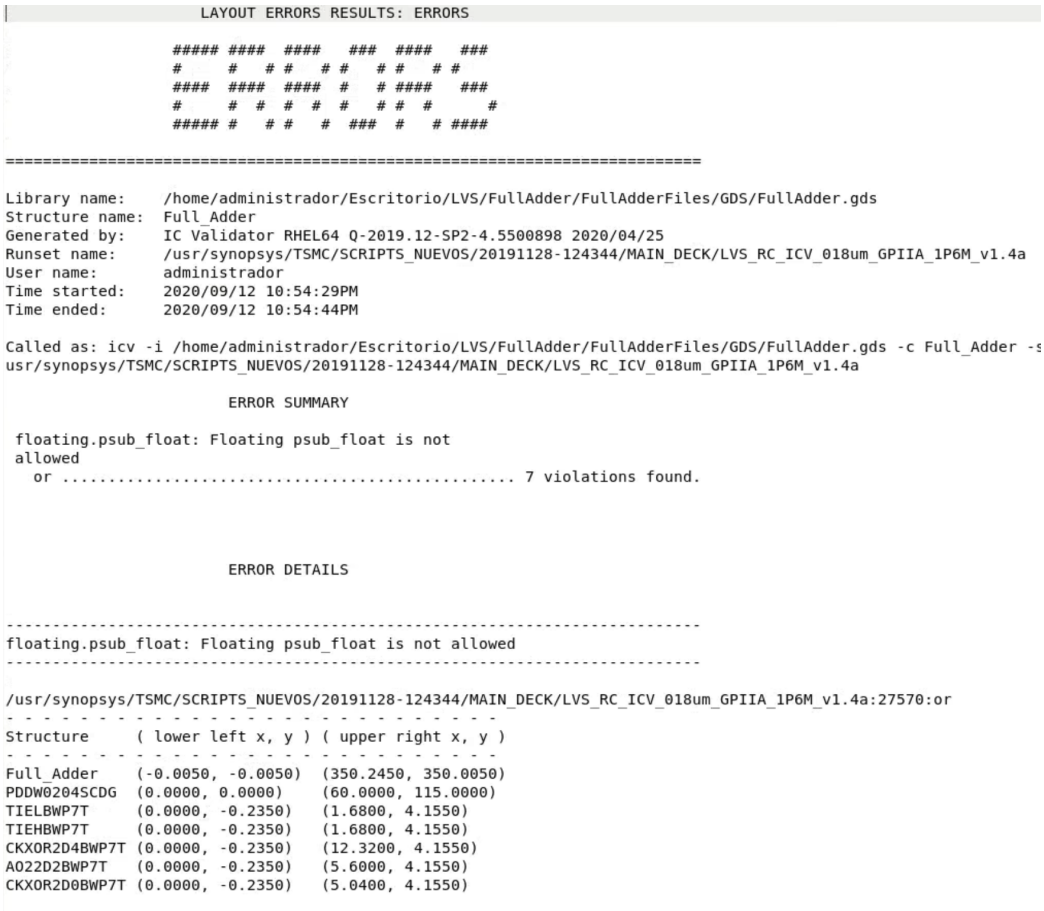


Figura 60: Errores en *layout* para *Full Adder* mediante comandos y VUE *Tool*

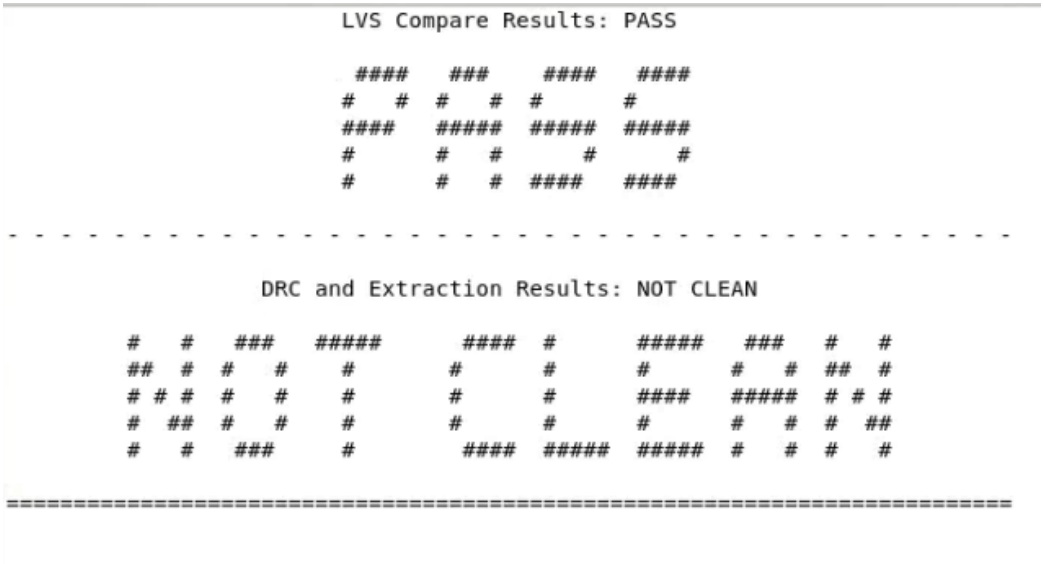


Figura 61: Resultados de LVS para *Full Adder* mediante comandos y VUE *Tool*

13.8. LVS para *Full Adder* mediante Custom Compiler

En la Figura #62 puede observarse los resultados generales de la prueba para el circuito *Full Adder*. En la parte superior se muestran los resultados para ERC. En la parte inferior, los resultados para LVS en donde puede verse que fueron evaluadas 4 *black boxes* y todas fueron correctas.

En la Figura #63 puede observarse la ventana de la VUE *Tool* que se abre al correr la prueba. Como se mencionó anteriormente, esta ventana muestra los resultados de la prueba de una manera bastante amigable y organizada. En el lado izquierdo de la ventana se observa los resultados de las celdas que hicieron *match* entre el *layout* y el esquemático. En el lado derecho de la ventana se observa una tabla que resume los resultados para las celdas. En este caso se encontraron 10 dispositivos y cada uno de ellos hizo *match* entre *layout* y esquemático. Cabe mencionar que en caso se presenten errores, esta misma ventana los presenta en la parte del lado izquierdo en donde se muestra *Unmatched*. Además, en caso de haber errores pueden verse directamente en el *layout*.

Por último, en la Figura #64 puede observarse la ventana que se muestra al correr la prueba. Acá mismo puede verse que en las pestañas están los archivos importantes generados. Por esto mismo correr LVS en Custom Compiler es la mejor manera de hacerlo, los resultados se muestran de una manera amigable y muy organizada.

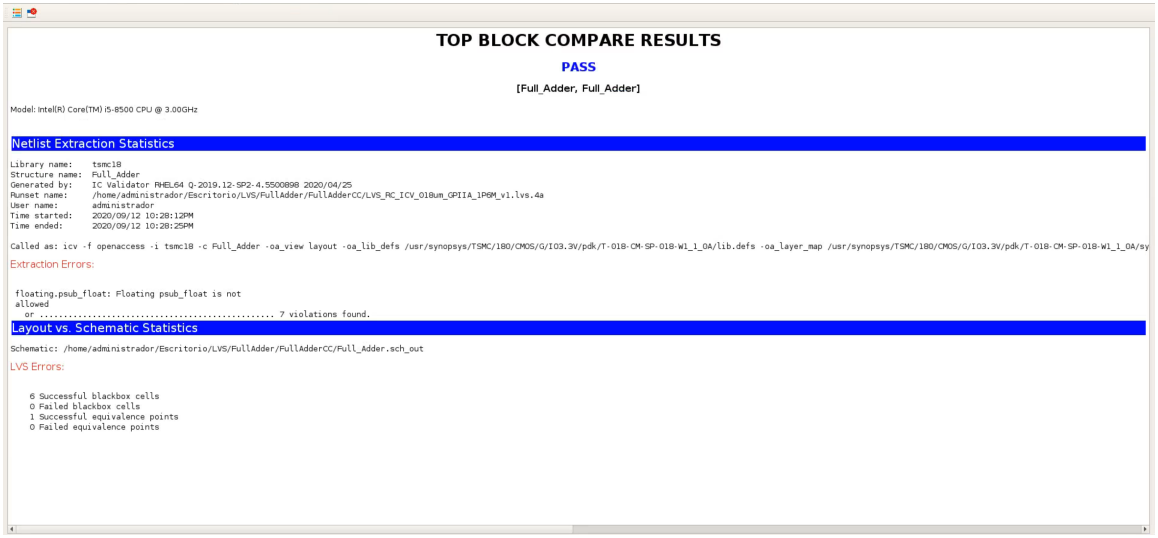


Figura 62: LVS para *Full Adder* mediante Custom Compiler

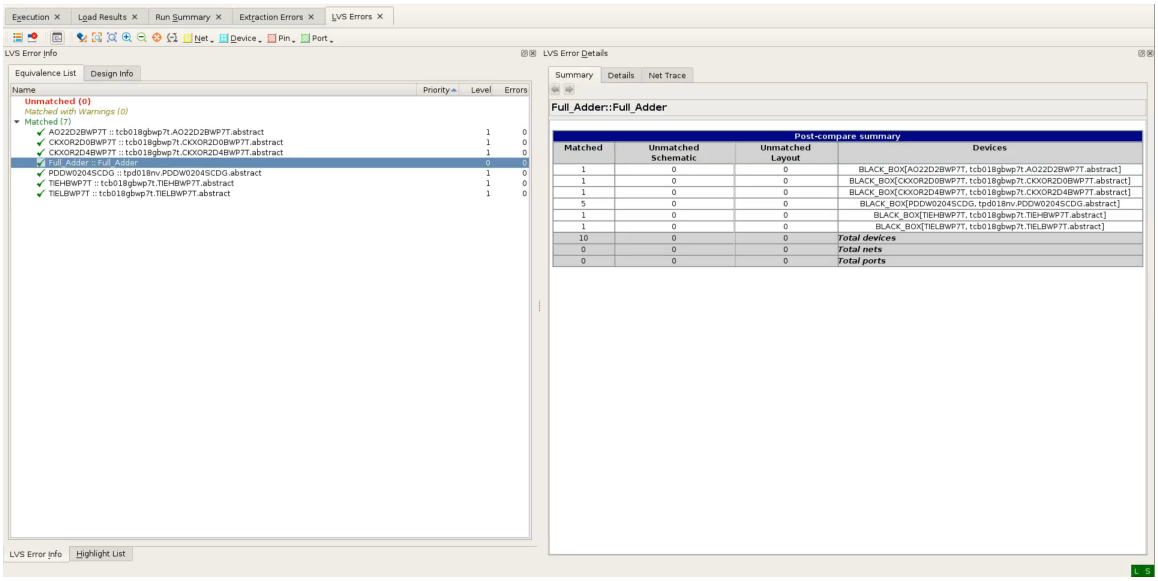


Figura 63: Resultados de LVS para *Full Adder* mediante Custom Compiler

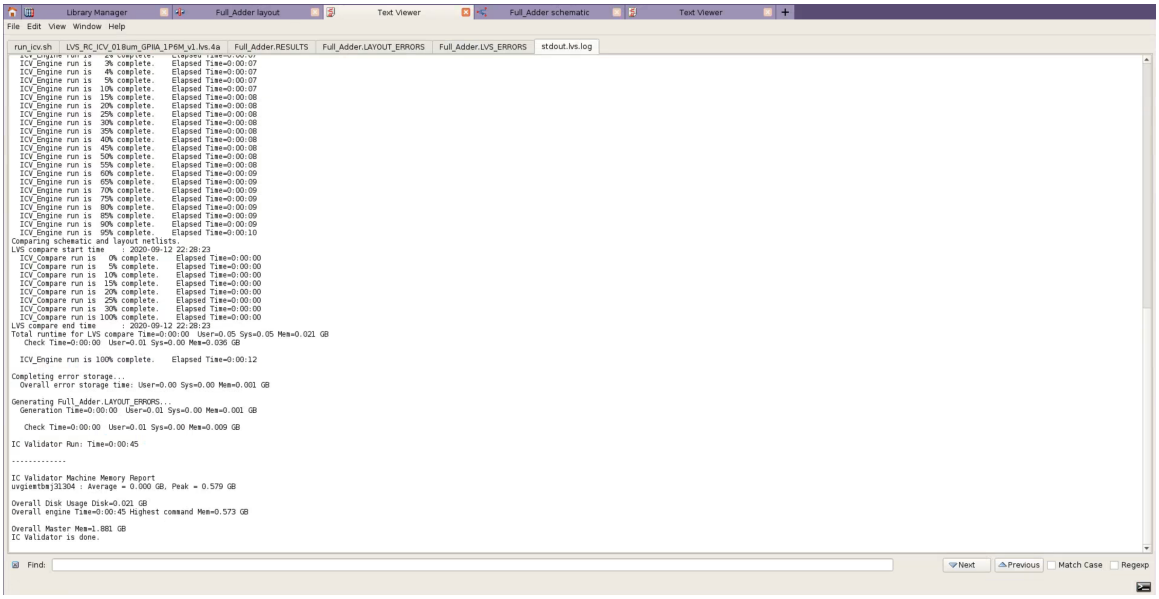


Figura 64: Archivos generados en Custom Compiler para *Full Adder*

13.9. LVS para *Ripple Carry Adder* con comandos y VUE *Tool*

En la Figura #65 puede observarse que los resultados de comparación fueron exitosos. Se corrió un *Black Box* LVS y puede observarse que las 5 *black boxes* definidas pasaron la prueba, pues esto indica que todas las instancias de los dispositivos se encontraron tanto en *layout* como también en el esquemático. En este caso el punto de equivalencia fue la *top cell*, que para este circuito se le llamó como RCA_IO, siendo este un circuito con mayor complejidad que un *Full Adder*.

En la Figura #66 se muestra el archivo de errores en el *layout*, en este caso los errores

que está marcando son directamente de ERC. Por parte de LVS no hay ningun error.

Por último, en la Figura #67 puede observarse lo que marca el archivo de resultados. En donde evidentemente pasa la prueba de LVS, pero existen errores en DRC. Es importante mencionar que estos mismos resultados son obtenidos mediante VUE *Tool*, por esto mismo la descripción es exactamente la misma.

```
LVS error file     = RCA_IO_CORRUPT.LVS_ERRORS
Layout error file  = RCA_IO_CORRUPT.LAYOUT_ERRORS
Schematic netlist  = /home/administrador/Escritorio/LVS/RCA/RCAcmd/RCA_IO_CORRUPT.sch_out
Layout netlist     = /home/administrador/Escritorio/LVS/RCA/RCAcmd/RCA_IO_CORRUPT.net
Runset file        = /usr/synopsys/TSMC/SCRIPTS_NUEVOS/20191128-124344/MAIN_DECK/LVS_RC_ICV_018um_GPIIA
Working directory  = /home/administrador/Escritorio/LVS/RCA/RCAcmd
Compare directory  = run_details/compare
Compare start time = 2020-09-15 18:05:42


--------------------------------------------------------------------------------
Final comparison result:PASS

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


TOP equivalence point:
               [RCA_IO, RCA_IO_CORRUPT]

Comparison summary

    5 Successful blackbox cells
    0 Failed blackbox cells

    1 Successful equivalence points
    0 Failed equivalence points

Schematic and layout agree at all user-intended equivalent points involved
in compare.

NOTE: THIS RUN USED BLACK BOXES - For sign-off LVS running with full equivalence points is recommended
```

Figura 65: Resultados de comparación para *Ripple Carry Adder* mediante comandos y VUE *Tool*

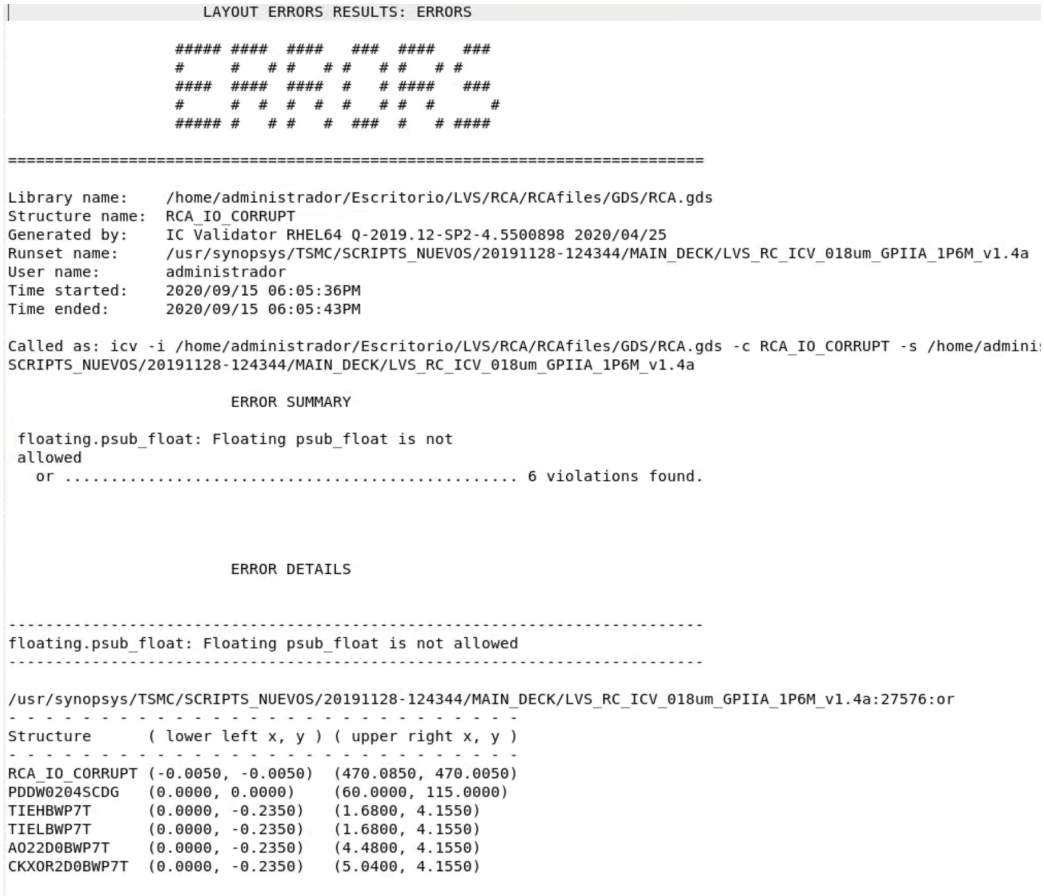


Figura 66: Errores en *layout* para *Ripple Carry Adder* mediante comandos y VUE *Tool*

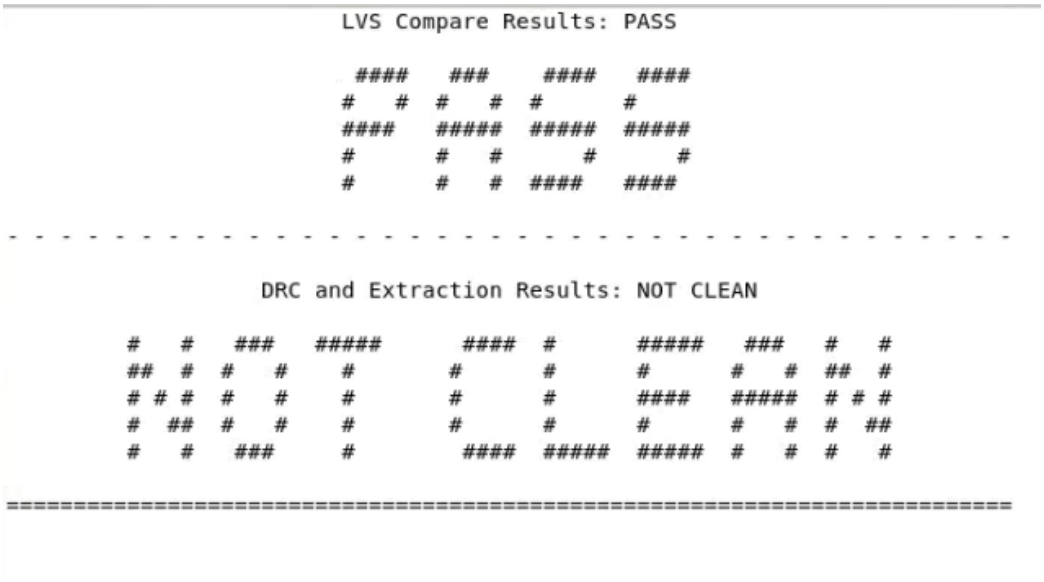


Figura 67: Resultados de LVS para *Ripple Carry Adder* mediante comandos y VUE *Tool*

13.10. LVS para *Ripple Carry Adder* mediante Custom Compiler

En la Figura #68 puede observarse los resultados generales de la prueba para el circuito *Ripple Carry Adder*. En la parte superior se muestran los resultados para ERC. En la parte inferior, los resultados para LVS en donde puede verse que fueron evaluadas 5 *black boxes* y todas fueron correctas.

En la Figura #69 puede observarse la ventana de la VUE *Tool* que se abre al correr la prueba. Como se mencionó anteriormente, esta ventana muestra los resultados de la prueba de una manera bastante amigable y organizada. En el lado izquierdo de la ventana se observa los resultados de las celdas que hicieron *match* entre el *layout* y el esquemático. En el lado derecho de la ventana se observa una tabla que resume los resultados para las celdas. En este caso se encontraron 29 dispositivos y cada uno de ellos hizo *match* entre *layout* y esquemático. Cabe mencionar que en caso se presenten errores, esta misma ventana los presenta en la parte del lado izquierdo en donde se muestra *Unmatched*. Además, en caso de haber errores pueden verse directamente en el *layout*.

Por último, en la Figura #70 puede observarse la ventana que se muestra al correr la prueba. Acá mismo puede verse que en las pestañas están los archivos importantes generados. Por esto mismo correr LVS en Custom Compiler es la mejor manera de hacerlo, los resultados se muestran de una manera amigable y muy organizada.

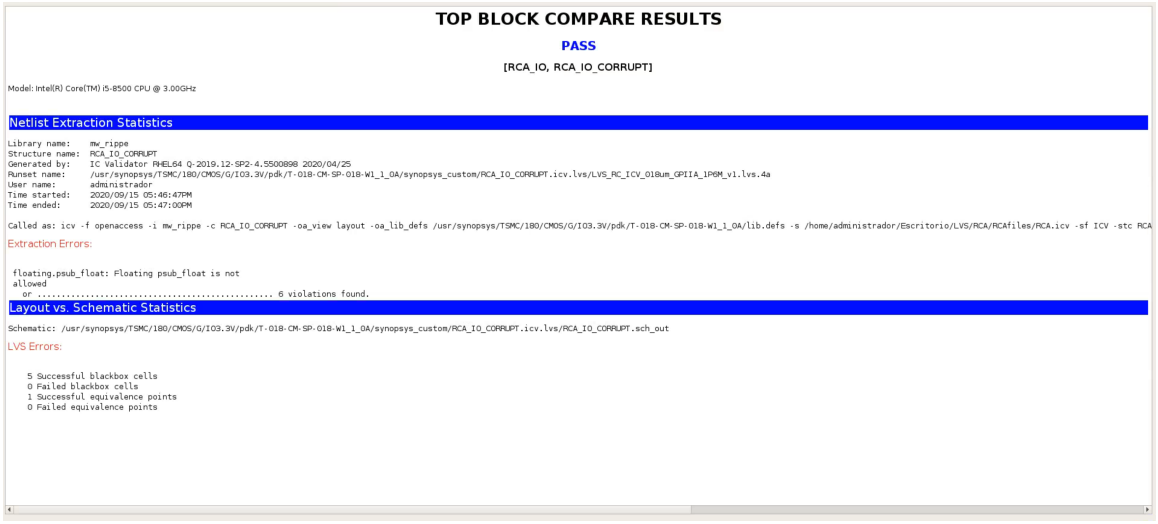


Figura 68: LVS para *Ripple Carry Adder* mediante Custom Compiler

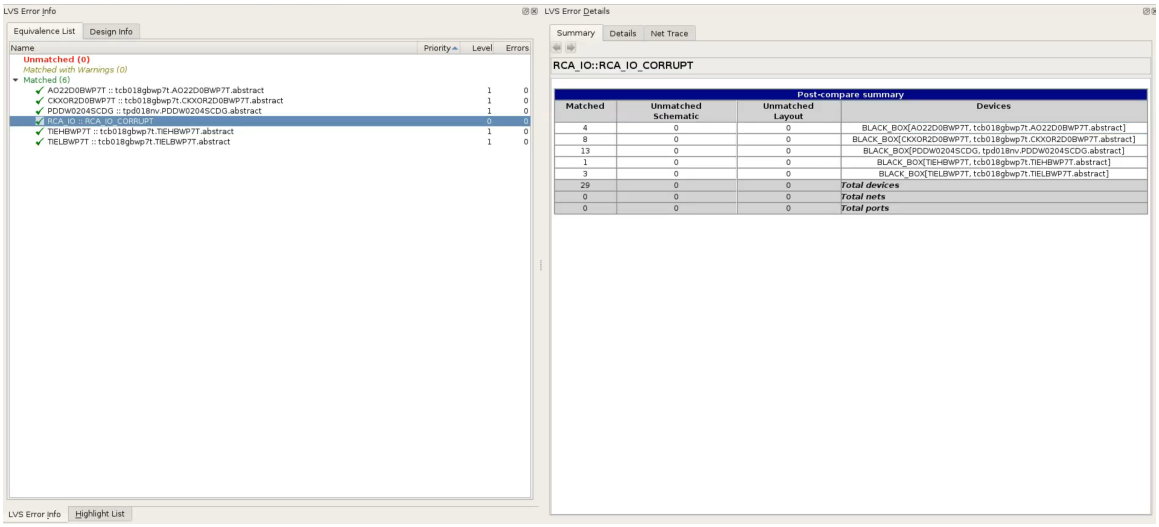


Figura 69: Resultados de LVS para *Ripple Carry Adder* mediante Custom Compiler

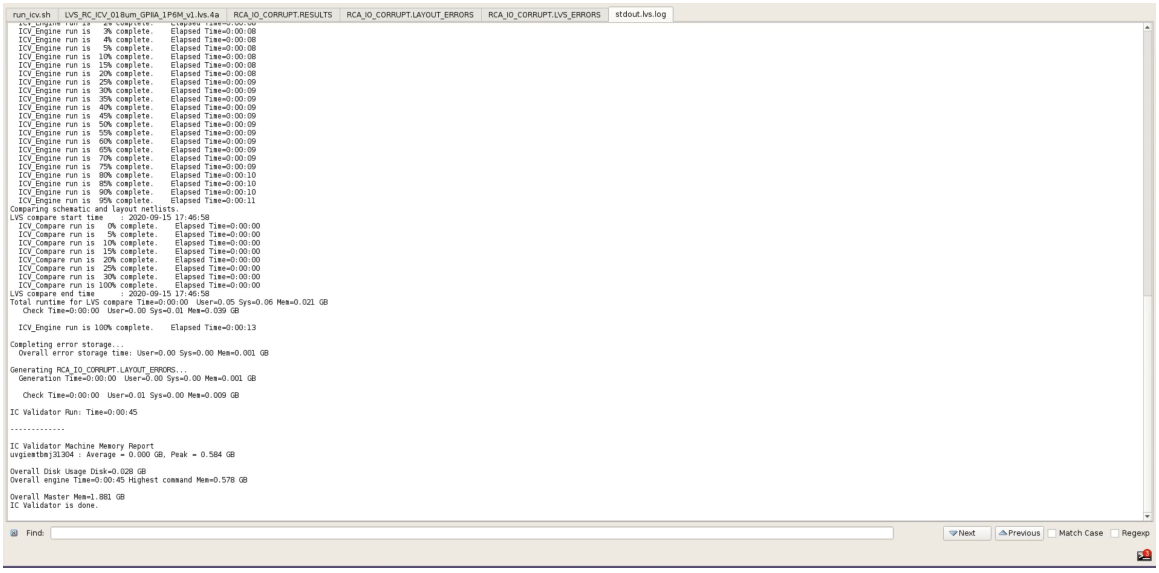


Figura 70: Archivos generados en Custom Compiler para *Ripple Carry Adder*

CAPÍTULO 14

Conclusiones

- Mediante las pruebas realizadas se comprobó que las herramientas Custom Compiler, VUE *Tool* y NetTran *Tool* presentan las características suficientes y necesarias para la verificación física de *Layout vs. Schematic* (LVS).
- Se comprobó que la descripción incompleta de celdas en las librerías proporcionadas presenta limitaciones al correr LVS causando que sea necesario hacer uso de *Black Box* LVS. La comparación de celdas se lleva a cabo con representaciones en caja negra de las mismas.
- Se comprobó que la herramienta de NetTran *Tool* es indispensable para la traducción del *netlist* de esquemático al formato correcto y que este pueda ser sometido a comparación.
- Se comprobó formato ICV de *netlists* es necesario en el flujo de LVS ya que describe jerárquicamente a un circuito y permite llevar a cabo la comparación de dos *netlists* de manera jerárquica.
- Se documentó el uso de las herramientas Custom Compiler, VUE *Tool* y NetTran *Tool* para poder correr LVS mediante el presente trabajo y videotutoriales.

CAPÍTULO 15

Recomendaciones

Este es un proyecto innovador para Guatemala. Varios de los trabajos contienen información de procesos que se llevan a cabo por primera vez en el país. Las herramientas utilizadas en el proyecto son altamente confidenciales y por esta razón la única manera de aprender a usar dichas herramientas es mediante la lectura de documentación de las mismas. Se recomienda a futuras personas involucradas en proyectos de nanoelectrónica que implique uso del flujo de diseño que utilicen los manuales de las herramientas como fuente principal para toma de decisiones y manejo de herramientas. Estos manuales contienen lo necesario para el uso correcto de las herramientas.

Se recomienda también la actualización tanto de herramientas como de documentación de las mismas. Esto porque en algunos casos los comandos para las herramientas se actualizan y también pueden traer nuevas funciones que podrían ser útiles.

Se recomienda que para el proceso de *Layout vs. Schematic* (LVS) se haga uso de los manuales [8] [14] [16]. En este trabajo se presentaron tres maneras distintas de correr LVS, pero la más recomendable es mediante Custom Compiler. Esto ya que esta herramienta presenta los resultados de una manera más amigable y ordenada que en otros casos. Si se utilizan comandos o VUE *Tool* para correr LVS, se recomienda el uso de carpetas que organicen los archivos de salida como está descrito en las guías de este trabajo. Esto ya que es una manera ordenada de obtener los archivos de salida de LVS y que estos mismos no causen confusión en la interpretación de los mismos.

Se recomienda al estudiante leer los trabajos de graduación de años anteriores. Esto ya que contienen información valiosa del flujo de diseño para un chip a escala nanométrica.

CAPÍTULO 16

Bibliografía

[1] N. Weste y D. Harris, *CMOS VLSI Design : A circuits and systems perspective*. Pearson India, 2015, ISBN: 9789332559042. dirección: https://books.google.com.gt/books?id=0RAwDwAAQBAJ.

[2] A. Kahng, J. Lienig, I. Markov y J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer Netherlands, 2011, ISBN: 9789048195916. dirección: https://books.google.com.gt/books?id=DWUGHyFVpboC.

[3] M. Birnbaum, *Essential Electronic Design Automation (EDA)*, ép. Prentice Hall Modern Semicondu. Prentice Hall PTR/Pearson Education, 2004, ISBN: 9780131828292. dirección: https://books.google.com.gt/books?id=IMVS2OsvuPQC.

[4] V. Bagad, *Fundamentals Of Cmos Vlsi*. Technical Publications, 2009, ISBN: 9788184317541. dirección: https://books.google.com.gt/books?id=mknM%5C_nqqiAEC.

[5] J. A. Santos Chonay, "Diseño de un sumador/restador de 32 bits con tecnología CMOS en un proceso de 28 nanómetros usando aplicaciones de diseño de la empresa Synopsys", Tesis de licenciatura Ingeniería Electrónica, Universidad del Valle de Guatemala, 2014.

[6] L. A. Nájera Vásquez, "Implementación de circuitos sintetizados a nivel netlist a partir de un diseño en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado", Tesis de licenciatura Ingeniería Electrónica, Universidad del Valle de Guatemala, 2019.

[7] S. H. Rubio Vásquez, "Definición del Flujo de Diseño para Fabricación de un Chip con Tecnologia VLSI CMOS", Tesis de licenciatura Ingeniería Electrónica, Universidad del Valle de Guatemala, 2019.

[8] Synopsys, *IC Validator LVS User Guide Q-2019.12*. SolvNet, 2020.

[9] L. Xiu, *VLSI Circuit Design Methodology Demystified: A Conceptual Taxonomy*. Wiley, 2007, ISBN: 9780470199107. dirección: https://books.google.com.gt/books?id=w98VY%5C_P7sUkC.

[10] D. Jansen, *The Electronic Design Automation Handbook*. Springer US, 2010, ISBN: 9780387735436. dirección: https://books.google.com.gt/books?id=br3gBwAAQBAJ.

[11] D. Doman, *Engineering the CMOS Library: Enhancing Digital Design Kits for Competitive Silicon*, ép. A John Wiley et Sons, Inc., publication. Wiley, 2012, ISBN: 9781118243046. dirección: `https://books.google.com.gt/books?id=XmIEset3uGYC`.

[12] Synopsys, *Custom Compiler Datasheet*. 2018. dirección: `https://www.synopsys.com/implementation-and-signoff/custom-design-platform/custom-compiler.html`.

[13] G. M. imec, *HOW TO CREATE A CDL NETLIST FROM VERILOG [Application Note 006]*. 2010.

[14] Synopsys, *IC Validator Reference Manual Q-2019.12*. SolvNet, 2020.

[15] G. M. imec, *HOW TO RUN A BLACK BOX LVS [Application Note 017]*. 2011.

[16] Synopsys, *IC Validator User Guide Q-2019.12*. SolvNet, 2020.

CAPÍTULO 17

Anexos

CAPÍTULO 18

Glosario

Behavioral se refiere a una forma de modelar el diseño de hardware en base a su funcionalidad. 11

Micrón unidad de medida utilizada para especificar reglas de diseño industrial. Equivale a $10^{-6}m$. 15

Milkyway se refiere a la librería que guarda el diseño en silicio del circuito obtenido mediante la síntesis física. 43

Nanoboard se refiere a *Field-Programmable Gate Array*(FPGA) utilizada para la implementación y depuración de diseños FPGA. Las NanoBoards se encargan de establecer comunicación hacia y desde la FPGA y también contienen una serie de periféricos que incluyen LCD, memoria flash, RAM, teclado y otros utilizados para el desarrollo de aplicaciones. 12

Nets se refiere a una colección de dos o más componentes interconectados. 14

Runset se refiere al conjunto de reglas y comandos para ejecutar un proceso o un conjunto de procesos específico. 17

Structural se refiere a una forma de modelar el diseño de hardware en donde se conectan diferentes partes para obtener el diseño final. 11

TSMC se refiere a *Taiwan Semiconductor Manufacturing Company* empresa de fundición de semiconductores más grande del mundo. 34