UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

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.

Trabajo de graduación en modalidad de tesis presentado por
Luis Arturo Nájera Vásquez
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,
2019

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

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.

Trabajo de graduación en modalidad de tesis presentado por
Luis Arturo Nájera Vásquez
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,
2019

Vo.Bo.:

(f) _____

M.Sc. Carlos Esquit

Tribunal Examinador:

(f) _____

M.Sc. Carlos Esquit

(f) _____

Ing. Jonathan de los Santos

(f) _____

Ing. Guilmar Escobar

Fecha de aprobación: Guatemala, 06 de diciembre del 2019.

Agradecimientos

A mi familia, por haberme dado la oportunidad de formarme en esta universidad y apoyarme durante este tiempo.

A la Universidad del Valle de Guatemala, los docentes de la carrera de Ingeniería en Electrónica, en especial a mi director de tesis M.Sc. Carlos Esquit por promover esta investigación y ayudarnos a cumplir esta meta.

Índice

Lista de figuras

Lista de tablas

Resumen

Este trabajo se enfoca en los primeros pasos del flujo de diseño para la elaboración de un circuito nanométrico con tecnología *CMOS*. El objetivo principal de este trabajo es la elaboración de circuitos en lenguaje descriptivo de *hardware* para su posterior mapeo a *netlists* de compuertas lógicas por medio de una síntesis, y su verificación de funcionamiento tanto antes como después de la síntesis.

La herramienta *Design Vision* fue instalada, estudiada y utilizada para la elaboración de la síntesis junto a las librerías de 90nm brindadas por Synopsys. De igual forma, las herramientas *Formality* y *VCS* fueron instaladas y utilizadas para corroborar que la síntesis se realizara de forma correcta y simular el comportamiento del circuito tanto antes como después de la síntesis.

Diversas pruebas de síntesis fueron realizadas utilizando distintos circuitos para comprender el funcionamiento de la herramienta. La obtención del software, librerías, su uso y características se resumen en este documento con el objetivo de guiar a futuros estudiantes en el desarrollo de diseños de circuitos nanométricos de mayor complejidad. Se elaboró una serie de vídeos explicativos para guiar a futuros estudiantes. Estos vídeos serán provistos al departamento de Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala.

CAPÍTULO 1

Introducción

A grandes rasgos la nanoelectrónica actual consiste en el estudio, diseño, simulación y fabricación de circuitos integrados. Este tipo de circuitos se caracterizan por ser elaborados a partir de una oblea o lamina de silicio la cual se somete a un proceso químico con el objetivo de crear transistores tipo *MOSFET* en su interior. [19]

Millones o incluso miles de millones de transistores pueden ser elaborados en un pequeño segmento de silicio de unos pocos milímetros, posteriormente estos son interconectados entre sí para generar circuitos de pequeñas dimensiones los cuales pueden cumplir con tareas tanto básicas como altamente complejas. [19]

La posibilidad de elaborar este tipo de circuitos conformados por una basta cantidad de transistores se debe a los softwares de diseño asistido creados en los últimos años y a las dimensiones que presentan los transistores, siendo estos de unos pocos nanómetros. [1]

Para concebir la fabricación de un circuito integrado debe presentarse un trabajo previo de diseño del circuito. El diseño de los circuitos nanométricos se trabaja en softwares especiales que permiten elaborar y verificar el circuito que se busca realizar a través de simulaciones y una variedad de herramientas. Entre los softwares mayormente utilizados a nivel mundial se encuentran Synopsys y Cadence. [1]

Se conoce como flujo de diseño a los pasos que deben de seguirse para realizar el diseño de un circuito nanométrico. Distintos flujos de diseño han sido creados en los últimos años y se utilizan dependiendo del tipo de circuito que se busque elaborar. Los flujos de diseño se clasifican como: *ASIC* o *Custom* (o personalizados). [4]

ASIC (*Application Specific Integrated Circuit*) se divide en distintas ramas, sin embargo, este se caracteriza por ser utilizado en la elaboración de circuitos integrados que cumplan con una tarea específica. Este suele estar relacionado a un proceso de diseño semi automatizado donde distintas herramientas son utilizadas para elaborar el diseño, conexiones, verificación de funcionamiento y demás características del circuito. [4]

Los circuitos nanométricos personalizados pueden definirse como circuitos elaborados para cumplir con una o varias tareas y en donde su flujo de diseño se basa en elaborar cada elemento del circuito de forma manual. [4]

Entre las principales ventajas del sistema *ASIC* se encuentra la posibilidad de realizar el diseño

de un circuito integrado en menor tiempo, debido a que no se requiere la elaboración detallada de cada segmento del circuito. Esto permite elaborar, verificar y obtener el diseño en el transcurso de meses (dependiendo del circuito a elaborar). [4]

Por el otro lado, un diseño personalizado suele requerir un mayor tiempo, generalmente años, y presenta una mayor dificultad pues busca elaborar un diseño totalmente nuevo, enfocándose en un bajo consumo energético o alto desempeño con la finalidad de generar un circuito integrado para su comercialización. [4]

Con el motivo de elaborar un flujo de diseño para un circuito integrado para su posterior fabricación se dio origen a este proyecto, siendo una iniciativa del departamento de Mecatrónica, Electrónica y Biomédica de la Universidad del Valle de Guatemala.

El circuito a diseñar tendrá la finalidad de generar un mensaje el cual será interpretado por una computadora. Para su elaboración el proyecto fue dividido en cuatro módulos, este trabajo presenta el primer módulo del proyecto y se enfoca en la elaboración del circuito en lenguaje descriptivo de *hardware*, verificación de su funcionamiento y síntesis a un *netlist* de compuertas lógicas.

La Universidad del Valle de Guatemala presenta la posibilidad de utilizar el software de diseño Synopsys, por lo que a lo largo de este documento se presentarán las herramientas utilizadas, su funcionamiento y resultados.

CAPÍTULO 2

Objetivos

2.1. Objetivo general

Implementar la arquitectura para el diseño de un circuito integrado con tecnología nanométrica.

2.2. Objetivos específicos

- Implementar la arquitectura del circuito a realizar por medio de lenguaje descriptivo verilog para realizar un primer paso en la elaboración del mapeo del circuito de lenguaje descriptivo verilog a un *netlist* de compuertas lógicas.
- Determinar, instalar y estudiar la herramienta de Synopsys que permita el mapeo de lenguaje descriptivo verilog a un *netlist* de compuertas lógicas para utilizar esta como medio para la elaboración del mapeo del circuito.
- Realizar el mapeo del circuito en lenguaje descriptivo verilog a un *netlist* de compuertas lógicas para obtener el diseño del circuito elaborado con componentes lógicos referenciados en las librerías que se utilicen.
- Comprobar el funcionamiento del circuito tanto antes como después de ser mapeado para poder asegurar que el circuito se comporta de la forma deseada.
- Documentar el funcionamiento de la herramienta utilizada por medio de videotutoriales y documentos escritos para su uso como referencia por parte de los futuros estudiantes de la Universidad del Valle de Guatemala.

CAPÍTULO 3

Justificación

Elaborar el flujo completo para el diseño del chip de un circuito integrado tiene implicaciones importantes, con esto ya no se depende de los chips que existen en el mercado que son, en su mayoría, de propósito general. El flujo permite hacer un chip para una aplicación específica, esto implica que el desempeño de este en esa aplicación será mucho más alto que utilizando chips de propósito general. Como consecuencia, este trabajo sienta las bases para la elaboración de proyectos únicos y el desarrollo de tecnología en Guatemala, ya que con esto se pueden abrir brechas laborales que se enfoquen en el diseño de chips para atacar problemas puntuales, creando así un nuevo mercado de negocios en el que también se pueda atraer inversión extranjera. [5]

El beneficio directo de este flujo es la facilidad que esto dará al momento de diseñar un chip. El tiempo se podrá invertir en aprender a utilizar herramientas y librerías nuevas, ya que la base estará lo suficientemente documentada. [5]

CAPÍTULO 4

Marco teórico

4.1. Transistores *MOSFET*

Los transistores pueden ser definidos como interruptores controlados eléctricamente, los cuales poseen tres terminales. Una terminal de control, y dos terminales que se encontrarán conectadas o desconectadas dependiendo de la tensión aplicada a la terminal de control. [19]

En las últimas tres décadas los transistores tipo *MOSFET* (*Metal-oxide-semiconductor Field-effect transistor*) se han posicionado como los dispositivos electrónicos más utilizados a nivel mundial, esto debido a que son el elemento fundamental que da forma a los circuitos integrados. [19]

En los años 1950 *Texas Instrument* comenzó con el desarrollo de circuitos integrados a partir de unos pocos transistores. En 2018, Xilinx introdujo un novedoso *FPGA* elaborado a partir de 50 mil millones de transistores *MOSFET*. [19][10]

El motivo de este cambio tan drástico se debe a una reducción en el tamaño de los transistores, desde su creación las dimensiones de los transistores *MOSFET* han disminuido un promedio de 50 % cada dos años desde hace más de 50 años. [19]

Los transistores *MOSFET* consiguieron posicionarse como el elemento fundamental de la electrónica nanométrica debido a su bajo costo de producción, la característica de ocupar poca área y la relativa simplicidad de su fabricación. [19]

Su nombre proviene de *Metal-oxide-semiconductor* (*MOS*), los materiales utilizados para crear el transistor. Y *Field-effect-transistor* (*FET*), la forma en que este funciona. [19]

El silicio se presenta como la base para los transistores *MOSFET*, este semiconductor posee una estructura atómica la cual forma enlaces covalentes con cuatro átomos adyacentes. El silicio en su estado puro se presenta como un mal conductor debido a que no posee electrones de valencia libres, ya que estos se encuentran enlazados químicamente (Figura #4.1(a)). Una forma de aumentar su conductividad se presenta al introducir un grupo de impurezas dentro del silicio, estas impurezas se conocen como dopantes. Al doparlo con un elemento del Grupo V de la tabla periódica es posible introducir nuevos átomos los cuales posean cinco electrones de valencia libres, cuatro de ellos establecerán enlaces covalentes mientras que uno quedara libre. Al dopar de esta forma el silicio este se clasifica como un segmento tipo-n (Figura #4.1(b)). [19]

Un segundo tipo de dopaje es realizado al introducir átomos del Grupo III de la tabla periódica dentro del silicio, estos átomos solo poseen tres electrones de valencia, los cuales formaran enlaces covalentes con los átomos de silicio, al no existir un cuarto electrón que forme el enlace requerido se formara un agujero. El agujero actúa como una carga positiva, por lo que este segmento se clasifica como tipo-p (Figura #4.1(c)). [19]

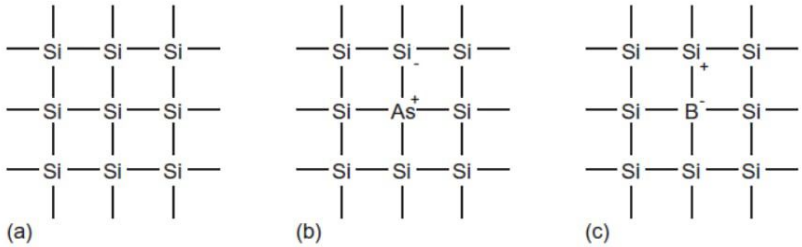


Figura 4.1: (a) Estructura de silicio sin dopaje. (b) Estructura al dopar el silicio con arsénico. (c) Estructura al dopar el silicio con boro. [19]

Al realizar una unión entre un segmento de silicio dopado tipo-n y un segmento de silicio dopado tipo-p el resultado es conocido como un diodo (Figura #4.2). Cuando un voltaje es colocado con su terminal positiva en el segmento tipo-p y el voltaje negativo en el segmento tipo-n la corriente fluye de un punto a otro. En caso de colocar un voltaje con su terminal positiva en el segmento tipo-n y el voltaje negativo en el segmento tipo-p se dice que el diodo esta en inversa, y una corriente sumamente pequeña fluirá de un punto a otro. Este comportamiento se debe a sus distintos dopajes, estructuras atómicas y como los electrones se comportan en cada caso. [19]

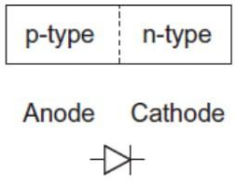


Figura 4.2: Estructura de un diodo. [19]

Para elaborar un transistor tipo *MOSFET* se utiliza un proceso similar, colocando uniones entre distintos tipos de segmento, ya sea dos segmentos tipo-n y un segmento tipo-p, o viceversa. Además de esto, se coloca una fina capa de oxido SiO_2 y un segmento de polisilicio para elaborar una zona que actuara como terminal de control. Dependiendo de los dopajes los transistores *MOSFET* pueden clasificarse como transistores tipo-n o tipo-p (Figura #4.3). [19]

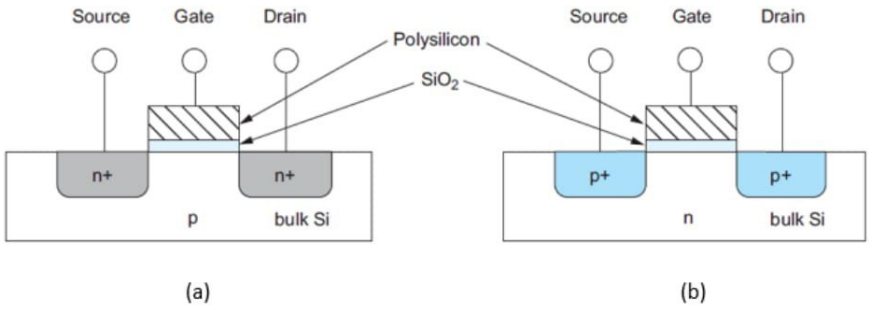


Figura 4.3: (a) Transistor tipo-n. (b) Transistor tipo-p. [19]

4.1.1. Funcionamiento de transistores

Los transistores *MOSFET* se encuentran conformados por cuatro segmentos: *Source*, *Gate*, *Drain* y *Buck*. Estos segmentos pueden observarse en la Figura #4.3. El funcionamiento de los transistores se basa en la diferencia de potenciales entre estos segmentos. De forma típica los transistores tipo n poseen su *Buck* conectado a tierra, mientras que los transistores tipo-p poseen su *Buck* conectado a voltaje positivo. [19]

El motivo de esto se presenta en la compuerta *Gate*. Esta compuerta tiene la finalidad de ser una terminal de control, la cual permitirá o no el flujo de electrones entre las compuertas *Source* y *Drain*. [19]

Si tomamos un transistor tipo-n el cual posee su *Buck* conectado a tierra y colocamos la compuerta *Gate* a un voltaje positivo se creará un campo eléctrico y un diferencial de potencial entre estos segmentos del transistor, provocando que cargas negativas se muevan a la parte superior y cargas positivas a la parte inferior. Al ser atraídas las cargas negativas a la compuerta *Gate* estas formaran un puente entre las compuertas *Source* y *Drain*. Al colocar un voltaje positivo en *Source* y tierra en *Drain* los electrones podrán fluir de un punto al otro del transistor por medio del puente creado. Al eliminar el voltaje en *Gate* los electrones no podrán fluir pues no existe un puente que se los permita. Este comportamiento puede ser observado en la Figura #4.4. [19]

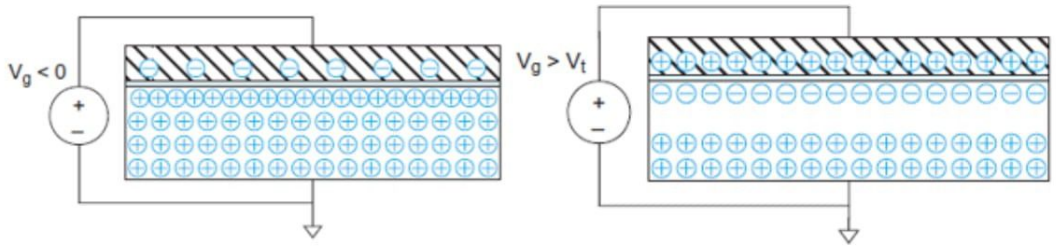


Figura 4.4: Puente de electrones formado en transistores *MOSFET*. [19]

Como se puede observar en la Figura#4.4, el voltaje de *Gate* debe de superar un voltaje Vt para poder formar el puente de electrones. Este voltaje Vt es conocido como Voltaje de *threshold*, y es el voltaje mínimo a superar para que el campo eléctrico sea lo suficientemente fuerte como para atraer a los electrones a la superficie. [19]

Los transistores tipo-p cumplen con el mismo funcionamiento, con la diferencia de que, al estar su *Buck* conectado a voltaje positivo la terminal *Gate* funcionara de forma inversa. Al colocar un voltaje positivo en el *Gate* no existirá una diferencia de potencial debido a que tanto el *Gate* como el *Buck* están a un voltaje positivo, por lo que, para que el canal se forme el *Gate* debe de colocarse a tierra. [19]

4.1.2. Transistores tipo n

Los transistores tipo-n presentan el funcionamiento anteriormente descrito. Los transistores tienen la tarea de colocar a su salida voltajes lógicos *VDD* o tierra. Estos voltajes son considerados como 1 o 0 y permitirán el funcionamiento del circuito. [19]

Al realizar un estudio mas profundo de los transistores tipo-n se presenta la característica de que estos pueden colocar a su salida un valor 0 perfecto al ser necesario, sin embargo, al requerir un voltaje 1 este se degrada. [19]

Esta característica se presenta debido a los voltajes mediante los cuales trabaja. Los transistores

tipo-n serán representados con la simbología mostrada en la Figura#4.5. [19]

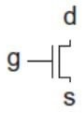


Figura 4.5: Transistor tipo-n. [19]

4.1.3. Transistores tipo p

El comportamiento de los transistores tipo-p a sido descrito anteriormente. Similar a los transistores tipo-n al realizar un estudio del comportamiento presentado por los transistores tipo-p se presenta la característica de obtener un valor 1 perfecto al momento de ser necesario, a la vez que se presenta un 0 degradado. Los transistores tipo-p serán representados con la simbología mostrada en la Figura #4.6. [19]

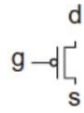


Figura 4.6: Transistor tipo-p. [19]

4.1.4. Estilos de diseño

Un estilo de diseño en el ámbito de la nanoelectrónica hace referencia a las normas utilizadas para elaborar un circuito a partir de transistores. Distintos estilos han sido creados con la finalidad de obtener un mejor desempeño en los circuitos nanométricos. Sin embargo, un estilo se alza por sobre los demás, siendo este la lógica *CMOS*. [19]

Debido a las características presentadas por los transistores tipo-n y tipo-p se desarrolló un conjunto de normas que permiten el uso de estos dos transistores para la elaboración de circuitos. Al utilizar los transistores tipo-p para obtener valores de voltaje 1 y los transistores tipo-n para obtener valores de voltaje 0 se consigue elaborar circuitos que presenten valores lógicos perfectos. [19]

A partir de este conjunto de normas se utiliza el mismo número de transistores tipo-p y tipo-n, la Figura #4.7 muestra algunas compuertas lógicas elaboradas utilizando transistores *MOSFET* y lógica *CMOS*. [19]

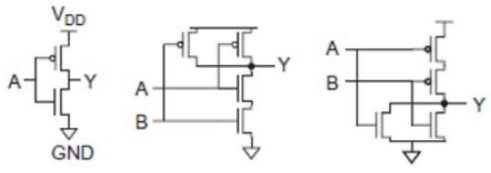


Figura 4.7: Compuetas *Not*, *Nand* y *Nor* respectivamente en lógica *CMOS*. [19]

Con la posibilidad de poder generar compuertas lógicas que operen de forma correcta a partir de transistores *MOSFET* se abre la posibilidad de elaborar circuitos que cumplan con cualquier tipo de tarea requerida. [19]

4.1.5. Fabricación

La fabricación de un circuito nanométrico consiste en un proceso mediante el cual los transistores son elaborados en una oblea de silicio. Consta de distintos pasos donde una variedad de elementos son depositados en la oblea. [19]

El primer paso consiste en obtener la oblea de silicio. El silicio es obtenido a partir de arena, esta es sometida a un proceso mediante el cual se forja un cilindro de silicio puro, el cual es rebanado es delgados discos, cada uno de estos discos será una oblea donde se fabricarán cientos de circuitos integrados. [19]

Figura 4.8: Representación de oblea de silicio. [19]

La oblea es depositada en un horno el cual creará el ambiente óptimo para que una capa de óxido de silicio crezca en su superficie. [19]

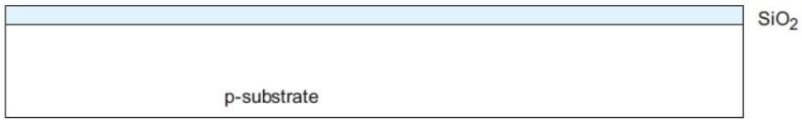


Figura 4.9: Generación de óxido de silicio. [19]

Una vez la oblea posee una fina capa de SiO_2 esta se saca del horno, posteriormente se aplica un fotorresistor en su superficie de forma uniforme. El fotorresistor reacciona a los fotones, por lo que al aplicarse luz al fotorresistor este se suaviza y puede ser removido. [19]

Figura 4.10: Aplicación de fotorresistor. [19]

Al poseer la oblea una capa de SiO_2 y por encima una capa de fotorresistor se utiliza un proceso de litografía. Maquinaria especializada envía fotones por medio de máscaras diseñadas específicamente para cada circuito, estas mascaras permiten el paso de fotones solamente a una zona especifica de la oblea, lo que provoca la eliminación de una porción del fotorresistor. [19]

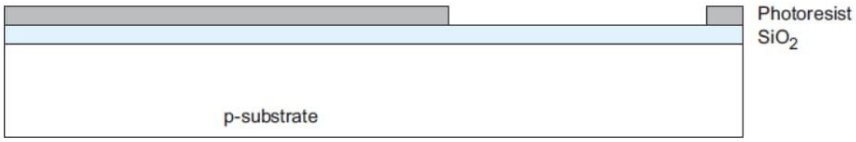


Figura 4.11: Retiro de una porción de fotorresistor [19].

Una vez eliminado el segmento de fotorresistor donde se realizará un dopaje se aplica un ácido a la oblea, el cual reaccionará eliminando el segmento de SiO_2 que no se encuentre recubierto por el fotorresistor, dejando ahora un segmento de la oblea accesible. [19]

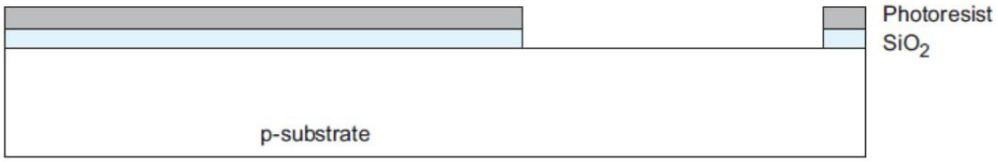


Figura 4.12: Retiro de una porción de óxido de silicio. [19]

El fotorresistor es retirado en su totalidad y la oblea es colocada en un horno donde se rociarán átomos de Arsénico o Boro, los átomos penetrarán solamente en el segmento descubierto de la oblea, dopando un punto especifico de esta. [19]

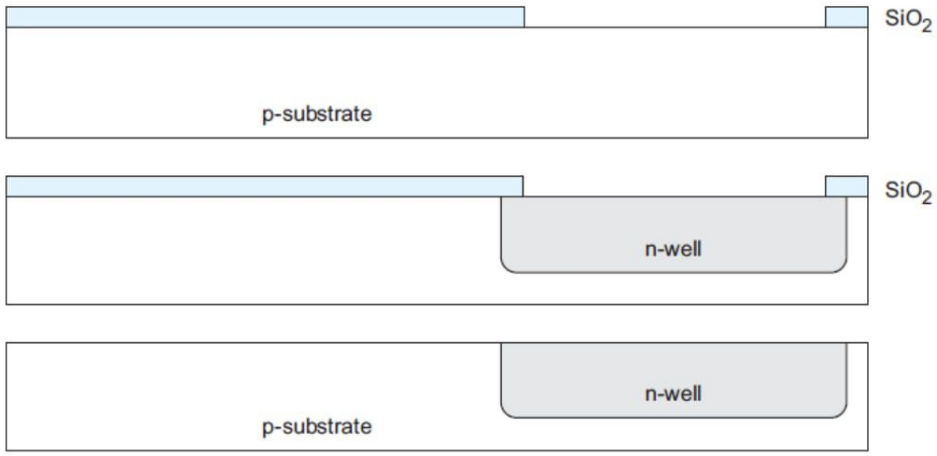


Figura 4.13: Dopaje de la oblea de silicio [19]

Para obtener una división N-P-N o P-N-P que sea simétrica se coloca una fina capa de óxido SiO_2 seguida por segmentos de polisilicio, los cuales posteriormente cumplirán la función de compuertas *Gate*. Estos segmentos de polisilicio permitirán obtener una simetría al momento de realizar los dopajes, a partir de este punto el proceso anteriormente descrito se repite. [19]

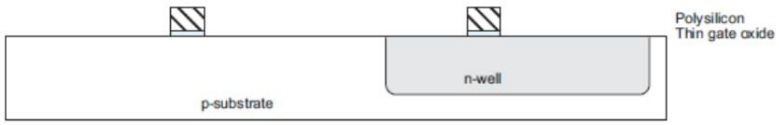


Figura 4.14: Colocación de polisilicio. [19]

Al realizar nuevamente el proceso se finaliza con un dopaje uniforme para el transistor N-P-N, como se observa en la Figura #4.15.

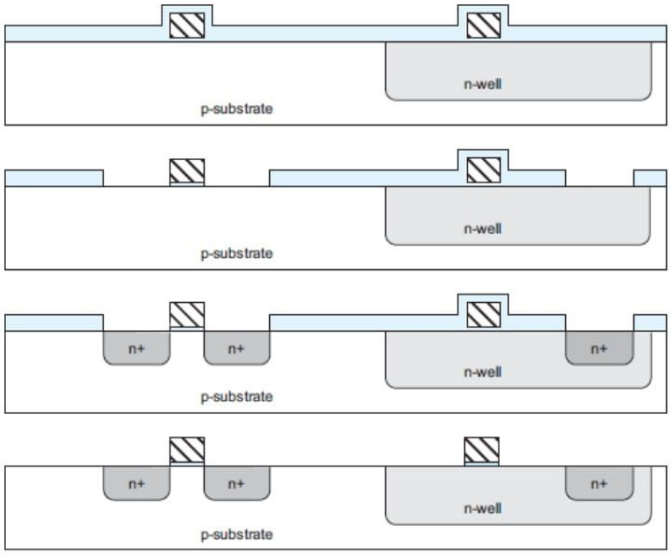


Figura 4.15: Dopaje de compuertas *Source* y *Drain* para transistor tipo-n. [19]

La situación mostrada previamente presenta el proceso para elaborar tanto un transistor tipo-n como un transistor tipo-p. Al repetir el proceso previamente descrito se finalizaría obteniendo ambos transistores acorde a lo mostrado en la Figura #4.16.

Figura 4.16: Representación de transistor tipo-n y transistor tipo-p en silicio. [19]

El dopaje tipo-p elaborado en el transistor tipo-n y viceversa para el transistor tipo-p son elaborados con la finalidad de reducir la resistencia que se genera en el *Buck*. La Figura #4.16 muestra ambos transistores, estos no presentarían ninguna funcionalidad al encontrarse cada uno individualmente, sin embargo, al ser interconectados podrían conformar una compuerta *Not*.

Para conectar ambos transistores se realiza un proceso similar de fabricación. Se comienza colocando una fina capa de SiO_2, seguida por fotorresistor. Se elimina los elementos fotorresistores y de oxido donde se colocará la conexión entre ambos transistores. Y finalmente se vierte metal que permitirá la conexión entre ambos.

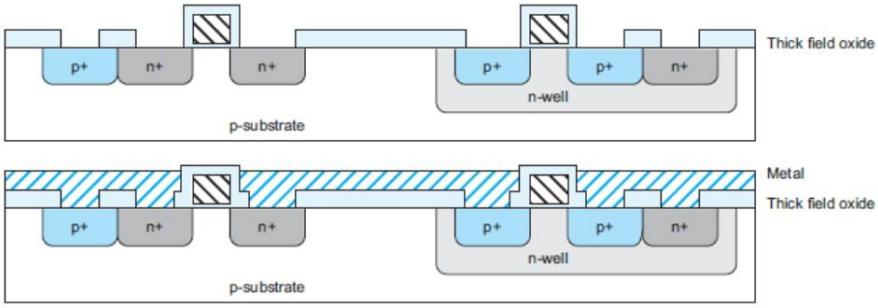


Figura 4.17: Conexión entre transistor tipo-n y transistor tipo-p para formar compuerta *Not*. [19]

Este proceso es aplicado para construir las distintas compuertas que conformarán el circuito nanométrico que se desee. Para cada circuito nanométrico deben de elaborarse mascaras especificas que permitan la elaboración de este.

4.2. Verilog

Verilog HDL se origino en el año de 1983 con la finalidad de permitir a sus usuarios elaborar circuitos de forma descriptiva. Previo a esto lenguajes como Pascal, C o Fortran fueron utilizados para suplir esta necesidad a pesar de no estar diseñados con ese objetivo. [14]

El lenguaje verilog tiene la finalidad de simular comportamientos de sistemas de forma sencilla. Presenta la ventaja de poder elaborar un sistema de forma sumamente abstracta para poder verificar su funcionalidad. Es posible optimizar los sistemas al modificarlos hasta obtener los resultados deseados. [14]

4.2.1. Jerarquías

El primer concepto a tener en cuenta al momento de elaborar diseños en verilog es la jerarquía. Existen dos tipos de jerarquías mediante las cuales es posible elaborar diseños, *top-down* y *bottom-up*. [14]

La jerarquía *top-down* consiste en establecer un bloque superior. Este bloque superior enviará datos a sub-bloques, estos sub-bloques a su vez enviará datos a sub-bloques nombrados *leaf cells*, los cuales presentarán los resultados. [14]

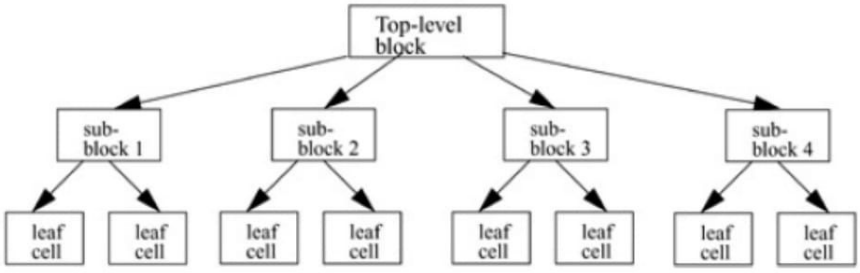


Figura 4.18: Jerarquía *top-down*. [14]

La jerarquía *bottom-up* consiste en elaborar módulos los cuales serán utilizadas para elaborar bloques de mayor magnitud que conformarán el bloque superior. Es decir, hacemos uso de múltiples bloques para conformar uno que elaborará la tarea deseada. [14]

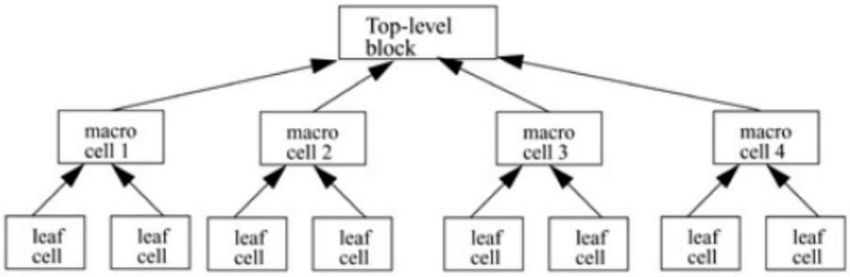


Figura 4.19: Jerarquía *bottom-up*. [14]

En general no suele utilizarse solamente una de las dos jerarquías al momento de realizar un diseño, sino una combinación de ambas. [14]

4.2.2. Módulos

Un módulo es considerado como el bloque fundamental en verilog, pues, a partir de este se formará un circuito de mayores dimensiones. Un módulo tiene la función de realizar una tarea específica, la cual se verá originada al colocar valores en sus puertos de entrada, y finalizará al presentar el resultado en sus puertos de salida, a la vez que oculta su implementación interna. Esta característica permite al diseñador modificar el módulo internamente sin afectar al sistema. [14]

Cada modulo posee un nombre que lo identifique, sus entradas y salidas deben de estar establecidas para permitir su uso. Estas son las única características que los módulos requieren para relacionarse entre sí, el funcionamiento interno del módulo dependerá totalmente del diseñador y las necesidades que se presenten. [14]

4.2.3. Implementaciones estructurales y de comportamiento

Verilog presenta la posibilidad de trabajar los módulos tanto de forma Estructural como *Behavioral* (o de comportamiento). La parte interna de un modulo puede ser elaborada a partir de cuatro niveles de abstracción distintos. El nivel de abstracción que se utilice dependerá enteramente de las necesidades. El módulo presentara un comportamiento igual no importando el nivel de abstracción utilizado, por lo que este puede ser modificado en cualquier momento. [14]

Los niveles de abstracción pueden ser:

Nivel *Behavioral* o de comportamiento : Este es el nivel de abstracción más alto, en donde basta con describir como el modulo funciona, sin tomar en cuenta la implementación en *hardware* de este. Lo siguientes tres niveles a mencionar son considerados como estructurales. [14]

Nivel de flujo de datos: En este caso el diseñador tiene conocimiento de como los datos se presentan y son procesados por el sistema en *hardware*, por lo que se basa en esta información para desarrollar los módulos del sistema. [14]

Nivel de compuertas lógicas: Este nivel se utilizan compuertas lógicas para elaborar los módulos y estos se desarrollan tal cual serian implementados en *hardware*. [14]

Nivel de interruptor: Este es el nivel más bajo de abstracción donde el sistema es elaborado a partir de interruptores, nodos de almacenamiento y la interconexión entre estos. [14]

Verilog permite el uso de distintos niveles de abstracción en un mismo diseño, a la vez esta es una práctica muy utilizada por los diseñadores. [14]

4.2.4. Estímulos del sistema

Una vez el sistema se encuentra completo o parcialmente completo y se desea comprobar su funcionalidad es posible elaborar un módulo conocido como *Stimulus* o *Testbench*. Una buena práctica es mantener el modulo *Stimulus* separado de los módulos que conforman al sistema. [14]

El bloque *Stimulus* tiene la finalidad de inicializar el diseño al generar señales de reloj o datos iniciales para el sistema. Existe la posibilidad de imprimir los resultados del sistema, así como de nodos en particular para analizar el comportamiento del circuito. [14]

4.3. Flujo de diseño

El diseño de un circuito integrado es sumamente complejo, dependiendo del objetivo que se desea que el circuito cumpla la dificultad de su diseño puede ser mayor o menor. Softwares de diseño asistido son requeridos para el diseño, y una de las dificultades se presenta al momento de decidir que herramientas utilizar y en qué orden. Dependiendo del lugar de trabajo una mayor o una menor cantidad de herramientas pueden encontrarse disponibles para el diseñador. Debido a estas características múltiples flujos de diseño han sido creados tanto por empresas como por universidades para el desarrollo de circuitos integrados. [1]

El flujo de diseño tiene la finalidad de guiar al diseñador con los pasos que este debe de seguir para obtener un diseño adecuado de su circuito. A pesar de existir múltiples formas de realizar un circuito los flujos suelen clasificarse en dos tipos: Personalizados o *ASIC*. [4]

4.3.1. *Custom* o personalizados

El diseño personalizado se basa en la elaboración de un circuito totalmente nuevo con una finalidad de comercialización. Se comienza el diseño del circuito desde cero trabajando cada segmento que lo conforma. Este tipo de circuitos son elaborados principalmente por compañías dedicadas al diseño y fabricación de circuitos integrados, ya que requiere de una basta cantidad de tiempo y personas trabajando a la vez. [4]

Al hablar de flujo de diseño personalizado este dependerá del lugar donde se desarrolle el circuito. Elaborando un ejemplo, Intel posee un flujo de diseño propio, elaborado para suplir sus necesidades. Sin embargo, este flujo seria distinto al realizado por AMD, o otra compañía. No existe verdaderamente un flujo de diseño a seguir en este caso, pues depende del sitio en que se desarrolle el circuito, las herramientas que se utilicen en el lugar, etc. [4]

4.3.2. *ASIC*

Un diseño *ASIC* hace referencia a *Application Specific Integrated Circuit*, el cual se refiere a un circuito desarrollado para cumplir con una tarea específica. En general, los circuitos de usos múltiples como microcontroladores o *FPGA* serán desarrollados mediante un proceso personalizado. [4]

Para el caso de circuitos que busquen cumplir con objetivos específicos se realizara un flujo de diseño *ASIC*. Al igual que con los diseños personalizados el flujo de diseño para un circuito *ASIC* dependerá del lugar donde se realice, sin embargo, al ser este tipo de circuitos mayormente diseñados en universidades y demás entidades de enseñanza se ha creado un flujo de diseño general el cual es mayormente seguido, variando generalmente en las herramientas que se utilizan. El flujo de diseño típico para un circuito *ASIC* puede observarse en la Figura #4.20. [4]

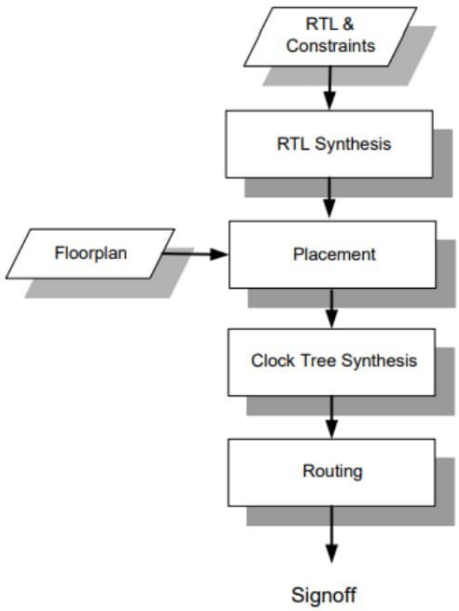


Figura 4.20: Flujo de diseño *ASIC* típico. [2]

El flujo comienza con la elaboración del *RTL* y *constraints*. *Register-transfer level* (*RTL*) se refiere a una elaboración abstracta del circuito que se desea elaborar, generalmente esto suele realizarse en lenguaje descriptivo verilog. *Constraints* hace referencia al conjunto de especificaciones del circuito, conocidas también como restricciones del diseño. [2]

Una vez elaborado el circuito que se desea fabricar por medio de un archivo verilog este es convertido por medio de un software a un *netlist* de compuertas lógicas. [2]

El *netlist* de compuertas lógicas es cargado en una herramienta capaz de realizar una conversión de este a una representaciones en silicio de los componentes que conforman el circuito. Este paso es conocido como *Floorplan/Placement*. Posteriormente se realizan verificaciones de funcionamiento y finalmente las conexiones entre los componentes para obtener una representación del circuito en silicio. [2]

4.4. *FPGA*

Field Programmable Gate Arrays (*FPGA*) son el resultado de avances en la electrónica y desarrollo en tecnológica nanométrica. Estos elementos electrónicos pueden ser definidos como dispositivos lógicos programables. [12]

Típicamente estos elementos se encuentran conformados por bloques de propósito general con interconexiones programables entre ellos. Esto permite que el *FPGA* no sea un circuito que cumpla

con una tarea en específico, sino un circuito elaborado para cumplir la tarea que el usuario requiera. [12]

Mientras que los microcontroladores pueden cumplir tareas específicas requeridas por el usuario utilizando sus recursos, los *FPGA* poseen un conjunto de miles de millones de transistores los cuales se interconectan para formar un circuito único que cumplirá con la tarea. Esto genera la posibilidad de poder elaborar circuitos nanométricos en unos pocos minutos, ya que, al colocar un diseño en un *FPGA* este elaborará internamente el diseño utilizando los circuitos que lo conforman y presentará el comportamiento de este diseño por medio de elementos electrónicos como *leds* o mediante software.

El *FPGA* presenta la ventaja de poder ser programado utilizando el lenguaje descriptivo verilog. Al elaborar un sistema electrónico de forma estructural utilizando compuertas lógicas y cargar este sistema al *FPGA* una réplica exacta del sistema se elaborará de forma física en el interior de este. [12]

Utilizando lenguaje verilog es posible verificar el comportamiento de un sistema por medio de software, sin embargo, al combinarlo con un *FPGA* es posible llevar la verificación del circuito a un sistema físico. Una característica importante es que los *FPGA* permiten generar circuitos a partir de cualquier nivel de abstracción utilizado en verilog.

4.5. Diseño asistido por computadora

Uno de los motivos por el cual es posible elaborar circuitos nanométricos en la actualidad se debe al desarrollo de software de diseño asistido por computador o *computer-aided design* (*CAD*). Con el desarrollo de nuevas tecnológicas se presento la oportunidad del desarrollo de software especializado capaz de facilitar y permitir el diseño de elementos de uso diario. Estos softwares permiten a las empresas cambiar sus prácticas de producción habituales, reemplazando un trabajo intensivo por métodos computarizados. [8]

Resultaría altamente complejo elaborar el diseño de circuitos a nano escala de forma manual, al tratarse de elementos sumamente pequeños se requiere de software sumamente especializado que permita el diseño y verificación de estos.

Estos software han sido desarrollados en las últimas décadas, siendo los dos principales software de diseño asistido por computador Synopsys y Cadence.

4.6. Synopsys

Synopsys se ha posicionado como la compañía desarrolladora de las principales herramientas para el desarrollo de circuitos nanométricos. Empresas tales como Intel, AMD o Xilinx utilizan su software para el desarrollo de nuevas tecnologías. [7]

La empresa cuenta con cientos de herramientas, cada una desarrollada para trabajar un segmento especifico del circuito integrado que se desea diseñar. [7]

4.6.1. *Design Compiler*

Design Compiler es un paquete de 17 herramientas desarrollado por Synopsys dentro del cual se encuentran herramientas como "*DC Expert*", "*DC Ultra*" o "*Design Vision*", herramientas elaboradas específicamente para realizar síntesis. [6]

La síntesis consiste en realizar una conversión de circuitos elaborados en lenguaje descriptivos verilog a un *netlist* de compuertas lógicas. Para la elaboración de la síntesis es necesario utilizar librerías con especificaciones de los componentes que conformarán el circuito, así como establecer los *constraints* o restricciones del diseño. [7]

En el presente trabajo se hará uso de la herramienta *Design Vision*, herramienta desarrollada por Synopsys en años recientes la cual presenta características de alto impacto para el desarrollo de síntesis, como la posibilidad de utilizar tanto una interfaz gráfica como comandos en consola, visualización de esquemáticos y gráficas.

4.6.2. *Constraints*

Las restricciones de diseño pueden ser considerados como las reglas o especificaciones de diseño que se desean para el circuito. Estos se establecerán durante la síntesis en la herramienta *Design Vision*, y contendrán información como: la frecuencia de reloj a la cual se desea que el circuito opere, el *delay* de entrada y salida del circuito, el área máxima del circuito, entre otros. [9]

Estos datos serán interpretados por *Design Vision*. La herramienta intentara optimizar el circuito hasta obtener las especificaciones establecidas, en caso de no poder obtener los resultados deseados elaborará la síntesis de tal forma que el circuito presente los resultados más cercanos a las restricciones establecidas, e informará que no fue posible generar los resultados esperados. [9]

En caso de que se requiera que las restricciones se cumplan y la herramienta no pueda optimizar el circuito lo suficiente para obtener estos resultados se deberá utilizar una arquitectura distinta la cual permita obtener el comportamiento deseado. [9]

4.6.3. *Netlist* de compuertas lógicas

Un *netlist* de compuertas lógicas es un archivo el cual contiene el circuito que se desea fabricar elaborado de forma estructural utilizando compuertas lógicas. Todas las compuertas lógicas que conforman este circuito se encuentran referenciadas en librerías provistas por Synopsys o por el fabricante. [6]

4.6.4. Librerías

En el caso de herramientas de síntesis como *Design Vision*, las librerías se limitaran a archivos .db o .sdb en donde se encontrarán almacenadas las características de una diversidad de componentes electrónicos. Estas poseen información sobre compuertas lógicas, multiplexores, amplificadores, capacitores y demás componentes electrónicos que conformaran el circuito en cuestión. Especificaciones sobre el tamaño de transistores, conexiones, potencia consumida y demás también se encuentran dentro de estas librerías. [6]

La importancia de estos archivos radica en que, al momento de realizar la síntesis el software que se utilice tomara el circuito en verilog y generará un archivo equivalente con el mismo circuito, con la diferencia de que en este nuevo archivo el circuito se encontrará elaborado utilizando las compuertas y componentes que las librerías posean. [6]

Esto es de suma importancia pues las librerías serán el puente entre la elaboración de un circuito descriptivo en verilog y la elaboración de este a nivel de transistores.

4.6.5. *VCS*

VCS (*Verilog Compiler Simulator*) es una herramienta de Synopsys desarrollada para simular sistemas elaborados en lenguaje descriptivo de *hardware*. Esta herramienta permite, por medio de una interfaz gráfica, observar el comportamiento de un sistema para determinar si este funciona correctamente. [16]

La herramienta presenta una gran importancia en el flujo de diseño utilizado en el presente proyecto, pues permite simular tanto el circuito previo a la elaboración de síntesis como después de esta. Permitiendo comprobar que el circuito funcione de forma adecuada en ambos casos. [16]

4.6.6. *Formality*

Posterior a la elaboración de síntesis de un circuito en lenguaje descriptivo verilog a un *netlist* de compuertas lógicas se debe de verificar que la síntesis se haya realizado de forma correcta, que el *netlist* obtenido presente un funcionamiento correcto y no exista una desigualdad en cuanto al comportamiento de ambos diseños. [17]

Para esto, la herramienta *Formality* de Synopsys presenta la posibilidad de comprobar la equivalencia entre dos diseños. Su principal uso se presenta en la comparación entre un *netlist* de compuertas lógicas y su equivalente en lenguaje descriptivo verilog. Tras la comparación, la herramienta reporta si los diseños son funcionalmente equivalentes o no. [17]

CAPÍTULO 5

Antecedentes

El curso Introducción a sistemas de diseño *VLSI* fue impartido por primera vez en el año 2013 por el Ing. Carlos Esquit, siendo este el primer acercamiento en el estudio de circuitos nanométricos por parte de la Universidad del Valle de Guatemala. Al año siguiente fue creado un acuerdo académico con la empresa Synopsys para poder hacer uso de sus herramientas, permitiendo a los estudiantes comprender de mejor forma la electrónica a nanoescala. Proyectos previos en el campo de la nanoelectrónica elaborados por estudiantes de la Universidad del Valle de Guatemala se presentan en las referencias [15] y [3]

Estos proyectos en conjunto con los cursos impartidos han creado las bases para la elaboración del presente proyecto.

A nivel internacional diversas instituciones educativas han realizado diseños de circuitos integrados. El trabajo presentado en [9] presenta similitud y se utilizo como referencia para el presente trabajo.

CAPÍTULO 6

Alcance

En esta investigación se pretende elaborar la conversión de distintos circuitos elaborados en lenguaje descriptivo de hardware a *netlists* de compuertas lógicas como primer paso en el desarrollo de un flujo de diseño para circuitos nanométricos.

Para el desarrollo de esta propuesta, se identificarán e instalarán las herramientas necesarias a utilizar para elaborar la síntesis de los circuitos elaborados en lenguaje descriptivo de *hardware*, se verificará el funcionamiento de los circuitos antes y después de la síntesis, y se documentará el proceso para que futuros estudiantes de la Universidad del Valle de Guatemala puedan hacer uso del presente trabajo como referencia en futuros proyectos.

Se pretende utilizar librerías estudiantiles para comprender el proceso mediante el cual la síntesis se realiza, así como utilizar librerías proporcionadas por fabricantes para la elaboración de un circuito en particular el cual se pretende sea fabricado.

CAPÍTULO 7

Preparación de software

7.1. Investigación inicial

Al no existir un proyecto similar realizado en la universidad o en el país se comenzó realizando una investigación para determinar los pasos a seguir para elaborar el diseño de un circuito nanométrico.

La Universidad del Valle de Guatemala posee acceso al software Synopsys y a una basta cantidad de herramientas desarrolladas por esta empresa, así como la posibilidad de instalar nuevas herramientas que pudieran resultar necesarias. Al conocer que este sería el software que se utilizaría para el desarrollo del diseño se decidió comenzar la investigación en la página de la empresa, obteniendo poca información y decidiendo redirigir la investigación.

Un primer acercamiento a la elaboración del diseño se dio al obtener información sobre la elaboración de circuitos *ASIC*. Una investigación más profunda permitió obtener un conjunto bibliográfico el cual sería utilizado a lo largo de la elaboración del diseño.

Se realizó un estudio sobre el flujo de diseño mediante el cual los circuitos *ASIC* se desarrollaban. A pesar de existir variaciones en cuanto a flujos de diseño para circuitos *ASIC* en su mayoría todos presentaban los mismos pasos variando únicamente en las herramientas que se utilizaban.

Según las referencias antes mencionadas se determinó la herramienta a utilizar para elaborar la síntesis del diseño. Gran parte de las referencias presentaban la herramienta *DC Ultra* de Synopsys como la indicada para realizar síntesis.

Al indagar en la página oficial de Synopsys se obtuvo más información de la herramienta, siendo esta parte de un paquete nombrado *Design Compiler*, el cual posee un conjunto de distintas herramientas utilizadas para el diseño de circuitos a nano escala. La gran mayoría de las herramientas de este paquete se encuentran diseñadas para elaborar síntesis de circuitos elaborados en lenguaje descriptivo verilog.

De igual forma, se determinó que las herramientas *VCS* y *Formality* permitirían realizar la verificación de funcionamiento del circuito tanto antes como después de la síntesis, permitiendo determinar que esta se realizara correctamente.

Al establecer las herramientas necesarias para realizar la síntesis y verificación se prosiguió elaborando la instalación de estas.

7.2. Instalación de herramientas

A pesar de contar con acceso al software de Synopsys la Universidad del Valle de Guatemala no cuenta con un departamento encargado de la instalación de este tipo de herramientas o un soporte técnico de Synopsys que provea ayuda en el tema. Debido a esto, se realizó una investigación para conocer el proceso de instalación para las herramientas.

Synopsys cuenta con una basta documentación para la instalación de herramientas, sin embargo, este proceso puede resultar complejo o presentar una diversidad de errores. El servicio de Solvnet, brindado por Synopsys, es una página web oficial de la empresa donde se encuentran las herramientas para su instalación, así como su documentación, guías de instalación e información para resolución de errores.

Se comenzó descargando las guías para la instalación de *Design Compiler*, *VCS* y *Formality*. Estas guías fueron utilizadas como referencia en todo momento durante la instalación de las herramientas. Para obtener una instalación adecuada es necesario editar ciertas variables del sistema mediante comandos establecidos en las guías de instalación, establecer directorios y rutas, y demás configuraciones.

Posterior a la instalación se obtuvo acceso a las herramientas *VCS*, *Formality* y al paquete *Design Compiler*. Dentro de este paquete se presentaba la herramienta *Design Vision*. Al realizar una investigación sobre este software se determino que presentaba múltiples ventajas por sobre *DC Ultra*. A pesar de que ambos se encuentran desarrollados para elaborar la misma tarea, *Design Vision* presenta una interfaz gráfica que facilita su uso, la posibilidad de observar el circuito sintetizado por medio de esquemáticos y la generación de múltiples reportes del circuito, entre otras características. Debido a esto se decidió utilizar esta herramienta para elaborar la síntesis del circuito.

Debido a que este trabajo no se enfoca en la instalación de las herramientas no se ahondará más en este proceso. Sin embargo, una serie de vídeos explicativos han sido elaborados como parte del proyecto, en caso de requerir más información de este tema se recomienda utilizarlos como referencia. De igual forma, se recomienda utilizar el trabajo presentando en [15], pues este desarrolla de forma completa el proceso de instalación de herramientas de Synopsys.

7.3. Obtención de librerías

Posterior a la instalación de la herramienta se comenzó a investigar sobre la obtención de librerías para elaborar síntesis de circuitos. La obtención de librerías presentó dificultad al no existir un fácil acceso a ellas, sin embargo, fue posible realizar su descarga.

Las librerías utilizadas poseen diversidad de componentes electrónicos de 90nm y se encuentran desarrolladas por Synopsys. Estas librerías fueron utilizadas con el único propósito de realizar pruebas de síntesis y comprender el funcionamiento de *Design Vision*.

Las librerías proporcionadas por Synopsys poseen acuerdos legales los cuales establecen que su uso se encuentra restringido a motivos académicos de aprendizaje, estas librerías no pueden ser utilizadas para la elaboración de circuitos que tengan fines comerciales o propósitos de fabricación. Por este motivo, las librerías permiten elaborar el flujo de diseño hasta obtener el circuito en su representación de silicio y realizar un *place and route*, sin embargo, no permiten continuar más allá de este paso pues no tienen definidas algunas de sus características. Esto permite que no se utilicen

con fines de fabricación. Para elaborar un flujo de diseño con la finalidad de fabricar se deben utilizar librerías brindadas por el fabricante.

Las librerías pueden ser obtenidas en el enlace presentado en [18]. Al día en que este trabajo es presentado el enlace presenta librerías en 28nm, 32nm y 90nm. Para la elaboración del flujo de diseño presentado en este trabajo deben de utilizarse las librerías de 90nm, pues solamente estas poseen los archivos requeridos para la elaboración de síntesis o *place and route*. Para su uso basta con descargar los archivos comprimidos que se presentan en el enlace, y descomprimir estos en una carpeta del ordenador.

La Figura #7.1 muestra las carpetas que contienen las distintas librerías proporcionadas por Synopsys, estas se colocaron en la dirección: usr>Synopsys>PDK

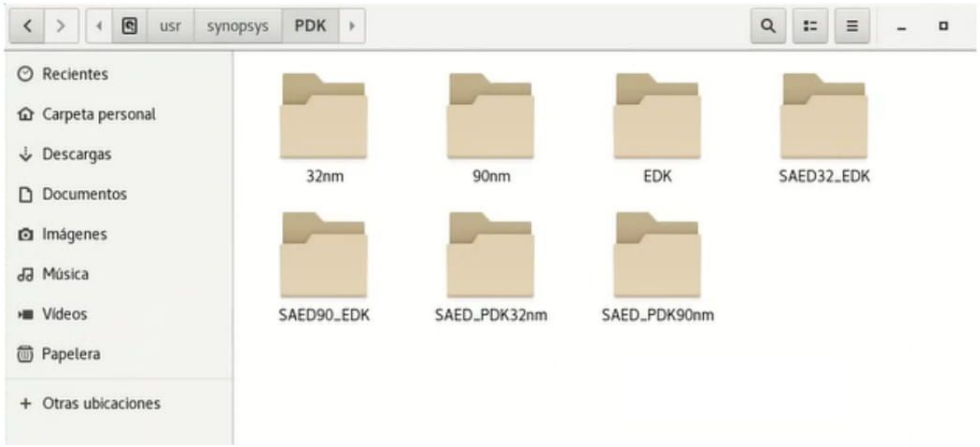


Figura 7.1: Librerías de Synopsys.

CAPÍTULO 8

Estudio de la herramienta *Design Vision*

8.1. Uso de librerías

Una vez instalada la herramienta *Design Vision* y descargadas las librerías de Synopsys se prosiguió estudiando el funcionamiento de estas. Al descomprimir los archivos de las librerías y colocarlos en los directorios indicados se exploraron los archivos que estas contenían.

Las librerías son un conjunto de archivos contenidos en una basta cantidad de carpetas. Estos archivos poseen las características de cientos de elementos electrónicos, dictando su comportamiento y las características de los transistores que los conforman.

Al presentarse una cantidad tan grande de archivos se prosiguió a buscar referencias bibliográficas que permitieran un mejor entendimiento. A partir de esta investigación se determinó que, *Design Vision* requiere de tres librerías para su funcionamiento: *Link library*, *Target library* y *Symbol library*. Mientras que *Formality* solamente requiere *Target library* y *VCS* la librería de la tecnológica en verilog.

Link library es una librería utilizada para enlazar elementos referenciados en el circuito elaborado en verilog. Es decir, al momento de elaborar un circuito en verilog es posible instanciar celdas que se encuentren contenidas en librerías .db de Synopsys. Al momento de realizar la síntesis esta librería se encargará de determinar si existe alguna celda de su propiedad referencia y, en caso de ser así, la reemplazará en el archivo sintetizado colocando todas las características requeridas para su elaboración en silicio. En caso de no existir ningún elemento referenciado en el circuito de verilog es posible colocar en esta librería la misma librería utilizada para la *Target library*.

Target library es utilizada como una librería que contendrá todas las compuertas lógicas y demás componentes que conformarán al circuito sintetizado. Es decir, el circuito sintetizado se encontrará elaborado a partir de los componentes almacenados en la *Target library*.

Finalmente, la *Symbol library* posee la representación en símbolos de los componentes utilizados para elaborar el circuito sintetizado. Esta librería permite desplegar una representación del circuito que se está trabajando.

Tanto la *Link library* como la *Target library* se encuentran como archivos .db, mientras que la *Symbol library* se presenta como un archivo .sdb.

Existen otros tipos de librerías las cuales pueden ser encontradas dentro de las carpetas, como lo son archivos .tluplus, .tf, .map, entre otros. Estos archivos son de importancia pues se utilizarán en el siguiente paso del flujo de diseño, para realizar el *place and route* del circuito. El uso de estos archivos se encuentra en el trabajo que da seguimiento a este.

Para comprender como las librerías se encuentran organizadas es posible observar que estas se encuentran divididas en *EDK* (*Educational Design Kit*) y *PDK* (*Process Design Kit*), como se muestra en la Figura#7.1. Ambas librerías han sido elaboradas por Synopsys con la finalidad de proveer herramientas educativas por medio de las cuales estudiantes puedan comprender las técnicas de diseño.

La diferencia entre ambas librerías se basa en que el paquete *EDK* se enfoca en proveer archivos para el desarrollo de circuitos integrados desde su fase de síntesis hasta su *place and route*, mientras que el paquete *PDK* se enfoca en proveer archivos a nivel de silicio para estudiar las reglas que rigen estos y como se verifica su funcionamiento a este nivel.

Debido a esto nos centraremos en las librerías *EDK*, al ingresar en estas es posible observar que estas se dividen en seis carpetas distintas, como se presenta en la Figura#8.1. Estas seis carpetas se dividen en subcarpetas y estas a su vez en subcarpetas.

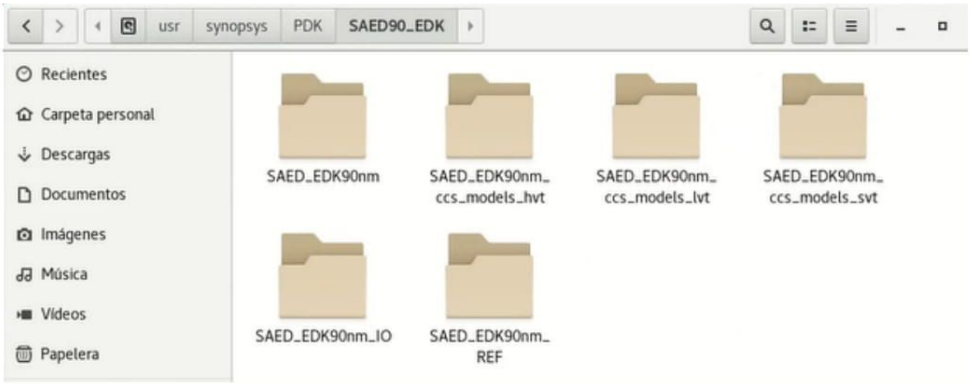


Figura 8.1: Organización de librerías de 90nm.

Debido a la gran cantidad de archivos que se encuentran almacenados, y debido a que el tema central de este trabajo no es la explicación de estos archivos nos centraremos en la carpeta SAED_EDK90nm_REF.

8.2. Ejemplos en librerías

Dentro de la carpeta SAED_EDK90nm_REF es posible encontrar cinco ejemplos: ChipTop, Leon3, OpenSPARC, PARSER y PowerPC405. Estos ejemplos poseen todos los archivos necesarios para la elaboración de síntesis y los próximos pasos en el flujo de diseño, dentro de cada carpeta es posible encontrar circuitos elaborados en verilog, *scripts*, las librerías a utilizar y archivos *ReadMe.txt* con información.

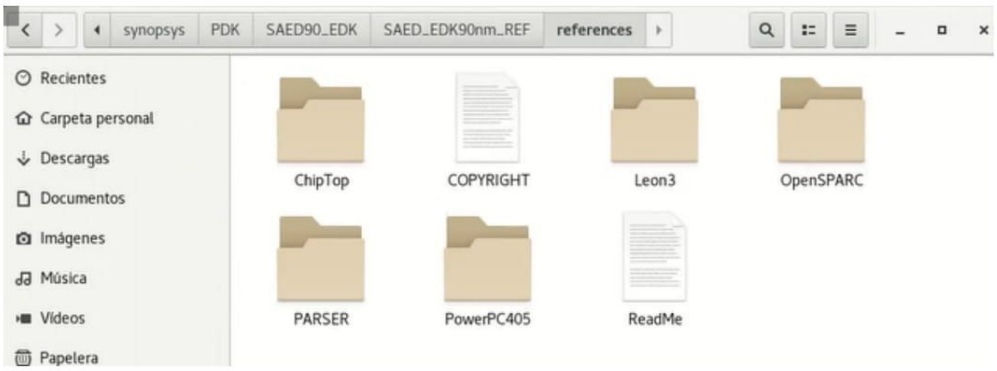


Figura 8.2: Ejemplos de síntesis utilizando librerías de 90nm.

Los ejemplos presentados son altamente intuitivos y fueron utilizados (junto a otras referencias) como una guía para la elaboración de la síntesis y posteriores pasos en el flujo de diseño. Principalmente el ejemplo PARSER.

8.3. Restricciones y listado de comandos relevantes

Los *constraints* o restricciones de diseño especificaran los objetivos que se esperan del diseño. Estos definirán información como el área máxima, frecuencia de operación o *delay* del circuito. A continuación se presentan el listado de comandos mediante los cuales las restricciones se definen y su función.

La Tabla #8.1 presenta las restricciones que deben de utilizarse para definir el área que ocupara el circuito con el cual se esta trabajando.

Comandos para definir restricciones	Función
set_max_area	Esta restricción especifica el área máxima que el diseño debería de tener. El valor es dimensionado en las unidades usadas para las celdas en la librería utilizada. Al especificar el área máxima como 0 resultara en la herramienta optimizando el diseño al máximo para obtener la menor área posible. Ej. design_vision>set_max_area 0

Tabla 8.1: Restricciones para establecer área del diseño.

La Tabla #8.2 presenta las restricciones que deben de aplicarse para establecer el reloj del sistema. En caso de utilizar varios relojes los comandos deben de aplicarse para cada uno de estos modificando el nombre del reloj.

Es importante resaltar que los valores que se utilicen deben de ser realistas, debido que el utilizar valores poco realistas para el reloj o *delay* del sistema puede resultar en un aumento significativo en el área del diseño.

Comandos para definir restricciones	Función
create_clock	Este comando se utiliza para definir el reloj al cual funcionará el sistema. El comando posee ciertos parámetros que permiten definir el comportamiento del reloj. Estos son: -name. Permite nombrar el reloj que se está creando. -period. Define el periodo al colocarle un valor -waveform. Controla el ciclo de trabajo. El reloj se coloca en el pin o puerto del diseño por medio del comando [get_port], al cual se le especifica el nombre del pin o puerto dentro de los corchetes. Ej. design_vision>create_clock –period 40 -name clock_1 –waveform {0 20} [get_ports clk] El ejemplo anterior crea un reloj con el nombre "clock_1", el cual tiene un periodo de 40 ns, con 50 % de ciclo de trabajo (ya que se define que este comienza en un valor alto en 0 ns y cambia a un valor bajo en 20 ns). También se establece que el reloj se colocará en el puerto "clk" del diseño.
set_clock_uncertainty	Define la incertidumbre para el reloj. Basta con colocar un valor junto a este comando. Ej. design_vision>set_clock_uncertainty 1 [get_clocks clock _1] En el ejemplo anterior se establece una incertidumbre de 1 ns para el clock nombrado "clock _1".

Tabla 8.2: Restricciones para establecer reloj del diseño.

La Tabla #8.3 presenta las restricciones para definir los *delays* a los que operará el sistema. En caso de no definir los *delays* o reloj del sistema la síntesis puede realizarse correctamente, sin embargo, el diseño no será optimizado. El uso de estas restricciones permite que el diseño opere a una mayor velocidad y que la herramienta reordene y reduzca los componentes para obtener un diseño optimizado.

Las restricciones para *delay* del circuito permiten establecer el tiempo necesario para que las señales de entrada y salida se estabilicen, así como definir el tiempo necesario para que el circuito realice la tarea que se desea.

Comandos para definir restricciones	Función
set_input_delay	Especifica el tiempo que le tomara a los datos de entrada estabilizarse luego del pulso de reloj. Al igual que el comando anterior, existen parámetros para definir el comportamiento. En este caso, existen las opciones -max y -min, para establecer el valor máximo y mínimo para el delay de entrada, y la opción -clock, para elegir el reloj con el cual se trabaja. Ej. design_vision>set_input_delay –max 23.0 –clock clock_1 \ [remove_from_collection [all_inputs] [get_ports clk]] design_vision>set_input_delay –min 20.0 –clock clock_1 \ [remove_from_collection [all_inputs] [get_ports clk]] El ejemplo anterior establece los valores de delay máximo y mínimo de entrada en nanosegundos, así como el uso del reloj nombrado “clock_1”. El comando “[remove_from_collection [all_inputs][get_ports clk]]” establece la restricción en todos los puertos de entrada
set_output_delay	Especifica el tiempo que le tomara a los datos de salida estabilizarse luego del pulso de reloj. Su uso es igual al del comando set_input_delay. Ej. design_vision>set_output_delay – max 32.0 –clock clock_1 [all_outputs]
set_max_delay	Esta restricción permite establecer el delay máximo que tendrá un segmento especifico del circuito. Ej. design_vision>set_max_delay 5 –from all_inputs() – to_all_outputs()
set_min_delay	Esta restricción permite establecer el delay mínimo que tendrá un segmento especifico del circuito. Ej. design_vision>set_min_delay 3 –from all_inputs() – to_all_outputs()

Tabla 8.3: Restricciones para establecer *delays* del diseño.

Las restricciones anteriormente presentados son las más utilizadas en diseños básicos pues son las que tienen un mayor impacto en la optimización del diseño. Sin embargo, la Tabla #8.4 presenta restricciones utilizadas para características un poco más precisas.

Acá se ha presentado las restricciones básicas y mayormente utilizadas, sin embargo, existen otros comandos los cuales se utilizan en caso de elaborar circuitos que requieran un mayor grado de especificaciones. Un listado con las restricciones omitidas, su función y ejemplos se presentan en [11] como recurso en caso de requerir un mayor grado de especificaciones en el diseño a realizar. De igual forma, el manuales de usuario de *Design Vision* posee un apartado del tema.

Comandos para definir restricciones	Función
reset_design	Remueve las restricciones del diseño Ej. design_vision>reset_design
set_operating_conditions	Existen tres elementos que afectan como la herramienta optimizará el circuito: el proceso de fabricación, el voltaje y la temperatura. Estos factores se encuentran establecidos en las librerías que se utilicen, sin embargo, es posible establecer las condiciones en las que se desea el circuito sea optimizado. Las condiciones del circuito pueden ser: peor caso, típicas o mejor caso. Dependiendo de las condiciones que se elijan la herramienta considerará un proceso de fabricación malo, bueno o excelente, temperaturas o voltajes malos, buenos o excelentes, y en base a esto realizará la optimización del circuito. Se utilizarán los parámetros WCCOM para el peor caso, TYPICAL para el caso típico y BCCOM para el mejor caso. Ej. design_vision>set_operating_conditions WCCOM design_vision>set_operating_conditions TYPICAL design_vision>set_operating_conditions BCCOM
set_drive	Establece la impedancia que se tendrá en los puertos de entrada. De forma predefinida se posee una impedancia con valor 0, se recomienda continuar con este valor pues impone una impedancia infinita de entrada. Ej. design_vision>set_drive 0 [all_inputs]
set_load	Establece la capacitancia en los puertos de salida en pf, el valor que se coloque dependerá de la tecnología con la que se esté trabajando. Ej. design_vision>set_load 5 [all_outputs]
set_max_transition	Establece el delay máximo que tendrá todo el circuito en ns. Ej. design_vision>set_max_transition 1.5 {nombre_diseño}
set_max_capacitance	Establece la capacitancia máxima que tendrá todo el circuito en pf. Ej. design_vision>set_max_capacitance 1.5 {nombre_diseño}

Tabla 8.4: Restricciones de diseño específicos

8.4. Prueba básica: *Full Adder*

Para realizar una primer prueba de funcionamiento de la herramienta *Design Vision* se elaboro el circuito de un *Full Adder* en verilog de forma estructural utilizando compuertas lógicas. Se decidió utilizar este circuito debido a su simplicidad, el circuito elaborado se presenta en la Figura #8.3.

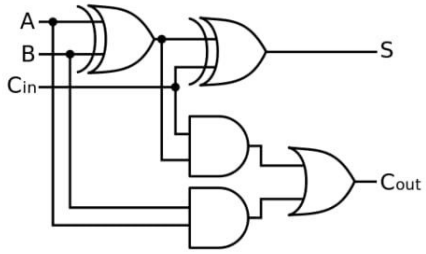


Figura 8.3: Esquemático de circuito *Full Adder*.

Una vez elaborado el circuito se prosiguió a abrir la herramienta *Design Vision* y cargar las librerías a utilizar. Para ejecutar la herramienta basta con escribir el comando '*design_ vision*' en la terminal del sistema. Se utilizaron las librerías saed90nm_max.db, saed90nm_min.db y saed90nm_typ.db para la *link library*. La libreria saed90nm_typ.db para la *target library*, y la librería saed90nm.sdb para la *Symbol library*.

En este ejemplo las librerías colocadas en *link library* no realizaran ninguna tarea pues el circuito elaborado no posee ninguna instancia de algún componente que se encuentre en estas librerías.

Para colocar las librerías se utilizo la interfaz gráfica de la herramienta. Para colocar las librerías utilizando la interfaz basta con presionar la pestaña *File->setup* y luego se desplegara la ventana mostrada en la Figura #8.4. A partir de este punto solamente se deberá de colocar la ubicación donde se encuentren las librerías, en este caso: /usr/synopsys/PDK/SAED90_EDK/SAED_EDK90nm_REF /references/PARSER/ref/models/

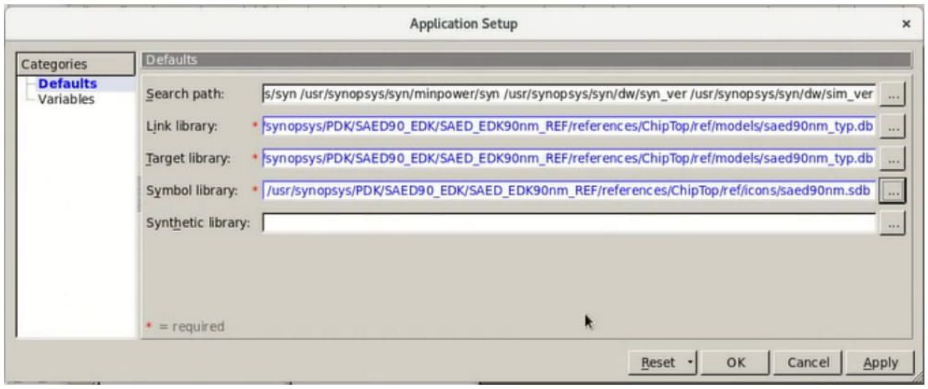


Figura 8.4: Librerías utilizadas para prueba de circuito *Full Adder*.

Una vez establecidas las librerías se debe de cargar el circuito elaborado en verilog. Para cargar el circuito se debe utilizar la opción *Analize* y posteriormente la opción *Elaborate*. La opción *Analyze* permite cargar el circuito, y se encarga de elaborar una revisión de errores sintácticos en el archivo. Posteriormente la opción *Elaborate* se encarga de los elaborar los módulos que conforman el circuito interpretándolos como hardware. La Figura #8.5 muestra ambas pestañas.

Design Vision presenta la opción *Read* en su interfaz gráfica, esta opción también puede ser utilizada para leer el circuito elaborado en verilog. Al utilizar esta opción la herramienta elabora los comandos *Analyze* y *Elaborate* de forma automática.

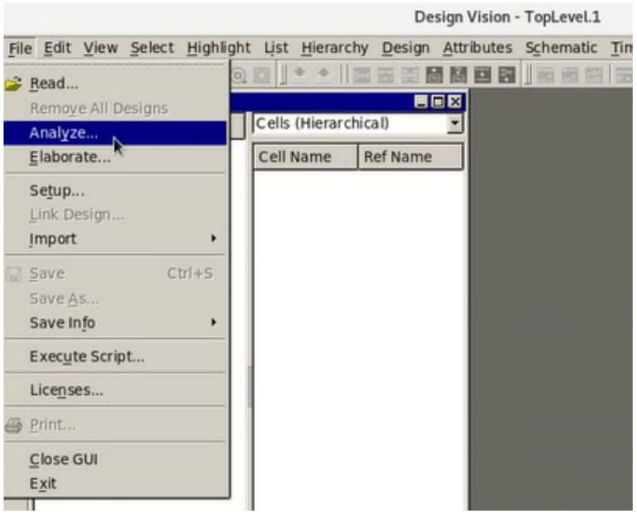


Figura 8.5: Pestañas de *Analyze* y *Elaborate*.

Posterior a seleccionar la opción *Elaborate*, *Design visión* presenta una o varias casillas con los módulos que conforman el circuito.

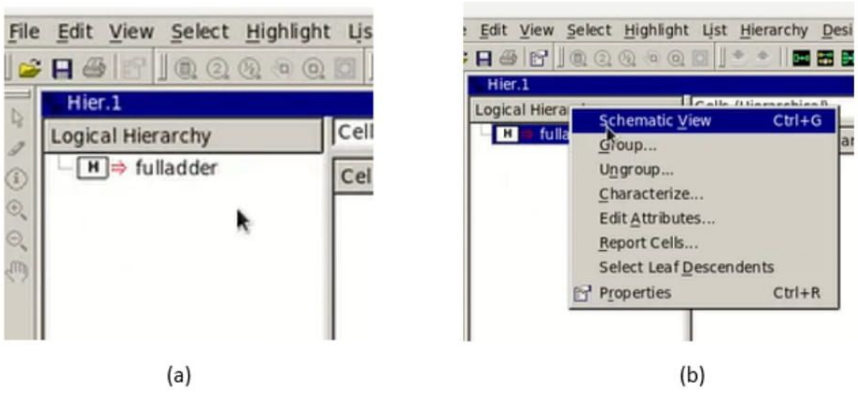


Figura 8.6: (a) Representación de módulos en *Design Vision*. (b) Opción para mostrar circuito esquemático.

Al hacer click sobre los módulos se genera una ventanilla que despliega distintas opciones. Al presionar la opción *Schematic view* se presenta una representación en forma de caja negra del circuito que estamos trabajando, en caso de darle doble click este presentará el esquemático del circuito.

Como es posible observar en la Figura #8.7, el circuito presentado en el esquemático es el mismo circuito elaborado en verilog, sin embargo, este se encuentra elaborado a partir de compuertas establecidas en la *target library*.

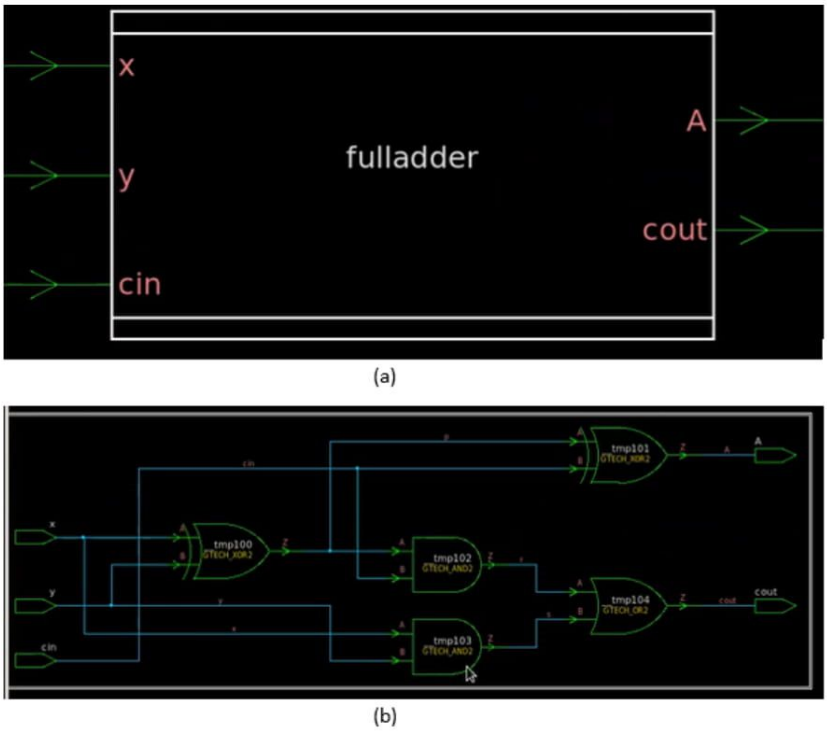


Figura 8.7: (a) Representación de caja negra del circuito. (b) Vista esquemática del circuito *Full Adder* en *Design Vision*.

Una vez cargado el circuito se procede colocando las restricciones del diseño. En esta primera prueba no se colocarán restricciones debido a que nos encontramos trabajando con un circuito sumamente pequeño y simple.

Posteriormente se continuó ejecutando las opciones *Check Design* y *Compile Design*. La opción *Check Design* realiza una verificación del circuito elaborado en lenguaje descriptivo, verifica las conexiones que este posee, las relaciones entre módulos, las salidas y entradas de estos, y determina si existe algún riesgo o error con el diseño realizado.

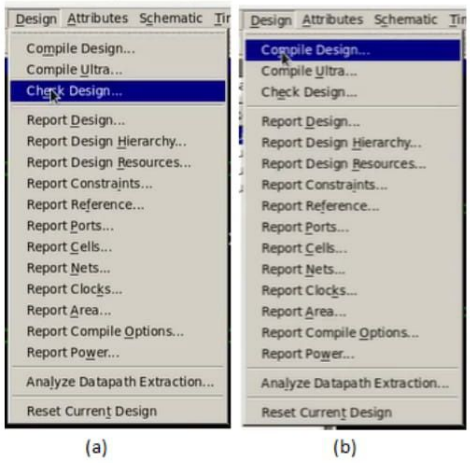


Figura 8.8: (a) Opción *Check Design*. (b) Opción *Compile Design*.

La opción *Compile Design* abre una nueva ventada donde se presenta la posibilidad de seleccionar que tanto se desea optimizar el diseño, en todos los casos generaremos el *netlist* sin modificar ninguna característica en esta ventana ya que no requerimos de una alta optimización. Posteriormente la opción genera el *netlist* de compuertas lógicas.

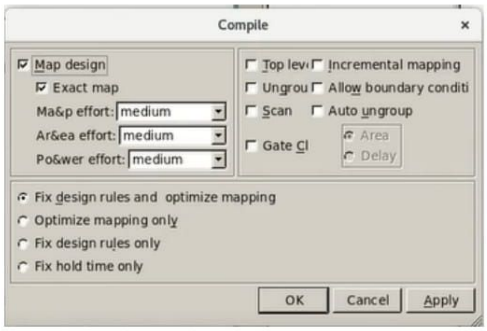


Figura 8.9: Ventana *Compile Design*.

Previo a guardar el *netlist* se presenta la posibilidad de generar reportes con información relevante del diseño elaborado. Estos reportes se presentan en la pestaña *Design* y presentan información del diseño sintetizado como el área, potencia, celdas, puertos, entre otros.

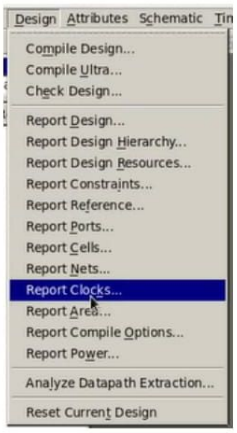


Figura 8.10: Generación de reportes con *Design Vision*.

Finalmente, el archivo se guarda, como en cualquier otro software, por medio de la opción *Save as*, utilizando un nombre elegido por el diseñador y un formato verilog. El archivo guardado es el *netlist* de compuertas lógicas, al abrir este es posible observar el mismo circuito elaborado en un comienzo, esta vez, elaborado a partir de múltiples compuertas lógicas establecidas en las librería de Synopsys colocada como *target library*.

Una comparación del circuito original y el circuito sintetizado se presenta en la Figura #8.11.

```
module fulladder
(
 input x,
 input y,
 input cin,

 output A,
 output cout
 );

wire  p,r,s;
       xor (p,x,y);
       xor (A,p,cin);

       and(r,p,cin);
       and(s,x,y);
       or(cout,r,s);

endmodule
```

(a)

```
module fulladder ( x, y, cin, A, cout );
  input x, y, cin;
  output A, cout;
  wire   n2;

  AO22X1 U4 ( .IN1(cin), .IN2(x), .IN3(y), .IN4(n2), .Q(cout) );
  XOR2X1 U5 ( .IN1(y), .IN2(n2), .Q(A) );
  XOR2X1 U6 ( .IN1(cin), .IN2(x), .Q(n2) );
endmodule
```

(b)

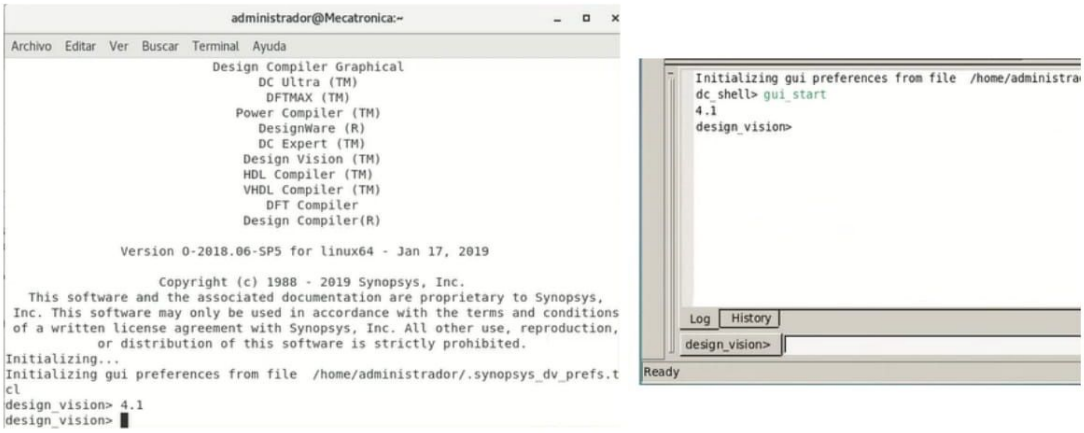
Figura 8.11: (a) Circuito elaborado en verilog. (b) *Netlist* de compuertas lógicas.

8.5. Prueba media: *Ripple Carry Adder*

Una segunda prueba fue realizada, esta vez elaborando un circuito estructural con compuertas lógicas y jerarquía *bottom-up*. El circuito a elaborar fue un *Ripple Carry Adder*, un sumador de 4 bits. Este se encontraba conformado de dos módulos, siendo el primero el circuito *Full Adder* utilizado previamente, y el segundo la interconexión en serie de cuatro veces el primer módulo.

La prueba realizada anteriormente presentaba el uso de la interfaz gráfica para la elaboración de la síntesis. Sin embargo, se presenta una mayor facilidad al utilizar la consola para elaborar los distintos pasos, además de ser de suma importancia poder realizar el procedimiento en consola, pues esto permite elaborar *scripts* que permitan la elaboración de todo el proceso en pocos segundos.

En esta prueba se presentará la síntesis del circuito utilizando comandos en consola. Los comandos pueden ejecutarse en la consola donde se encuentra activo *Design Vision*, como se muestra en la Figura #8.12(a). Al igual que en la parte inferior de la interfaz gráfica, como se muestra en la Figura #8.12(b).



(a) (b)

Figura 8.12: (a) Consola. (b) Interfaz gráfica.

Se comenzó colocando las librerías a utilizar. Para la síntesis de esta prueba se utilizaron las mismas librerías presentadas en la prueba anterior. Para colocar las librerías utilizando la consola

se utilizan los siguientes comandos:

design_vision> lappend search_path /usr/synopsys/PDK/SAED90_EDK/SAED_EDK90nm_REF /references/PARSER/ref/models/

design_vision> set link_library "* saed90nm_max.db saed90nm_min.db saed90nm_typ.db"

design_vision> set target_library "saed90nm_typ.db"

El comando 'lappend search_path' es utilizado para indicar la ubicación de las librerías. Los comandos 'set link_library' y 'set target_library' se utilizan para seleccionar los archivos a utilizar. Una vez ejecutados estos comandos las librerías se establecen en la herramienta, los siguientes comandos ejecutan las opciones *Analize* y *Elaborate*:

design_vision> analyze –library WORK –format verilog /home/administrador/RCA.v

design_vision> elaborate –architecture verilog –library WORK /home/administrador/RCA.v

También es posible ejecutar el comando *read_file*, el cual ejecuta la opción *Read*. Esta opción se encarga de ejecutar las opciones *Analyze* y *Elaborate*.

design_vision> read_file -format verilog /home/administrador/RCA.v

Posterior a realizar la opción *Elaborate* o *Read* fue posible observar la siguiente representación de los módulos, mostrada en la Figura #8.13. Pudiendo observar que *Design Vision* elabora una representación de los módulos tomando en cuenta su jerarquía y permite observar como cada uno de estos se encuentra elaborado.

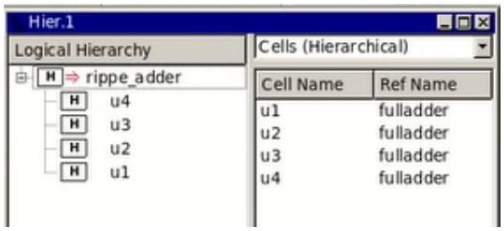


Figura 8.13: Representación jerárquica de módulos en *Design Vision*.

Al observar el circuito esquemático elaborado por *Design Vision* este se presenta elaborado tomando en cuenta la jerarquía utilizada en el diseño. La Figura #8.14 permite ver cómo este se desarrolla al adentrarse en su arquitectura.

Una vez cargado el archivo verilog se prosiguió colocando las restricciones del diseño. Para este circuito se utilizaron restricciones para área, condiciones de operación y *delay* en el circuito, estos fueron los siguientes:

design_vision> reset_design

design_vision> set_max_area 0

design_vision> set_operating_conditions TYPICAL

design_vision> set_min_delay 1 -from [all_inputs]

design_vision> set_max_delay 2 -from [all_inputs]

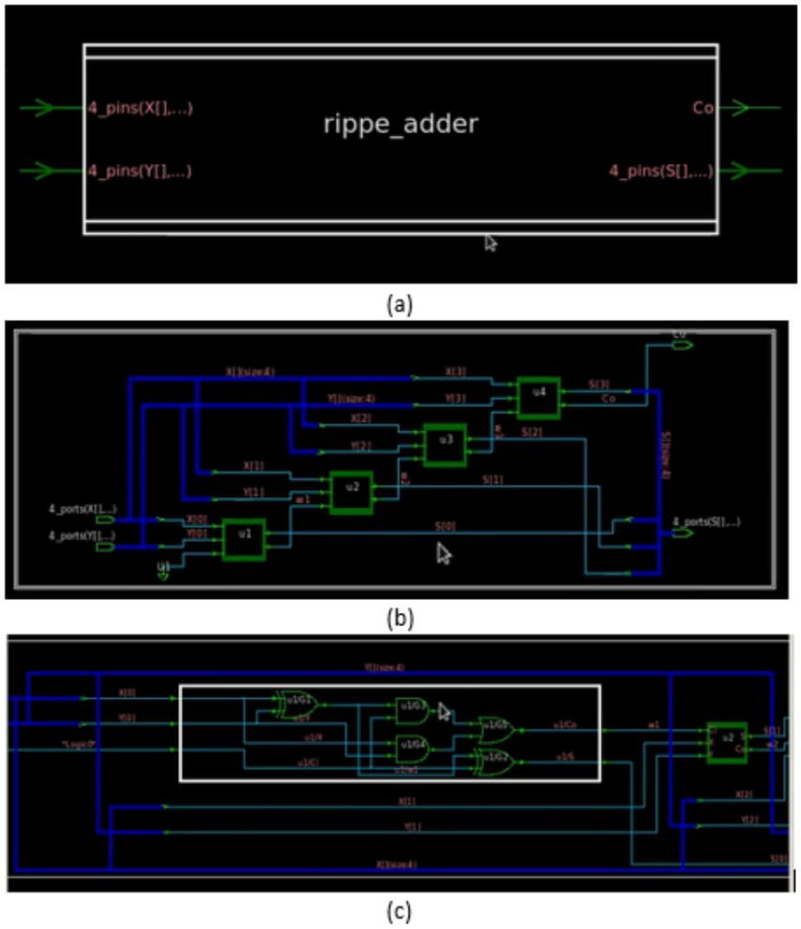


Figura 8.14: (a) Representación de caja negra del circuito. (b) Representación del circuito por medio de los módulos que lo conforman. (c) Modulo fundamental del circuito.

Al observar el esquemático del circuito luego de colocar las restricciones es posible notar como este es modificado debido a estas reglas de diseño. El principal cambio se debe a las restricciones de *delay*, las cuales establecen un tiempo de entre 1 y 2 ns para que las señales de entrada se estabilicen. El esquemático del circuito tomando en cuenta las restricciones se presenta en la Figura #8.15

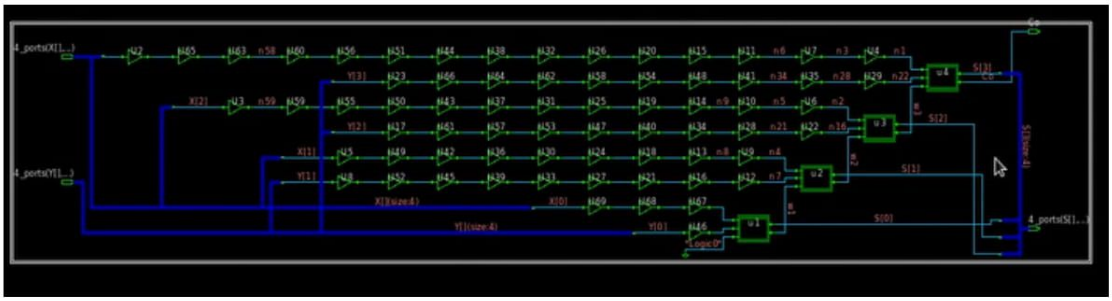


Figura 8.15: Esquemático de circuito *RCA* con restricciones.

El motivo por el cual el circuito utiliza tantos *buffers* en sus entradas se debe a que el tiempo

establecido en las restricciones es demasiado lento para un circuito de estas dimensiones, por lo que debe utilizar todos estos componentes para poder cumplir con las reglas de diseño establecidas. Una versión más adecuada del circuito, donde este si presenta una optimización se presenta en la Figura #8.16, en este caso las restricciones de tiempo fueron reemplazados por los siguientes:

design_vision> set_min_delay 0.0000001 -from [all_inputs]

design_vision> set_max_delay 0.0000002 -from [all_inputs]

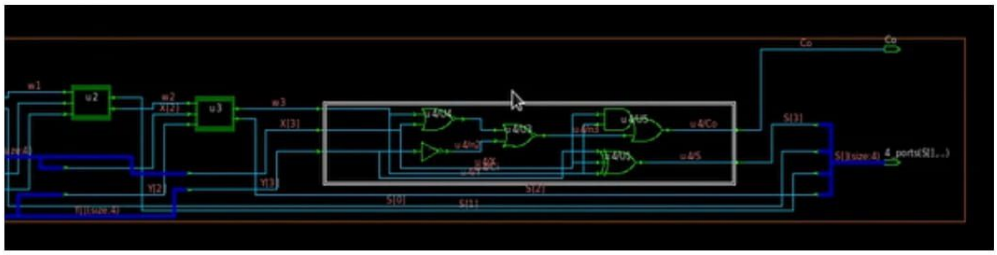


Figura 8.16: Esquemático de circuito *RCA* con restricciones de tiempo adecuadas.

En este caso solamente el bloque fundamental del circuito se modifico para poder cumplir con las especificaciones de tiempo establecidas, permitiendo obtener una mayor velocidad a la generada originalmente por el circuito.

Una vez establecidas las restricciones se prosiguió verificando el diseño por medio de la opción *Check Design* y elaborando el *netlist* por medio de la opción *Compile Design*. Para ejecutar estas opciones en consola se utilizan los siguientes comandos:

design_vision> check_design

design_vision> compile -map_effort medium -area_effort medium

La obtención de reportes puede realizarse por medio de los comandos mostrados a continuación. Si se desea guardar el reporte basta con colocar un símbolo '>' junto al comando seguido de la dirección donde se guardará el reporte.

design_vision> report_area > reportes/area.rpt

design_vision> report_cell > reportes/celdas.rpt

design_vision> report_timing > reportes/timing.rpt

En la prueba básica presentada anteriormente se mostró como guardar el *netlist* de compuertas lógicas. A continuación, se presentará el comando mediante el cual es posible guardar el *netlist*, además de esto, para la presente prueba se guardarán dos archivos extra.

El primero será un archivo con extensión '.ddc' y el segundo un archivo '.sdc', la importancia de estos archivos se presentara al continuar con el desarrollo del flujo de diseño. Una vez elaborado el *netlist* de compuertas lógicas y verificado su funcionamiento se continuará el flujo de diseño por medio de la herramienta *ICC*.

ICC permitirá obtener una representación del circuito en silicio, para hacer esto requerirá el *netlist* de compuertas lógicas y el archivo con extensión '.ddc'. El archivo con extensión '.ddc' almacena las restricciones establecidas para el circuito.

La herramienta *ICC* presenta una segunda opción, la cual consiste en el uso del archivo '.sdc', este archivo puede utilizarse en lugar del *netlist* de compuertas lógicas y el archivo '.ddc', pues

contiene tanto el circuito sintetizado como las restricciones establecidas para este.

En caso de decidir utilizar el archivo '.sdc' es de importancia guardar también el *netlist* de compuertas lógicas, pues este deberá de ser sometido a pruebas para determinar que su funcionamiento sea el correcto. A continuación se presentan los comandos para guardar los distintos archivos.

design_vision> write -hierarchy -format verilog -output .../RCA/RCA_syn.v

design_vision> write -format ddc -h -o .../RCA/RCA_ddc.ddc

design_vision> write_sdc .../RCA/RCA_sdc.sdc

CAPÍTULO 9

Elaboración de síntesis

9.1. Diseño del circuito estructural

El diseño a ser elaborado consiste en un circuito el cual contará con 12 pines. 2 pines destinados a voltaje y tierra, 1 pin destinado a una entrada de reloj, 1 pin destinado a una entrada de reset y 8 pines de salida. Al recibir un pulso de reloj los ocho pines de salida mostrarán una representación de 8 bits de una letra. Con calda pulso de reloj la letra mostrada en los pines de salida cambiará, el objetivo de esto es imprimir un mensaje, el cual será el siguiente:

"SOY EL PRIMER CHIP CON TECNOLOGÍA NANOMÉTRICA DISEÑADO POR UN PROGRAMA DE LICENCIATURA DE UNA ESCUELA DE INGENIERÍA EN LA HISTORIA DE CENTROAMETRICA. MIS CREADORES SON: CARLOS ESQUIT, DIEGO SOLER CASTAÑEDA, STEVEN HIRAM RUBIO, LUIS ARTURO NAJERA Y PABLO ORTIZ BARILLAS. DEPARTAMENTO DE INGENIERIA ELECTRONICA. UNIVERSIDAD DEL VALLE DE GUATEMALA. 2019"

El mensaje será interpretado por un microcontrolador con la finalidad de poder ser mostrado en un ordenador.

Un primer acercamiento a la elaboración del diseño en verilog del circuito se dio mediante la propuesta de elaborarlo como una máquina de estados finita. Esto con la finalidad de que el circuito elaborado en verilog tuviera un nivel de abstracción estructural.

Se comenzó elaborando un Excel para organizar las transiciones entre cada letra que serían necesarias hacer, la representación de las letras en código *ASCII* y corroborar que no existiera ningún error en el mensaje.

Posteriormente se utilizo el software *Logic Friday* para elaborar el esquemático de la maquina de estados finitos. Tras colocar todas las transiciones requeridas en el software se genero el esquemático y fue posible observar que este presentaba una alta complejidad, representando un circuito de grandes dimensiones.

El paso a seguir consistía en elaborar el circuito obtenido por medio de *Logic Friday* en verilog,

lo cual significaría una gran cantidad de horas de trabajo y una alta posibilidad de cometer errores en las conexiones. Debido a esto se decidió realizar una segunda aproximación a la elaboración del circuito.

9.2. Diseño del circuito *behavioral*

El principal motivo por el cual se decidió elaborar el circuito de forma estructural utilizando una maquina de estados finitos se debió a que los diseños estudiados en las referencias presentaban diseños estructurales de sus circuitos a sintetizar. Sin embargo, al realizar una investigación de la herramienta *Design Vision* fue posible comprobar que esta realizaba una síntesis correctamente al utilizar circuitos *behavioral*.

Se realizó una primer prueba elaborando un archivo verilog *behavioral* para una compuerta *Not*, obteniendo un resultado correcto por parte de la herramienta. Por lo que se decidió realizar el circuito que se deseaba fabricar utilizando este nivel de abstracción.

Se busca que el circuito muestre una letra por medio de sus pines de salida al momento de recibir un pulso de reloj. Para el mensaje anteriormente mostrado se presenta un total de 346 caracteres.

Para elaborar el circuito se comenzó desarrollando un contador de 9 bits limitado a alcanzar el numero 346, posteriormente el contador se reinicia. Utilizando las condicionales *if* y *else if* es posible determinar la letra que el circuito presenta. La lógica del circuito consistía en que, en caso de encontrarse el contador en 0 los pines de salida mostrarán los ocho bits que conforman la letra 'S', en caso de encontrarse el contador en 1 los pines de salida mostrarán los ocho bits que conforman la letra 'O', este proceso continua hasta realizar los 346 estados y reiniciar.

De esta forma fue posible elaborar el circuito en forma *behavioral* para obtener los resultados deseados. Tanto el contador como las condicionales fueron elaborados en un solo modulo. La funcionalidad del diseño se comprobó elaborando un módulo estimulo, utilizando icarus verilog e imprimiendo la respuesta mostrada por las salidas del circuito.

Cada una de las 346 respuestas mostradas por el circuito fue verificada, resolviendo cualquier equivocación en caso de existir alguna. Parte del circuito descriptivo en verilog elaborado se presenta en la sección de anexos, y el circuito completo puede encontrarse en el repositorio presentado en [13].

9.3. Síntesis del circuito a fabricar

Una vez elaborado el circuito se prosiguió a realizar la síntesis de este. Se trabajo solamente en consola y se elaboró un *script* que puede encontrarse en la sección de anexos al igual que en el repositorio en [13].

Posterior a ejecutar el *script* se presentaron dos modulo en *Design Vision*, la Figura#9.1 muestra su representación en la herramienta.

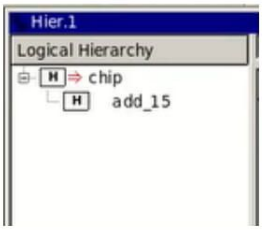


Figura 9.1: Representación de módulos para circuito elaborado en modo *behavioral*

Las restricciones utilizadas pueden observarse en la sección de anexos al igual que en repositorio presentado en [13]. Al utilizar restricciones la herramienta optimizo el circuito acorde a las especificaciones colocadas. En caso de no utilizar restricciones la herramienta elaboraría la síntesis de forma correcta, sin embargo, el circuito presentaría mayores dimensiones a la vez que operaria a una frecuencia menor. La Figura #9.2 presenta una comparativa entre los esquemáticos del circuito elaborado sin el uso de restricciones (a) y utilizando restricciones (b). Esta comparativa nos permite comprender la importancia del uso de restricciones en el diseño del circuito.

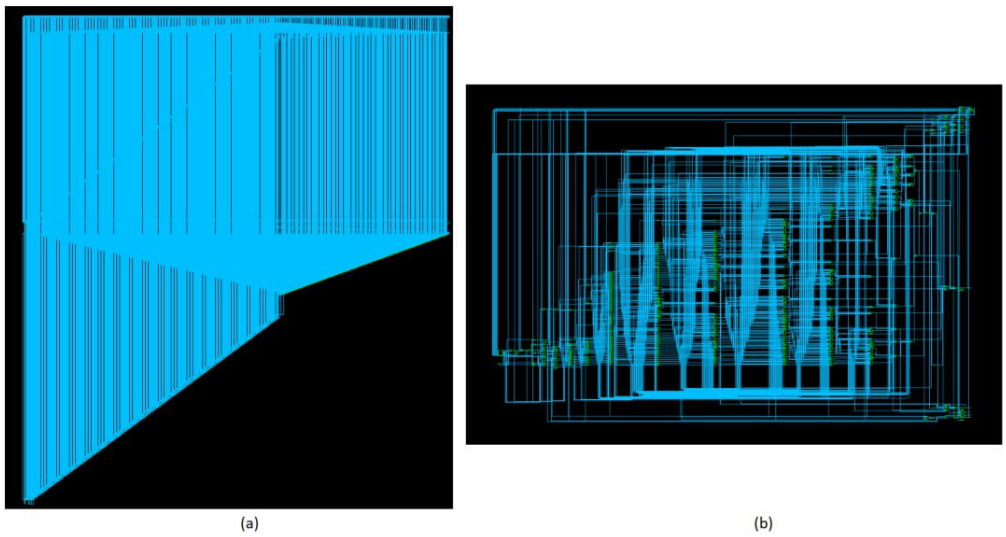


Figura 9.2: (a) Esquemático del circuito sin el uso de restricciones. (b) Esquemático del circuito utilizando restricciones.

El motivo por el cual el esquemático de la Figura#9.2(a) se presenta como un bloque de color azul se debe a las conexiones del circuito. Las conexiones son representadas en color azul por *Design Vision*, mientras que las compuertas lógicas son representadas con color verde. No es posible observar a simple vista las compuertas lógicas debido al tamaño de estas en relación con el tamaño del circuito, y debido a la cantidad de conexiones que este posee.

Al realizar un acercamiento del esquemático es posible observar las compuertas y demás bloques que lo conforman. Una imagen de un segmento del circuito se presenta en la Figura #9.3.

A pesar de ser un circuito que cumpla con una tarea sencilla la lógica que requiere su elaboración presenta un nivel alto de desarrollo, motivo por el cual utiliza una alta cantidad de compuertas lógicas y una alta cantidad de conexiones entre estas. Al utilizar restricciones esta cantidad se reduce, permitiendo su elaboración de forma optimizada, como se muestra en la Figura #9.2(b).

Dentro del *script* se establece el comando para guardar el archivo '.sdc' generado durante la síntesis, mientras que el *netlist* de compuertas lógicas se guardo por medio de la interfaz gráfica.

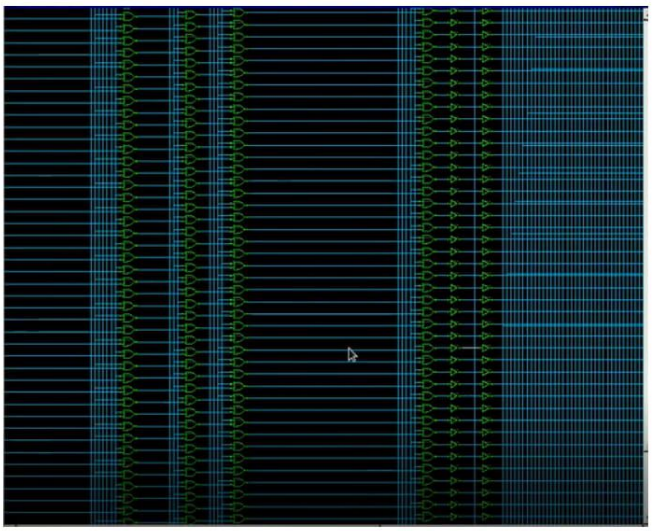


Figura 9.3: Acercamiento del esquemático del circuito sin el uso de restricciones.

CAPÍTULO 10

Verificación de funcionamiento

10.1. Verificación en *VCS*

La herramienta *VCS* se utilizará como parte de la verificación de funcionamiento de los circuitos. Esta permite observar el funcionamiento de los circuitos pre y post síntesis al colocar un estimulo en estos, por lo que, previo a su uso es necesario elaborar un modulo estimulo para el circuito que se desea verificar.

Los módulos estímulos utilizados para esta verificación se presentarán en la sección de anexos al igual que en el repositorio presentado en [13]. A continuación, se presentará el proceso mediante el cual se verificó el funcionamiento del circuito que se desea fabricar tanto antes como después de su síntesis.

Previo a comenzar a utilizar la herramienta se preparo una carpeta con los archivos que se necesitarían para la verificación, Figura #10.1. Estos archivos son, el estímulo para el circuito pre-síntesis, nombrado *testBench*. El estimulo para el circuito post-síntesis, nombrado *testBench_s*. La librería 'Saed90nm.v', la cual contiene la definición de todas las compuertas lógicas utilizadas en la síntesis en lenguaje verilog, y el *script* a utilizar.

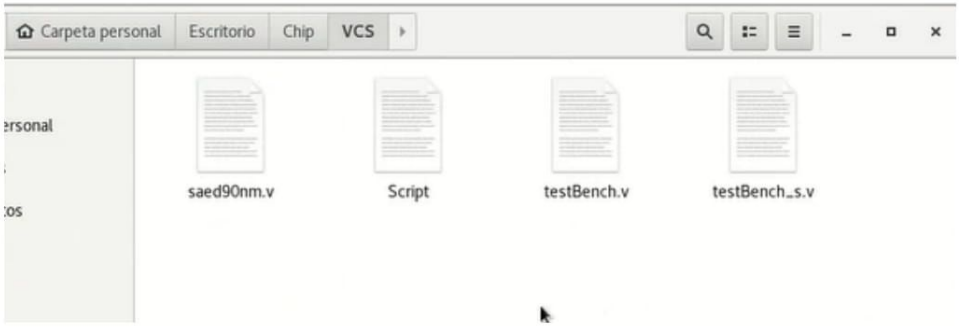


Figura 10.1: Archivos requeridos para verificación.

Una vez se tienen estos archivos preparados es posible ejecutar la herramienta. Al día que se redacta este trabajo, debido a que ciertos parámetros de la herramienta aun no han podido ser establecidos en el sistema, esta se ejecuta por medio de los siguientes comandos en consola:

```
VCS_HOME=/usr/synopsys/vcs-mx
export VCS_HOME
PATH= $VCS_HOME/bin: $PATH
export PATH
PATH=/usr/synopsys/verdi/ bin:$PATH
export PATH
```

Posterior a ejecutar los comandos anteriormente mostrados VCS se ejecuta en consola. Para elaborar la verificación del circuito pre-síntesis se deberá de ejecutar el comando que se mostrará a continuación.

vcs -Mupdate -RPP -v .../chip.v .../testBench.v -o demo -full64 -debug all

-/demo -gui

La función de los argumentos del comando se presenta en la Tabla #10.1. Estos argumentos son los mayormente utilizados al momento de verificar un circuito pre sintetizado en *VCS*. El comando final ejecuta la salida del análisis realizado al circuito en la interfaz gráfica, lo cual permite observar el comportamiento del circuito, como se presentará a continuación.

Argumento	Función
Mupdate	Crea una carpeta csrc donde se almacenaran todos los archivos de la simulación.
RPP	Establece que el diseño se abrirá en post procesamiento, es decir, vcs ejecutara la simulación una vez que se termine de analizar el circuito.
v	Ejecuta el comando en modo 'verbose', es decir, se mostrará en consola todos los comandos que la herramienta ejecute automáticamente. Esto permite obtener más información del proceso que la herramienta realiza.
o	Comando out, se le coloca un nombre para ejecutar la salida. En este caso 'demo'.
Debug all	Permite utilizar la interfaz gráfica una vez la herramienta termine de analizar el circuito.
Full64	Especifica que el circuito será copilado y simulado en una computadora de 64 bits.

Tabla 10.1: Argumentos del comando para analizar el circuito pre-síntesis en *VCS*.

Una vez ejecutados los comandos se abrirá la interfaz gráfica, esta desplegara en su lado izquierdo un listado con los módulos que se encuentran en el circuito, en este caso solo se presentara el módulo 'stimulus' y el módulo 'chip'. En caso de hacer click sobre alguno de estos módulos se despegará en la parte media de la interfaz las entradas y salidas de este, y en la parte derecha de la interfaz el código del módulo.

Se deberá de hacer click derecho en el modulo estimulo y posteriormente seleccionar las opciones *add To Waves* y *New Wave View*, como se muestra en la Figura #10.2. Al seleccionar estas opciones se desplegará una nueva interfaz como se muestra en la Figura #10.3.

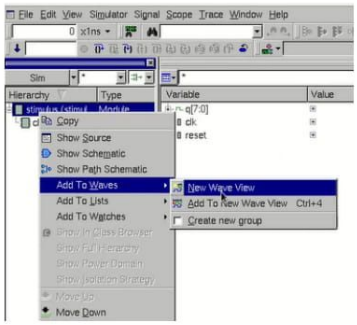


Figura 10.2: Opciones *add To Waves* y *New Wave View* en *VCS*.

Esta nueva interfaz mostrara de forma gráfica el comportamiento del circuito, para esto es necesario ir a la pestaña *Simulator* y elegir la opción *Start/Continue*, esto hará que la herramienta muestre la simulación de entradas y salidas del circuito.

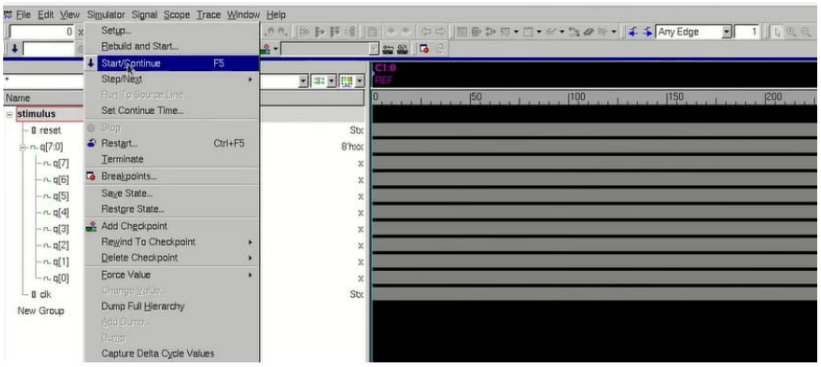


Figura 10.3: Pestaña *Simulator* y elegir la opción *Start/Continue* en *VCS*.

La simulación solamente mostrara una parte de las salidas y entradas del circuito, para observar todo el comportamiento debe de hacerse click derecho sobre la simulación y elegir las opciones *Zoom* y *Zoom Full*.

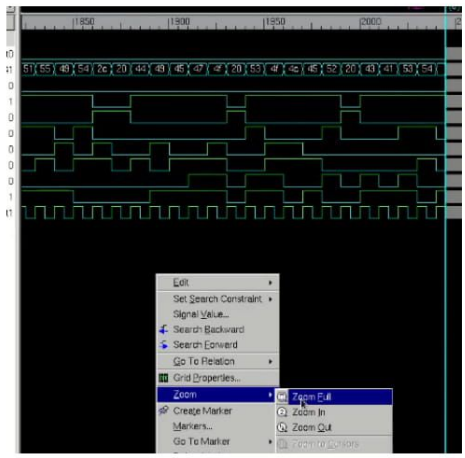


Figura 10.4: Opciones *Zoom* y *Zoom Full* en *VCS*.

Posterior a seleccionar las opciones antes mencionadas se desplegará la simulación del circuito con todas las entradas y salidas definidas en el estímulo.

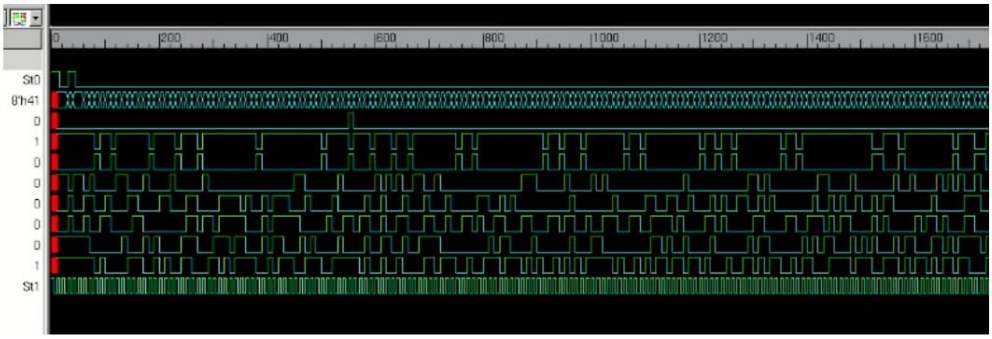


Figura 10.5: Simulación del circuito pre-síntesis.

El proceso para realizar la verificación del circuito sintetizado es similar a la mostrada. Se comenzará ejecutando los mismos comandos presentados al comienzo para ejecutar la herramienta en consola. Posteriormente se ejecutará el comando mostrado a continuación.

vcs -V -R . . . /saed90nm.v . . . /chip_syn.v . . . /testBench_s.v -o demo -full64 -debug all

./demo -gui

El argumento R ejecutara la simulación al terminar el análisis, los demás argumentos elaboraran las tareas antes descritas. En este caso debe de incluirse el archivo saed90nm.v, pues este permitirá el análisis del circuito.

Al abrir la interfaz gráfica se presentará una lista bastante extensa de módulos, esto se debe a que cada compuerta utilizada es considerada un modulo del circuito. Se deberá seleccionar el modulo estimulo y repetir el proceso antes descrito.

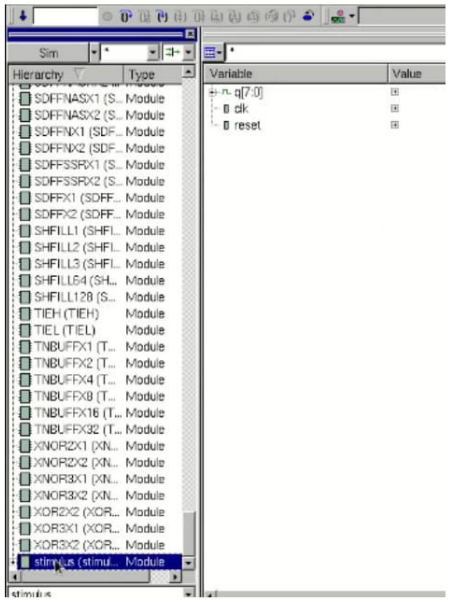


Figura 10.6: Módulos de circuito post-síntesis.

Al finalizar deberemos obtener una respuesta igual por parte de ambos circuitos. En caso de no ser así se deberá verificar todo el procedimiento de síntesis y colocación de *constraints* para determinar el error.

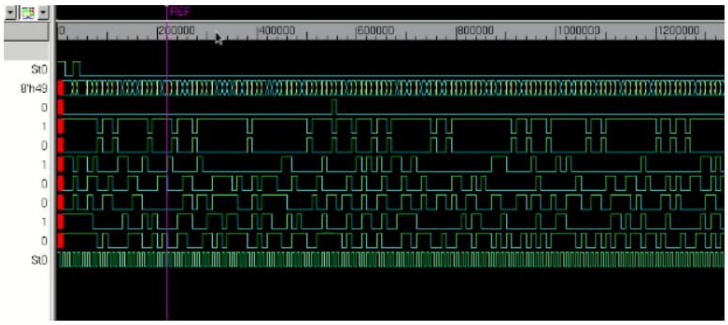


Figura 10.7: Simulación del circuito post-síntesis.

10.2. Verificación en *Formality*

La herramienta *Formality* permite determinar si dos circuitos son equivalentes entre sí, es decir, si dos circuitos elaborados en verilog cumplen con la misma tarea a pesar de haber sido elaborados de forma distinta. Esto permite verificar si el circuito elaborado originalmente y el circuito sintetizado son equivalentes.

Cabe mencionar que el uso de la herramienta no se basa únicamente en la comparación *RTL-Netlist*, esta permite elaborar comparaciones *RTL–RTL* o *Netlist–Netlist* en caso de ser necesario. La importancia de determinar equivalencia entre los circuitos pre y post síntesis se basa en que, al elaborar esta prueba podemos asegurar que todo el proceso de síntesis ha sido exitoso, ya que ambos circuitos elaboran la misma tarea de forma correcta.

Como se ha presentado anteriormente, este proceso puede elaborarse igualmente en *VCS*, al elaborar un estímulo y verificar todas las salidas de ambos circuitos. Sin embargo, en caso de que el circuito elaborado posea una cantidad sumamente alta de vectores de entrada y salida, *Formality* permitirá elaborar la comprobación sin la necesidad de verificar manualmente cada salida de ambos circuitos.

Para realizar la verificación del circuito se comienza abriendo la herramienta, la cual desplegará una interfaz gráfica. En la presente explicación se mostrará como utilizar la herramienta por medio de su interfaz gráfica.

La interfaz gráfica presentara en su parte superior una serie de pestañas numeradas, las cuales se presentan en la Figura #10.8. Estas pestañas representan los pasos a seguir para elaborar la verificación del circuito, al momento de abrir la herramienta esta se encontrará en la pestaña 0. *Guid.*

Figura 10.8: Pestañas numeradas con los pasos a seguir para la verificación.

Esta pestaña presenta la opción de cargar archivos (.svf), este tipo de archivos ayudan a la herramienta a elaborar la verificación tomando en cuenta cambios elaborados en el diseño por otras herramientas previamente. El uso de estos archivos se presenta en diseños de mayores dimensiones y en donde se requiere de una verificación en puntos específicos del circuito, por lo que no se utilizará en el presente trabajo.

La segunda pestaña, 1. *Ref.*, presentada en la Figura #10.9, es utilizada para cargar el primer diseño a comparar, en este caso el circuito pre-síntesis. Esta ventana resulta altamente intuitiva, en esta se presentan opciones para elegir el tipo de archivo a cargar, ya sea este Verilog, SystemVerilog, VHDL, DB o DDC. Igualmente presenta la opción de elegir la librería del diseño, está por será *WORK* por defecto, y el año de la versión de verilog o versión del archivo cargado.

También es posible observar que esta pestaña presenta opciones numeradas, estas opciones serán los pasos a seguir para cargar el diseño. El primer paso será cargar el circuito pre-sintetizado. El segundo paso cargar la librería .db, debido a que este circuito aun no ha sido sintetizado no se ha utilizado esta librería por lo que esta pestaña se deja en blanco. El tercer paso será definir el módulo jerárquicamente más alto del diseño, es decir, el módulo principal del circuito.

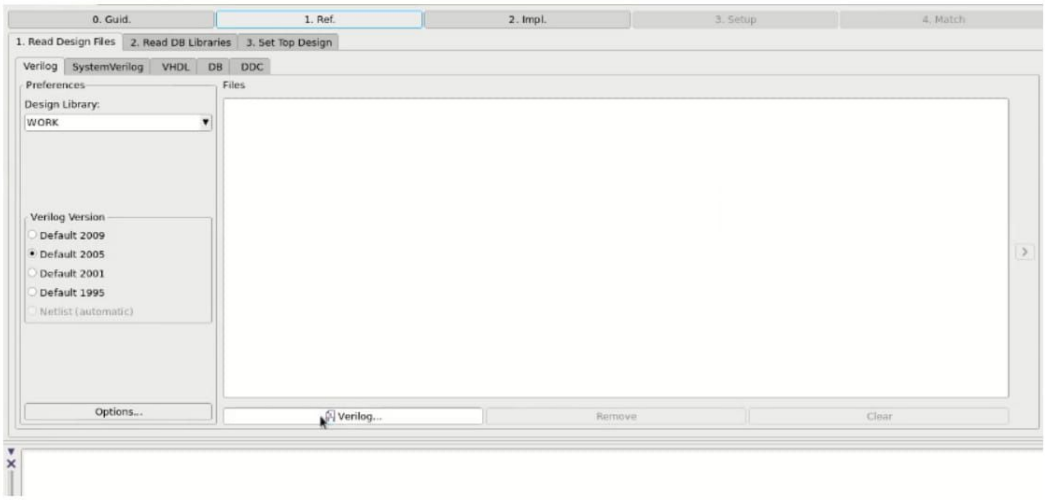


Figura 10.9: Pestaña 1. *REF*. Utilizada para cargar el diseño pre-síntesis.

La tercer pestaña, 2. *Impl.*, presentada en la Figura #10.10, será totalmente igual a la presentada en la segunda pestaña. El proceso descrito anteriormente debe de repetirse acá con dos diferencias, se deberá cargar el archivo verilog del circuito sintetizado y se deberá cargar la librería (.db) utilizada para sintetizar el circuito. Posteriormente se debe definir el módulo jerárquicamente más alto del diseño y es posible continuar.

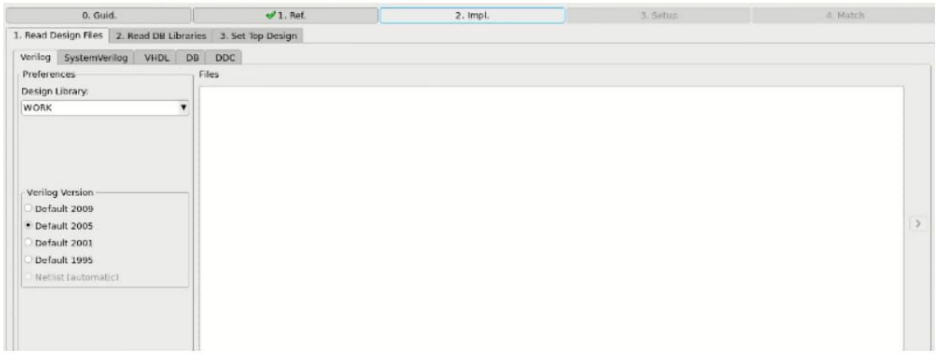


Figura 10.10: Pestaña 2. *Impl.* Utilizada para cargar el diseño post-síntesis.

La cuarta pestaña, 3. *Setup.* Permite colocar información extra previo a la verificación de los circuitos. Por ejemplo, en el caso de que el diseño presente regiones las cuales sabemos que no concordaran o que deseamos que no sean analizadas, esta pestaña nos permite establecer esas áreas y anunciar a *formality* que omita su verificación. O si se desea analizar solo cierto sector del circuito, es posible establecerlo en esta pestaña.

Al igual que la primera pestaña, 0. *Gui.*, esta pestaña se utiliza cuando se desea un análisis más específico del circuito. En nuestro caso no deseamos un análisis con requisitos especiales por lo que dejamos esta pestaña en blanco.

La quinta pestaña, 4. *Match.*, será la primera de dos verificaciones que realizará la herramienta. En esta pestaña bastará con presionar el botón *Run Matching* para que la herramienta verifique que existe una concordancia entre los diseños.

El funcionamiento de esta prueba se basa en tomar el circuito pre-sintetizado y el circuito sintetizado y dividir ambos en bloques iguales. Posteriormente la herramienta tomara el primer bloque del circuito pre-sintetizado y buscara modificarlo haciendo uso de la librería (.db) proporcionada, si la herramienta es capaz de hacer que este primer bloque pre-sintetizado sea igual al primer bloque del circuito sintetizado se presentará un *match* en la herramienta.

Este proceso se repite con cada bloque, en caso de no presentarse ningún error, la herramienta habrá verificado que ambos circuitos son equivalentes en su construcción.

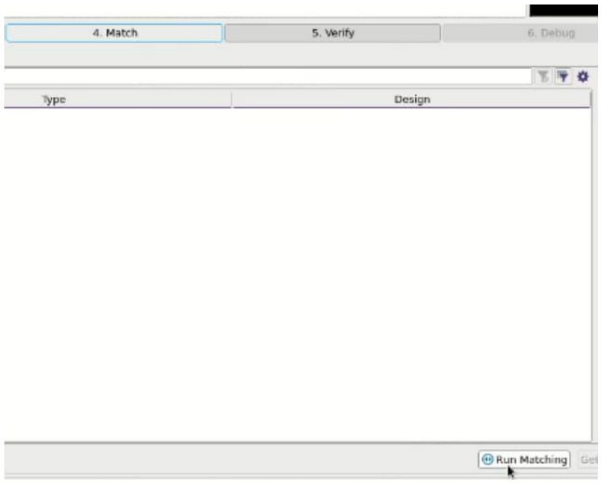


Figura 10.11: Pestaña 4. *Match.*

La sexta pestaña, 5. *Verify.*, elaborará la segunda verificación entre los circuitos. Al igual que con la pestaña anterior bastará con presionar el botón *verify* para elaborar la verificación. Esta comprobación se basa en el uso de algoritmos en las entradas de los circuitos y verificación de sus salidas para determinar que ambos se comportan de igual forma ante los mismos estímulos.

Formality cuenta con distintos algoritmos desarrollados para probar el funcionamiento de circuitos y asegurar que ambos funcionan exactamente igual. De no presentarse ningún error la herramienta habrá verificado que ambos circuitos son equivalentes en funcionamiento.

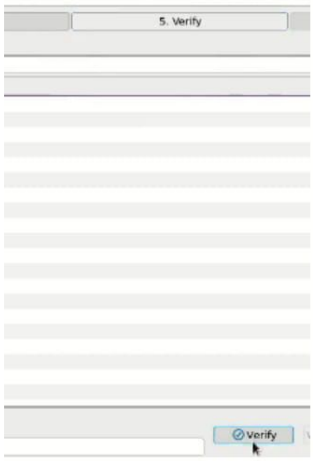


Figura 10.12: Pestaña 5. *Verify.*

La séptima pestaña, 6. *Debug*, presentará un listado de los distintos bloques en que el circuito fue dividido. Al hacer click derecho sobre cualquier elemento de esta lista se presentará la posibilidad de observar información especifica sobre este, como su esquemático o código pre y post síntesis, una tabla de verdad de su funcionamiento, entre otras características.

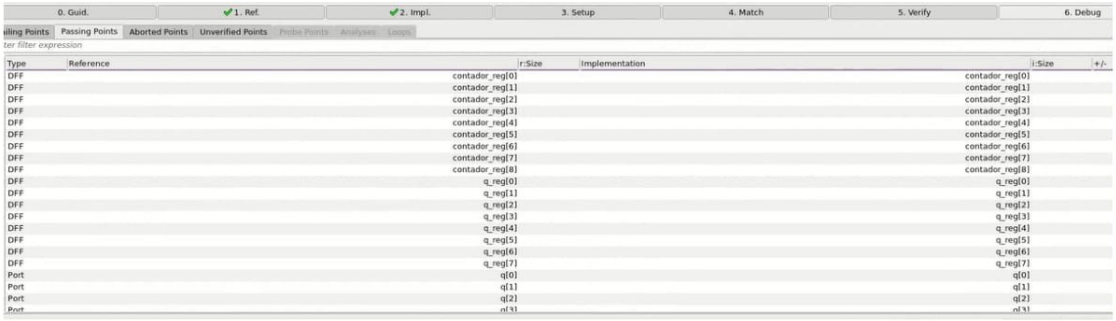


Figura 10.13: Pestaña 6. *Debug.*

CAPÍTULO 11

Conclusiones

- A través de las pruebas realizadas se comprobó que la herramienta *Design Vision* presenta las características necesarias para la elaboración de síntesis de circuitos elaborados en lenguaje descriptivo de *hardware* a *netlist* de compuertas lógicas. Siendo esta la herramienta a utilizar para elaborar la primera parte del flujo de diseño.
- Se comprobó cómo el uso de restricciones de diseño en los circuitos permite obtener *netlists* de compuertas lógicas con los circuitos optimizados, permitiendo mejorar características especificas que el diseñador desee, tales como el área o *delay* del circuito.
- Se utilizaron las herramientas *VCS* y *Formality* para estudiar el comportamiento del circuito tanto antes como después de la síntesis. Se obtuvo un mismo comportamiento para ambos circuitos lo cual permitió comprobar que la síntesis se realizo de forma correcta y que ambos circuitos realizaban la tarea deseada.
- Se documentó el funcionamiento de las herramientas *Design Vision*, *VCS* y *Formality* por medio del presente trabajo y de videotutoriales donde se muestran diversos ejemplos.

CAPÍTULO 12

Recomendaciones

Dentro de un proyecto tan novedoso como este, existe la posibilidad de haber pasado por alto múltiples elementos que puedan proveer mejores resultados al trabajo realizado. Este trabajo representa la primer parte de un flujo de diseño de circuitos nanométricos, el cual será utilizado por la Universidad del Valle de Guatemala y sus futuros estudiantes para el desarrollo de circuitos integrados.

Se recomienda a futuros estudiantes que tengan interés en el proyecto la constante actualización de las herramientas de trabajo que se utilicen. Las herramientas de Synopsys utilizadas en el presente trabajo, así como las utilizadas en todo el flujo de diseño, presentan la desventaja de requerir una actualización manual, por lo que se recomienda a los estudiantes verificar si existen actualizaciones de las herramientas ya que estas pueden proveer nuevas características que ayuden de una u otra forma en el diseño del circuito.

De igual forma, Synopsys cuenta con una cantidad muy grande de herramientas y constantemente realiza nuevos lanzamientos. Se recomienda el estudio de nuevas herramientas las cuales puedan beneficiar al flujo de diseño.

En caso se desee llevar a cabo la instalación de una nueva herramienta se recomienda referirse al trabajo [15], donde se documenta todo el proceso de instalación de herramientas de forma clara.

En el presente trabajo se presentaron las restricciones mayormente utilizadas, sin embargo, existe una basta cantidad de reglas de diseño que pueden establecerse en el circuito, por lo que, en caso se desee elaborar un diseño con características más demandantes se recomienda referirse al manual de *Design Vision* [6] donde se presenta un listado de restricciones o al sitio en [11].

En el presente trabajo se verificó el funcionamiento del circuito antes y después de la síntesis utilizando las herramientas de Synopsys, sin embargo, si el estudiante desea elaborar una verificación física se recomienda utilizar un *FPGA* en conjunto con el software Altium. Bastará con cargar el archivo del circuito sintetizado en formato verilog, la librería utilizada en formato verilog y el módulo estímulo a la *FPGA* para comprobar el funcionamiento del *netlist* de forma física.

Se recomienda al estudiante explorar las distintas opciones que presentan las herramientas de Synopsys en sus interfaces gráficas. Debido a que cada una de estas herramientas presenta una cantidad enorme de opciones no ha sido posible documentar cada una de ellas, por lo que se recomienda al estudiante explorar la herramienta mediante su interfaz gráfica con el objetivo de encontrar nuevas características que puedan aportar beneficios al diseño que se realice.

CAPÍTULO 13

Bibliografía

[1] Aswale, Pramod S, Mukesh P Mahajan, Manjul V Nikumbh y Omkar S Vaidya: *Implementation of Baugh-Wooley Multiplier and Modified Baugh Wooely Multiplier using Cadence (Encounter) RTL.* International Journal of Science, Engineering and Technology Research (IJSETR), 4(2):293–298, 2015.

[2] Bhatnagar, Himanshu: *Advanced ASIC chip synthesis.* Springer, 2002.

[3] Castellanos Pellecer, Mónica Lorena: *Impacto del ruido por acople capacitivo en las violaciones de estado en nodos con operación subumbral en sistemas de 180, 90 y 65 nanómetros.* Facultad de Ingeniería Universidad del Valle de Guatemala, Guatemala, tesis licenciatura en ingeniería electrónica edición, 2016.

[4] Chinnery, David y Kurt Keutzer: *Closing the gap between ASIC & custom: tools and techniques for high-performance ASIC design.* Springer Science & Business Media, 2002.

[5] Comer, David J: *The changing face of circuit design.* IEEE Potentials, 25(3):26–30, 2006.

[6] Compiler, Design, R User y Modeling Guide: *Synopsys.* Inc., see http://www. synopsys. com, 2001.

[7] Dhem, Jean Francois y Nathalie Feyt: *Hardware and software symbiosis helps smart card evolution.* IEEE Micro, (6):14–25, 2001.

[8] Finger, Susan y John R Dixon: *A review of research in mechanical engineering design. Part I: Descriptive, prescriptive, and computer-based models of design processes.* Research in engineering design, 1(1):51–67, 1989.

[9] Kommuru, Hima Bindu y Hamid Mahmoodi: *ASIC design flow tutorial using synopsys tools.* Nano-Electronics & Computing Research Lab, School of Engineering, San Francisco State University San Francisco, CA, Spring, 2009.

[10] Korcyl, Grzegorz y Piotr Korcyl: *Towards Lattice Quantum Chromodynamics on FPGA devices.* arXiv preprint arXiv:1810.04201, 2018.

[11] MicroIP: *Synopsys(DC).* Inc., see https://www.micro-ip.com/drchip.php?mode=2cid=8, 2014.

[12] Monmasson, Eric y Marcian N Cirstea: *FPGA design methodology for industrial control systems—A review.* IEEE transactions on industrial electronics, 54(4):1824–1842, 2007.

[13] Nájera, Luis Arturo y Rubio, Steven: *Repositorio Tesis: Implementación de circuitos sintetizados a nivel netlist a partir de diseños en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado.* Github, see https://github.com/naj15581/Tesis, 2019.

[14] Palnitkar, Samir: *Verilog HDL: a guide to digital design and synthesis.* Pearson Education India, 2003.

[15] Santos Chonay, Jonathan Alberto de los: *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.* Facultad de Ingeniería Universidad del Valle de Guatemala, Guatemala, tesis licenciatura en ingeniería electrónica edición, 2014.

[16] Synopsys: *VCS/VCSi User Guide.* Inc., see http://www. synopsys. com, 2005.

[17] Synopsys: *Formality User Guide.* Inc., see http://www. synopsys. com, 2007.

[18] Synopsys: *University Program: 90nm, 32/28nm Generic Libraries and iPDKs.* Inc., see http://www.europractice.stfc.ac.uk/vendors/SynopsysiPDK.pdf, 2007.

[19] Weste, Neil HE y David Harris: *CMOS VLSI design: a circuits and systems perspective.* Pearson Education India, 2015.

CAPÍTULO 14

Anexos

14.1. Reportes elaborados por *Design Vision*

```
****************************************
Report : area
Design : chip
Version: O-2018.06-SP5
Date   : Tue Jul 30 10:49:15 2019
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:

    saed90nm_typ (File: /usr/synopsys/PDK/SAED90_EDK/SAED_EDK90nm_REF
                        /references/PARSER/ref/models/saed90nm_typ.db)

Number of ports:                          28
Number of nets:                          459
Number of cells:                         426
Number of combinational cells:           406
Number of sequential cells:               17
Number of macros/black boxes:              0
Number of buf/inv:                        62
Number of references:                     32

Combinational area:               3210.610004
Buf/Inv area:                      342.860013
Noncombinational area:             505.958004
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  3716.568008
Total area:                 undefined
```

Figura 14.1: Reporte de área.

```
****************************************
Report : clocks
Design : chip
Version: O-2018.06-SP5
Date   : Tue Jul 30 10:50:00 2019
****************************************

Attributes:
    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
    G - generated_clock
    g - lib_generated_clock

Clock          Period   Waveform            Attrs     Sources
--------------------------------------------------------------------------------
clk             80.00   {0 40}                        {clk}
--------------------------------------------------------------------------------
```

Figura 14.2: Reporte de relojes utilizados.

```
****************************************
Report : cell
Design : chip
Version: O-2018.06-SP5
Date   : Tue Jul 30 10:49:30 2019
****************************************

Attributes:
   BO - reference allows boundary optimization
    b - black box (unknown)
    h - hierarchical
    n - noncombinational
    r - removable
    u - contains unmapped logic

Cell                    Reference       Library             Area  Attributes
--------------------------------------------------------------------------------
U401                    INVX0           saed90nm_typ    5.530000
U402                    INVX0           saed90nm_typ    5.530000
U403                    NOR2X0          saed90nm_typ    5.530000
U404                    NOR2X0          saed90nm_typ    5.530000
U405                    NAND2X0         saed90nm_typ    5.443000
U406                    NAND2X0         saed90nm_typ    5.443000
U407                    NAND2X0         saed90nm_typ    5.443000
U408                    OR4X1           saed90nm_typ    10.152000
U409                    AO221X1         saed90nm_typ    12.902000
U410                    NAND4X0         saed90nm_typ    8.294000
U411                    AO221X1         saed90nm_typ    12.902000
U412                    OA21X1          saed90nm_typ    9.216000
U413                    OA21X1          saed90nm_typ    9.216000
U414                    OA21X1          saed90nm_typ    9.216000
U415                    OA21X1          saed90nm_typ    9.216000
U416                    OA21X1          saed90nm_typ    9.216000
```

Figura 14.3: Reporte parcial de celdas.

14.2. Diseño en *VHDL behavioral* para circuito a fabricar

```
//Chip
module chip(q, reset, clk);

    output [7:0] q;
    input reset;
    input clk;

    reg [8:0] contador;
    reg [7:0] q;

    always @ (posedge reset or posedge clk)
    if (reset)
        contador<=9'b00000000;
    else if (contador < 351)
        contador <= contador + 1;
    else
        contador <= 0;


    always @ (posedge clk)

    if(contador == 'd0)
    begin
        q<=8'b01010011;
    end
    else if(contador == 'd1)
    begin
        q<=8'b01001111;
    end
    else if(contador == 'd2)
    begin
        q<=8'b01011001;
    end
    else if(contador == 'd3)
    begin
        q<=8'b00100000;
    end
    else if(contador == 'd4)
```

Figura 14.4: Circuito parcial elaborado en verilog 1.

```
begin
    q<=8'b01000101;
end
else if(contador == 'd5)
begin
    q<=8'b01001100;
end
else if(contador == 'd6)
begin
    q<=8'b00100000;
end
else if(contador == 'd7)
begin
    q<=8'b01010000;
end
else if(contador == 'd8)
begin
    q<=8'b01010010;
end
else if(contador == 'd9)
begin
    q<=8'b01001001;
end
else if(contador == 'd10)
begin
    q<=8'b01001101;
end
else if(contador == 'd11)
begin
    q<=8'b01000101;
end
else if(contador == 'd12)
begin
    q<=8'b01010010;
end
else if(contador == 'd13)
begin
    q<=8'b00100000;
```

Figura 14.5: Circuito parcial elaborado en verilog 2.

14.3. *Script* de síntesis

```
###### Synthesis Script #######
##### Luis Nájera, 15581 ######

#Set Libraries
lappend search_path /usr/synopsys/PDK/SAED90_EDK/SAED_EDK90nm_REF/references/PARSER/ref/models/
set link_library " * saed90nm_max.db saed90nm_min.db saed90nm_typ.db"
set target_library "saed90nm_typ.db"

#Read Verilog file
read_file -format verilog {/home/administrador/Escritorio/chip/chip.v}

#Set Constraints#
#Remueve los constraints del diseño#
reset_design

#Constraint de Area#
set_max_area 0

#Constraints para tiempo de reloj#
create_clock -name "clk" -period 80 -waveform { 0 40  }  { clk  }
set_clock_uncertainty 1 [get_clocks clk]

#Constraints input/output Delays#
set_input_delay -clock clk -min 2 [remove_from_collection [all_inputs] [get_ports clk]]
set_input_delay -clock clk -max 4 [remove_from_collection [all_inputs] [get_ports clk]]
set_output_delay 6 -min -clock clk [all_outputs]
set_output_delay 8 -max -clock clk [all_outputs]
```

Figura 14.6: *Script* para síntesis 1.

```
#Constraints drive - load#
set_drive 0 [all_inputs]
set_load 5 [all_outputs]

#Condiciones de operación#
set_operating_conditions  TYPICAL

#Design Rule Constraints#
set_max_transition 1.5 {chip}
set_max_capacitance 1.5 {chip}

#Compile/save
compile
write -format ddc -h -o /home/administrador/Escritorio/chip/chip_ddc.ddc
```

Figura 14.7: *Script* para síntesis 2.

14.4. Módulos estímulo para verificación en VCS

```
//Test_Bench
module stimulus;
reg clk;
reg reset;
wire [7:0] q;

chip chip1(q, reset, clk);

initial
clk = 1'b0;
always
#1 clk=~clk;

initial
begin

reset = 1'b1;
#15 reset =1'b0;
#15 reset =1'b1;
#15 reset =1'b0;
#2000 $finish;
end

endmodule
```

Figura 14.8: Módulo estímulo para circuito pre síntesis

```
//Test_Bench_s
`timescale 1ns/1ns
module stimulus (  );

  reg clk,reset;
  wire [7:0] q;

  chip chip1 (.q(q),.reset(reset),.clk(clk));

  initial
  begin
  clk=1'b1;
  reset=1'b1;
  end
  always
  #5 clk=~clk;
  initial
  begin
  reset = 1'b1;
  #15 reset =1'b0;
  #15 reset =1'b1;
  #15 reset =1'b0;
  #2000 $finish;
  end

endmodule
```

Figura 14.9: Módulo estímulo para circuito post síntesis.

CAPÍTULO 15

Glosario

ASIC *Application Specific Integrated Circuit*, el cual se refiere a un circuito integrado desarrollado para cumplir con una tarea específica.

Circuito Integrado Circuitos de pequeñas dimensiones elaborados a partir de una lámina de silicio la cual se somete a un proceso químico con el objetivo de crear transistores tipo *MOSFET* en su interior.

CMOS *Complementary Metal-Oxide-Semiconductor*, consiste en el uso de transistores *p-mos* y *n-mos* en conjunto para la elaboración de circuitos.

Constraints Reglas de diseño que definen la respuesta del circuito.

Design Vision Software diseñado para elaborar síntesis de un circuito elaborado en lenguaje descriptivo de hardware a un *netlist* de compuertas lógicas.

Flujo de Diseño Serie de pasos a seguir para la elaboración de un circuito integrado.

Formality Software que permite determinar la equivalencia entre dos circuitos.

FPGA *Field-Programmable Gate Array*, consiste en un circuito programable el cual permite verificar el comportamiento de un circuito nanométrico elaborado en un lenguaje descriptivo de hardware.

Lenguaje Descriptivo de Hardware Lenguaje de programación que permite definir el comportamiento de un circuito por medio de módulos.

Modulo Se considera el bloque fundamental de los lenguajes descriptivos de hardware, consiste en un segmento del circuito el cual elabora una tarea específica según sus entradas y salidas.

MOSFET *Metal-oxide-semiconductor Field-Effect Transistor*, clasificación de transistores activados por medio de campos eléctricos.

Netlist **de compuertas lógicas** Circuito elaborado a partir de una síntesis, realizado con componentes lógicos referenciados en librerías y a partir de un diseño previo elaborado en lenguaje descriptivo de hardware.

RTL *Register-Transfer Level*, consiste en el diseño modelado a partir de un lenguaje descriptivo

de hardware.

Síntesis Consiste en el mapeo del circuito elaborado en lenguaje descriptivo de hardware a un *netlist* de compuertas lógicas.

Synopsys Empresa desarrolladora de softwares altamente utilizados en el desarrollo de circuitos nanométricos.

VCS Herramienta que permite verificar el funcionamiento de un circuito nanométrico tanto antes como después de la síntesis.

Verilog Consiste en un lenguaje de descripción de hardware que permite modelar sistemas electrónicos.