

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica



**Integración de un microprocesador de aplicación específica en un
flujo de diseño de circuitos integrados digitales.**

Informe de Proyecto de Graduación para optar por el título de
Ingeniero en Electrónica con el grado académico de Licenciatura

Reinaldo Castro González

Versión de 2 de mayo de 2017

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado: Integración de un microprocesador de aplicación específica en un flujo de diseño de circuitos integrados digitales, realizado por el señor Reinaldo Castro González y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador



Ing. Roberto Molina Robles

Profesor lector



Ing. Ronny García Ramírez

Profesor lector



Ing. Alfonso Chacón Rodríguez

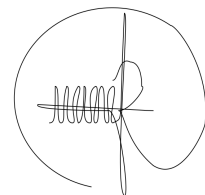
Profesor asesor

Cartago, 2 de Mayo de 2017

[h]

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.



Reinaldo Castro González

Cartago, 2 de mayo de 2017

Céd: 3 0454 0969

Resumen

En este documento se expone una metodología de diseño semipersonalizado de circuitos integrados digitales, mediante el uso de las herramientas Synopsys empleando una tecnología CMOS de 0,13 micrómetros. El flujo de diseño se efectúa mediante una estructura de scripting, principalmente en lenguajes TCL, Bash, y PERL, congruentes con la semántica de las herramientas de Synopsys. Se expone como se desarrolla la jerarquía de archivos y directorios para ubicar los archivos fuente y los productos de la ejecución del flujo.

Se analiza la efectividad del flujo, sometiendo el RTL de un microprocesador de aplicación específica (ASP) en arquitectura RISC-V. Se expone la integración de una celda física para la unidad de punto flotante del ASP y las celdas físicas de las memorias para datos y programa del ASP. El análisis se realiza sobre los reportes generados por la herramienta, y sobre una verificación determinística sobre el RTL y los GLN, mediante simulaciones generadas a partir de bancos de prueba (testbench) derivados de un modelo dorado de alto nivel.

Palabras clave: Back End, Front End, GLN, Integración, RTL, Script, Síntesis, STA, Synopsys

Abstract

This document presents a semipersonalized design methodology of digital integrated circuits, by using of Synopsys tools and a $0,13 \mu m$ CMOS technology. The design flow is made through a scripting structure, mainly in languages like TCL, Bash, and PERL, wich are consistent with the semantics of Synopsys tools. This work discusses how the file and directory hierarchy is developed to locate source files and output products.

The effectiveness of the flow is analyzed by subjecting the RTL of a RISC-V architected Specific Application Microprocessor (ASP). It discusses the integration of a physical cell for the ASP floating-point unit and the physical cells of the ASP data and program memories. The analysis is performed on the reports generated by the tool, and deterministic verification on the RTL and GLN, through simulations generated from test benches, that derives from a golden reference model implementet in a high level programming language.

Keywords: Back End, Front End, GLN, Integración, RTL, Script, Síntesis, STA, Synopsys

a mis queridos padres

Agradecimientos

A mis padres que han sido el pilar fundamental en mi vida, ustedes son los responsables de que sea la persona que soy. Gracias por sus sabios consejos y apoyo incondicional.

A mi hermano Rolando, quien ha sido un gran apoyo en todos mis proyectos, siempre un ejemplo para mí, y mi modelo a seguir mientras crecía.

A mi primo Miguel por ser siempre un amparo para continuar con mis proyectos, gracias por ser otro hermano mayor para mí y brindarme tu apoyo en todas mis metas.

Al Ing. Dr. Alfonso Chacón Rodríguez por toda la colaboración que me brindó, no sólo para que este proyecto se consolidase, sino por toda su ayuda para que pudiese continuar estudiando en esta institución. Gracias por sus consejos, sus duras lecciones y amistad.

Finalmente a los Ingenieros: Dr. Juan Agustín Rodríguez, quien sentó las bases para poder llevar a cabo este proyecto, al M.Sc. Carlos Salazar y M.Sc. Ronny Gracia, quienes colaboraron en la supervisión para que este proyecto fuese exitoso.

Reinaldo Castro González

Cartago, 2 de mayo de 2017

Índice general

Índice de figuras	iii
Índice de tablas	vii
1 Introducción	1
1.1 Entorno de la tesis	1
1.2 Descripción del problema y justificación	4
1.3 Síntesis del problema	4
1.4 Enfoque de la solución	4
1.4.1 Generalidades de la solución	4
1.4.2 Síntesis de la solución	5
1.5 Trabajos anteriores	5
1.6 Objetivos	7
1.6.1 Meta	7
1.6.2 Objetivo general	7
1.6.3 Objetivos específicos	7
2 Marco teórico	9
2.1 Tipos de diseño circuitos integrados digitales	9
2.1.1 ICs totalmente personalizados	9
2.1.2 ICs programables	10
2.1.3 ICs semipersonalizados	11
2.2 Flujo de diseño digital para el diseño de circuitos integrados digitales	12
2.2.1 Generalidades del flujo de diseño digital	13
2.2.2 Flujo de diseño digital en las herramientas de Synopsys	16
2.2.3 Front End	16
2.2.4 Back End	18
3 Flujo de Diseño Digital	21
3.1 Estructura de directorios y archivos	22
3.2 Scripts para diseño de front end	27
3.2.1 Script de síntesis lógica	27
3.2.2 Simulación Lógica: evaluación pre y post síntesis lógica	37
3.3 Scripts de diseño back end	39
3.3.1 Script de implementación o síntesis física	39

3.3.2	Script de simulación física: evaluación post implementación	48
3.3.3	Script de análisis de temporizado estático STA	48
3.4	Script de síntesis de IP Cores y bibliotecas autónomas	52
3.5	Análisis de resultados: Scripting	54
4	Síntesis lógica de un microprocesador RISC-V	57
4.1	Estrategia de síntesis y validación	59
4.1.1	Restricciones del diseño	59
4.1.2	Comprobación de la síntesis	60
4.2	Análisis de resultado: síntesis lógica del ASP	61
4.2.1	Evaluación pre síntesis	61
4.2.2	Evaluación post síntesis	63
5	Implementación física de la FPU del ASP y bloques de memoria	69
5.0.1	Análisis de los resultados obtenidos en la implementación de la FPU .	70
5.0.2	Reportes del análisis de temporización estática (STA)	84
5.1	Integración de los <i>IP Cores</i> de memorias SRAM	88
6	Conclusiones y Recomendaciones	91
6.1	Conclusiones	91
6.2	Recomendaciones	91
	Bibliografía	93
A	Hoja de Información del Proyecto	97

Índice de figuras

1.1	Procesamiento realizado por cada nodo miembro de la RIS. Figura de autoría propia, basada en la expuesta en [8].	2
1.2	Diagrama de bloques del Sistema de Reconocimiento de Patrones Acústicos. Figura de autoría propia, basada en la expuesta en [8].	3
2.1	Tipos de Diseño de Circuitos Integrados. Figura de autoría propia.	10
2.2	Diagrama de Gajski. Espiral que ilustra el flujo digital de diseño de circuitos integrados y cuyas aristas establecen los ámbitos por los que debe atravesar el diseño. Figura de autoría propia basada en lo expuesto en la sección 1.6 de [29].	13
2.3	Esquema representativo e ilustrativo del flujo de diseño de circuitos integrados digitales. Figura de autoría propia.	14
2.4	Esquema ilustrativo del flujo de diseño de circuitos integrados digitales, en la etapa inicial de síntesis lógica (conocida como etapa de front end). Imagen de autoría propia.	16
2.5	Esquema ilustrativo del flujo de diseño de circuitos integrados digitales, enfocado a las herramientas de síntesis física, en esencia las herramientas de <i>Back End</i> . Figura de autoría propia.	18
3.1	Esquema de la jerarquía de directorios y archivos para implementar el flujo de diseño digital. El archivo .gitignore es un archivo oculto, que contiene información para que la herramienta de control de versiones usada en el DCILab opere de forma personalizada. Este archivo carece de relevancia para la estructura del flujo; sin embargo, permite mantener un repositorio más ordenado. Figura de autoría propia.	23
3.2	Esquema completo de la jerarquía de directorios y archivos que implementan el flujo de diseño digital. El bloque TECH hace referencia a la biblioteca de la tecnología. No es una buena idea agregar este directorio en la misma ubicación que el proyecto ya que es un directorio de tamaño considerable y copiarlo en cada proyecto representa un mal uso de los recursos. Figura de autoría propia.	24
3.3	Esquema ilustrativo de la relación entre los scripts de bash que inicializan las herramientas EDA y crean las variables que contienen los punteros a los archivos y la biblioteca de la tecnología de integración. Figura de autoría propia.	25

3.4	Diagrama de flujo del scripting que ejecuta la síntesis lógica. Figura de autoría propia.	28
3.5	Algoritmo recursivo de lectura y análisis de los módulos <i>HDL</i> del diseño. Figura de autoría propia.	30
3.6	Ilustración del proceso de compilación de front end. En figura <i>a</i>) se aprecia la estructura jerárquica de códigos en HDL, y en la figura <i>b</i>) la jerarquía abstraída, implementada mediante distintos niveles de elementos digitales, pasando desde funciones complejas, módulos digitales avanzados, hasta compuertas lógicas fundamentales. Aunque no siempre se llega hasta estas últimas. Figura de autoría propia.	35
3.7	Estructura del script de simulación por comportamiento. El archivo <i>source_list</i> apunta hacia el RTL, que se ubica en algún subdirectorio de <i>source</i> . El script de simulación llama al archivo <i>source_list</i> al invocar al simulador VCS. Figura de autoría propia.	38
3.8	Estructura del script de simulación post síntesis. El archivo <i>precompile</i> , apunta a la biblioteca de simulación y al netlist. El archivo <i>source_list</i> apunta hacia el <i>netlist</i> , el código de estímulo, y la biblioteca de simulación. El <i>script</i> usado invoca al simulador, luego precompila y después llama al archivo <i>source_list</i> para realizar la simulación. Figura de autoría propia.	39
3.9	Primer propuesta de distribución de potencia para el trazado de la unidad aritmética de punto flotante discutida en el capítulo 4.	44
3.10	Diagrama de flujo que expone el proceso genérico de la implementación física en la herramienta <i>IC Compiler</i> . <i>Figura de autoría propia</i>	49
3.11	Estructura del script de bash para la simulación post implementación. Figura de autoría propia.	50
3.12	Diagrama de flujo del análisis de temporizado estático (STA). Figura de autoría propia.	50
3.13	Diagrama de flujo del script para un análisis básico de STA en la herramienta PrimeTime de Synopsys. Figura de autoría propia.	51
3.14	a) Diagrama de flujo de la síntesis de los <i>IP Cores</i> de memorias SRAM y generación de las bibliotecas autónomas. b) Estructura de directorios y archivos que intervienen en la síntesis de los IPcores de las memorias, herramientas involucradas y los archivos generados. Figuras de autoría propia.	54
4.1	Diagrama funcional del microprocesador ASP. RISC-V. Imágen tomada del trabajo [8], con la autorización del M.Sc Salazar.	58
4.2	Resultado de la simulación por comportamiento del ASP con la FPU diseñada en [24]. Según lo estipulado por el M.Sc Salazar en [8]. Esta es la referencia dorada para este trabajo.	61
4.3	Verificación del funcionamiento del ASP al cambiar la FPU por la versión mejorada de [25, 35]. El resultado final del cálculo de probabilidad es erróneo.	62

4.4	Resultado de simulación post síntesis lógica del ASP, con la FPU de [25, 35]. Nótese la existencia de señales indefinidas. Un análisis más detallado mostrará violaciones en la restricción de tiempo de mantenimiento en el resultado de la síntesis lógica.	62
4.5	Detalle de los resultados de la verificación post-síntesis, proporcionados por VCS. Nótese la violación de hold. Esto implica que hay que corregir el problema, ya sea desde DesignCompiler, o modificando el RTL si el DC no logra su cometido.	66
5.1	Foto captura del layout de la celda física generada para la unidad de punto flotante del ASP	71
5.2	Gráfico de los resultados de las distintas simulaciones asociadas con el módulo de suma y resta de la FPU. Observar que hay un desfase entre los datos, hecho a drede para apreciar mejor la dinámica de las distintas gráficas.	86
5.3	Gráfico de los resultados de las distintos simulaciones asociadas con el módulo de multiplicación con signo de la FPU Observar que hay un desfase entre los datos, hecho a drede para apreciar mejor la dinámica de las distintas gráficas.	86
5.4	Gráfica del porcentaje de desviación, de las simulaciones post síntesis de la unidad de suma y resta de la FPU, respecto a simulación por comportamiento. Se que la simulación post síntesis lógica, presenta algunas incongruencias para ciertos datos, mientras la síntesis física es congruente.	87
5.5	Gráfica que muestra el porcentaje de desviación, de las simulaciones post síntesis de la unidad de multiplicación de la FPU. Observe que ambas líneas son perfectamente horizontales, lo que indica que la simulaciones post síntesis son consistentes con la de comportamiento	88
5.6	Foto captura del layout resultante de la impementación del <i>IP Core</i> de una memoria SRAM de 4kx32.	89
5.7	Simulación post implementación física del <i>IP Core</i> para la memoria de datos (4kx32) con el modelo de retardos mínimos	90
5.8	Simulación post implementación física del <i>IP Core</i> para la memoria de datos (4kx32) con el modelo de retardos típicos	90

Índice de tablas

5.1	Resumen del reporte de potencia de la FPU post síntesis lógica.	71
5.2	Resumen del reporte de potencia de la FPU post síntesis física.	72

Índice de listados

3.1	Directivas para crear las variables de entorno utilizadas por los scripts de las herramientas e invocar el scrip de las herramientas de Synopsys.	24
4.1	Reporte de temporizado del microprocesador ASP post síntesis lógica.	63
4.2	Reporte de área del microprocesador ASP post síntesis lógica.	67
5.1	Reporte de área de la unidad de punto flotante de ASP post síntesis lógica.	72
5.2	Reporte de área de la unidad de punto flotante de ASP post síntesis lógica.	73
5.3	Reporte de temporizado del microprocesador ASP post síntesis lógica.	73
5.4	Reporte de temporizado de la FPU post síntesis lógica.	76
5.5	Reporte de temporizado del microprocesador ASP post síntesis lógica.	79
5.6	Reporte STA de la FPU. Anotaciones sobre el análisis entre registros y puer- tos.	82
5.7	Reporte STA de la FPU. Anotaciones sobre el análisis entre registros y puer- tos.	84

Capítulo 1

Introducción

1.1 Entorno de la tesis

Costa Rica es uno de los países que se encuentran a la vanguardia en materia de conservación ambiental, esto debido al sistema de áreas protegidas, que cubre cerca del 25% del territorio nacional cuenta con alguna categoría de protección. Las políticas nacionales de protección no sólo abogan por preservar la flora y fauna local sino también por mantener un desarrollo sostenible garantizando la perpetuidad de los recursos naturales. [1]

Los recursos naturales son, innegablemente, el medio fundamental para el crecimiento integral de las sociedades modernas. Estos se encuentran implícitos en todas las actividades humanas y son indispensables para sostener el estilo de vida del ciudadano moderno, desde su nutrición, materia prima para la manufactura de la tecnología que emplea, los servicios que recibe, entre otros. [2]

Por otra parte la riqueza natural representa una gran fuente de ingresos económicos para el país, debido al turismo. Según el informe del ICT, durante el primer trimestre del 2015 al país arribaron por todas las vías de ingreso un total de 811,379 personas, mientras que para el primer trimestre del 2016 arribaron un total de 919,624 lo que representa un incremento del 13.3%. [3]

No obstante al marco jurídico en materia ambiental y los programas de protección ambiental hacia la flora y fauna, Costa Rica tiene serios problemas de caza furtiva y tala indiscriminada, pues los programas de conservación y la legislación actual son insuficientes para velar por la correcta protección de los recursos naturales (i.e. flora, fauna, minerales, etc), en mayor medida debido a la carencia de los fondos y personal necesarios para ejercer con más rigurosidad las políticas de conservación y penalización. De forma ilustrativa según el Sistema Nacional de Áreas de Conservación, en el 2014 eran necesarios, aproximadamente, 300 funcionarios dedicados al control y protección de las áreas protegidas (i.e. guardabosques) y el Estado era incapaz de hacer las contrataciones por falta de presupuesto. [1]

Dado el dilema expuesto, estatalmente, se han puesto en marcha diversas estrategias para enfrentar la problemática; sin embargo, aumentar el recurso humano no es la opción más rentable para el Estado, siendo probable que una densidad mayor de guardabosques tampoco sea la solución. Partiendo de estas premisas se ha optado por un cambio de visión en camino a resolver el problema. Se considera que la posible solución reside en el uso de tecnología para facilitar la labor de los guardabosques.

Esta propuesta de solución tecnológica consiste en la implementación de una Red Inalámbrica de Sensores (*RIS*) para vigilar de forma remota las zonas protegidas. Una *RIS* no es otra cosa que una serie de dispositivos autónomos distribuidos en un área de interés, utilizando elementos de instrumentación (i.e. sensores) para monitorear condiciones físicas o ambientales. Cada miembro de la red corresponde a un nodo [4]. Se considera que si la *RIS* no es la solución óptima, sí es una alternativa muy eficiente pues una *RIS* es capaz de medir parámetros, almacenarlos, procesarlos y enviarlos entre sí, lo cual permite ejecutar muchas de las labores realizables por un guardabosques. Como consideración especial, debe tenerse en cuenta que en un bosque no hay un acceso fácil a fuentes de alimentación eléctrica, por lo que es imperativo que la *RIS* tenga un bajo consumo energético.

En respuesta a esta restricción, se realizó el proyecto: "Sonidos Ilegales" [5] de la Escuela de Ingeniería Electrónica del Tecnológico de Costa Rica (*TEC*), el cual refiere a la implementación de un sistema de reconocimiento de sonidos de motosierras, disparos de armas de fuego, como indicadores indirectos de actividades de tala y caza ilegales en zonas protegidas. Este proyecto consistió en el diseño de un sistema de hardware *SoC*¹ de bajo consumo energético el cual llevaría a cabo las operaciones de procesamientos mencionas de la *RIS*. [6, 7]

El funcionamiento de la *RIS* propuesta consistió en la obtención de muestras de señales acústicas del medio, que en este caso se trata de un bosque. Seguidamente, los datos eran procesados aplicando reconocimiento de patrones. En caso de encontrar un estado de alarma: disparo o motosierra, se activaría el bloque de transmisión de información donde posteriormente la persona a cargo de la protección de la zona (el guardabosques), debía recibir una notificación donde se le indicará el tipo de suceso y en cual nodo de la red se hizo de detección. En la figura 1.1 se muestra un diagrama de bloques de un nodo de la *RIS* que se propuso, la cual permite, a grosso modo, entender el funcionamiento de la misma.

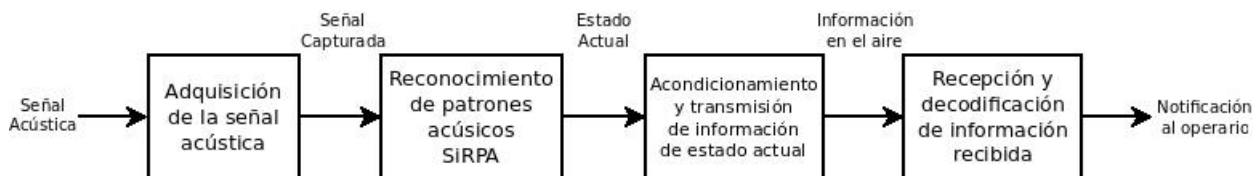


Figura 1.1: Procesamiento realizado por cada nodo miembro de la *RIS*. Figura de autoría propia, basada en la expuesta en [8].

¹SoC: Single on Chip. Sistema integrado en un chip

En cuanto al procesamiento, el bloque llamado *SiRPA*², sigue una estructura estándar de un sistema de reconocimiento de patrones, compuesto de un bloque de preprocesado, una etapa de identificación/extracción de características fundamentales y un bloque de clasificación.

El preproceso consiste en acondicionar la señal analógica, que significa normalizar y digitalizar la señal acústica. El bloque de identificación está compuesto por una batería de filtros, un estimador de energía, un reductor de dimensiones y un árbol clasificador. En el banco de filtros el espectro de la señal acústica se separa en 8 bandas distintas y luego se estima la energía a cada banda.

Mediante una transformación lineal, el reductor de dimensiones proyecta la salida del estimador de energía, que se encuentra en un espacio de ocho dimensiones, a un espacio de tres dimensiones. Según lo predice la *maldición de dimensionalidad*, el conjunto de símbolos necesarios para describir de manera adecuada las observaciones realizadas es tres órdenes de magnitud superior si se emplea un espacio de ocho dimensiones en comparación a uno de tres. [10, 9]

El árbol binario o generador de símbolos, permite ubicar el centroide más cercano a un patrón de entrada de tres dimensiones, según la distancia L1 o Manhattan. Consecuentemente, los símbolos discretos se generan asociándolos, a las trayectorias continuas en el espacio del vector tridimensional de entrada, que describe la señal acústica en un determinado instante. El árbol binario genera un alfabeto de 32 símbolos discretos. [11, 12, 13, 14]

La etapa de clasificación es la encargada de calcular la probabilidad de que un conjunto de símbolos de entrada, formen parte de una señal acústica, de acuerdo con las categorías: bosque, motosierra o disparo. En este bloque, se utiliza un clasificador basado en la técnica de modelos ocultos de Markov (HMM). En la figura 1.2 se muestra un diagrama de bloques ilustrando como funciona internamente el SiRPA.

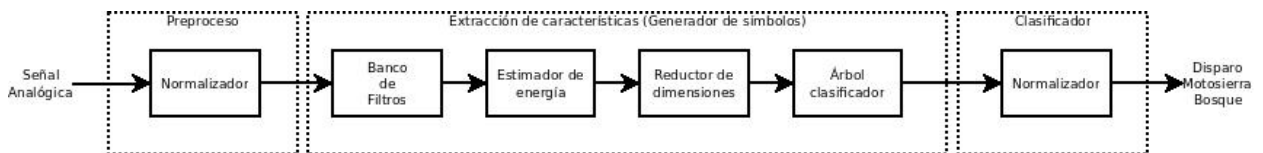


Figura 1.2: Diagrama de bloques del Sistema de Reconocimiento de Patrones Acústicos. Figura de autoría propia, basada en la expuesta en [8].

²SiRPA, es el acrónimo en español para Sistema de Reconocimiento de Patrones Acústicos

1.2 Descripción del problema y justificación

Luego de muchas etapas iterativas en el diseño del *SiRPA*, el proyecto se encuentra codificado en un lenguaje de descripción de hardware (*HDL*). Se han efectuado pruebas usando plataformas de sistemas embebidos como *Beagle Boards* y *textitFPGAs* para la validación del diseño, se han publicado algunos papers al respecto, y se ha demostrado la viabilidad del proyecto, por lo que compete continuar la labor de implementar la codificación *HDL* a su etapa final, la cual consiste en la síntesis de los módulos descritos en *HDL* sobre un chip de silicio.

Durante el 2014 el *DCILab* contó con el apoyo del ingeniero argentino Dr. Juan Agustín Rodríguez, quien desarrolló un flujo de diseño digital sobre las herramientas de software Synopsys que implementan los módulos en *HDL* del *SiRPA* al silicio para el circuito integrado. El trabajo del Dr. Rodríguez quedó incompleto y el *SiRPA* no ha podido ser fabricado. Es por ello que el *DCILab* requiere terminar el trabajo empezado por el Dr. Rodríguez e integrar la parte final del *SiRPA*, que consiste en un microprocesador desarrollado por el Ing. M.Sc. Carlos Salazar García, para así dejar todo el proyecto *SiRPA* depurado y listo para su fabricación. Además el *DCILab* está interesado en que dicho proceso de síntesis quede documentado y se establezca una estructura de diseño digital genérica, capaz de ser adaptada para futuros proyectos.

Es posible fabricar estos prototipos vía *MOSIS*³, un consorcio que permite a universidades y pequeños fabricantes acceder a fábricas con costos manejables. [15]

1.3 Síntesis del problema

El proyecto *SiRPA* se encuentra inconcluso y el *DCILab* requiere de una persona con conocimientos en microelectrónica capaz de someter los componentes del *SiRPA* a un flujo de diseño *ASIC*, y fabricar un prototipo *SoC*.

1.4 Enfoque de la solución

1.4.1 Generalidades de la solución

Dado que se requiere diseñar un circuito integrado digital que parte de una codificación *HDL*, es necesario desarrollar una jerarquía de diseño digital que permita realizar el layout de los módulos descritos en *HDL*, por lo que se necesita de una herramienta de software especializada para tal fin. *Design-Compiler* e *IC-Compiler* son herramientas interactivas de

²Una FPGA (del inglés Field Programmable Gate Array) es un dispositivo electrónico que contiene bloques de lógica y electrónica digital que puede ser reconfigurada “in situ” mediante un lenguaje de descripción de hardware.

³MOSIS (Metal Oxide Semiconductor Implementation Service)

Synopsys que le permiten al usuario controlar el desarrollo del circuito integrado sobre el silicio, definiendo las características físicas como: conexión eléctrica, distribución espacial y física, limitaciones espaciales y sus implicaciones electromagnéticas para que su posterior fabricación e implementación física sean exitosas.

El proceso de diseño de un circuito integrado es lento, tedioso, y tiene un componente iterativo, lo que implica tiempo para que un ingeniero se familiarice con el flujo de diseño. Es por ello que aparte de crear el flujo de diseño de un chip sobre la herramientas, es conveniente establecer para futuros proyectos, una estructura de archivos y *scripts*, propios de las herramientas de Synopsys, para así facilitar el diseño y rediseño de proyectos futuros del DCILab. Como comprobación de la efectividad del flujo de diseño digital anteriormente mencionado, deberán someterse los componentes del ASP mediante las herramientas de Synopsys, para su posible fabricación a futuro, y establecer una documentación oportuna para agilizar el trabajo, de los colaboradores del DCILab en cuanto a síntesis sobre silicio respecta.

1.4.2 Síntesis de la solución

Establecido el panorama anterior, la solución consiste en desarrollar una jerarquía de flujo de diseño digital en las herramientas de Synopsys, que permita someter los componentes del SiRPA (específicamente el microprocesador ASP) al proceso de síntesis al silicio, según las reglas de diseño del proceso de fabricación de IBM: 8RF de 0,13 micrómetros (IBM013), que es la tecnología a la cual tiene acceso el DCILab, y así contar con el proyecto SiRPA completo hasta la última etapa previa a su fabricación que podría hacerse vía MOSIS en un futuro. Cabe destacar que la índole del proyecto y las restricciones del laboratorio en términos de licencias de software solamente permiten una opción de EDA ⁴, la cual es Synopsys.

1.5 Trabajos anteriores

SiRPA es un proyecto de investigación que ha sido explorado y analizado desde diversas perspectivas de solución cada investigador ha dado forma a cada etapa que componen el sistema. Tanto a nivel de software como a nivel de hardware, los aportes incluyen desde propuestas de diseño, implementación, mejoras o rediseño de etapas que han presentado problemas. Entre estos trabajos se encuentran los siguientes:

- En [16], se efectuó la primera prueba de concepto para evaluar la viabilidad de implementar el SiRPA. Se empleó el software MATLAB, para generar un modelo de alto nivel cuyo objetivo fue el análisis de las señales acústicas y se desarrolló una etapa de identificación de características utilizando la teoría de onditas (wavelets en inglés). Este trabajo constituye la primer referencia del sistema. La estrategia de extracción de características mediante onditas fue descartada posteriormente

⁴EDA: Electronic Design Automation. Software usado en la síntesis de circuitos integrados

- En [17], se implementó el SiRPA en una FPGA utilizando el lenguaje de descripción de hardware VHDL. Aquí se estableció que el uso de un banco de filtros utilizando una misma unidad de filtro para cada una de las ocho bandas, como una metodología de clasificación idónea para los requerimientos de hardware del SiRPA. Es en este trabajo donde se intenta implementar por primera vez el algoritmo de HMM en hardware mediante un MAP (del inglés Matrix and Arrays Processor), el cual consiste en una estructura digital cuya arquitectura se optimiza de acuerdo con la ejecución de operaciones en punto flotante en 32 bits de forma combinacional.
- En [18], se desarrolló un módulo reductor de dimensiones mediante la estrategia de transformación lineal, fundamentada en el algoritmo de discriminantes de Fisher. Este módulo permitió transformar un espacio de 8 dimensiones en un espacio de 3 dimensiones, que implica mayor facilidad en la clasificación de señales acústicas y ahorro energético, al disminuir la densidad de hardware.
- En [13], se desarrolló la herramienta HMMSoft que consiste en un software de alto nivel (lenguaje C/C++) con interfaz gráfica, capaz de realizar el entrenamiento del algoritmo de HMM utilizado por el clasificador. Esta aplicación permite identificar características de patrones acústicos lo cual contribuye a validar el reconocimiento de un audio en particular. Esta herramienta representa un hito considerable ya que mediante ella, es posible establecer un marco de referencia para la verificación funcional del sistema.
- En el trabajo de [9], se realizó la síntesis de la sección de extracción de los patrones acústicos y entrenamiento a partir de conjuntos de datos de audio similares a los que se esperara detectar. El proceso de extracción mencionado fue integrado a la aplicación HMMSoft [13]. Consecuentemente el proyecto SiRPA fue modelado en el lenguaje C, e implementado en un sistema embebido (BeagleBoard xM) [19] y usando la herramienta HMMSoft además del modelo en C del SiRPA se desarrolló el primer modelo de verificación para SiRPA.
- En [20], se implementaron los módulos de hardware del reductor de dimensiones, generador de símbolos y una unidad de HMM en Verilog. Partiendo de la estructura de la unidad HMM desarrollada por [9] para el sistema embebido, en [20] se propone implementar la unidad mediante una máquina microprogramada. Posteriormente al realizar pruebas sobre una FPGA, se observó que los resultados divergían de lo esperado según el modelo de [9] y por tanto clasificar los patrones acústicos correctamente es imposible. Según expone [8] el hardware presentó problemas de cálculo ya que los resultados tiendían a cero rápidamente debido a que en [20], no se consideraron los problemas de escalamiento que se describen ampliamente en [21].
- En [22], se expuso la verificación funcional de los módulos descritos en hardware que conforman el SiRPA.

- En [23], se tomó como punto de partida el trabajo de [17] y se rediseñó la unidad contemplando efectos de submuestreo. De acuerdo con un análisis del bloque de reducción de dimensiones elaborado en [20] se estableció la necesidad de aumentar el formato de palabra de 16 a 24 bits. Por lo tanto se integraron además todos los bloques para formar la etapa de identificación o extracción de características. Finalmente, se evaluó la funcionalidad de esta etapa, tomando como referencia dorada el trabajo realizado en [9].
- En [8] se expone la necesidad de implementar la unidad de clasificación *HMM* mediante un microprocesador de aplicación específica ya que durante la experimentación en [23] se determinó que la unidad *HMM* implementada en [20] no era correcta.
- En cooperación con el trabajo de [8], se encuentra ella trabajo [24] donde se expone el proceso de diseño e implementación en un *FPGA* de la unidad de punto flotante (*FPU*) que usará el microprocesador de [8]. Posteriormente [25] expone la implementación de algunas mejoras para la *FPU* de [24].

1.6 Objetivos

1.6.1 Meta

Desarrollar un sistema de detección de disparos de armas de fuego, motosierras y otras actividades ilegales en un bosque, que esté implementado en un circuito integrado de bajo consumo energético.

1.6.2 Objetivo general

Implementar un microprocesador de aplicación específica en una tecnología CMOS de 0,13 micrómetros para posteriormente ser integrado al proyecto SiRPA.

1.6.3 Objetivos específicos

- Diseñar una jerarquía de scripting⁵ que implemente el flujo de síntesis y simulaciones "Post Colocado y Enrutamiento" (*Place&Route*), para optimizar el tiempo del proceso de diseño para futuros proyectos.
- Sintetizar lógicamente de manera correcta los módulos RTL del Microprocesador de Aplicación Específica (ASP) sobre celdas estándar de una tecnología CMOS 0,13.

⁵En informática un script es un archivo que contiene un conjunto de órdenes e instrucciones, las cuales ejecutan una función, ya sea en un lenguaje de programación o una herramienta de software. Se invita al lector a consultar un diccionario de términos de computación

- Sintetizar físicamente de manera correcta la unidad aritmética de punto flotante del microprocesador y las memorias de datos y programa del microprocesador ASP sobre la tecnología CMOS 0,13

Capítulo 2

Marco teórico

Diseño de ASICs

Este trabajo no pretende exponer el proceso de fabricación de un circuito integrado. En las secciones siguientes se expondrá brevemente, la metodología usada para que a futuro se pueda llevar a cabo la implementación de un sistema sobre un dado de silicio ¹ (silicon dice), usando las herramientas de Synopsys.

En primera instancia producir un sistema en un circuito integrado representa una mejor protección para los derechos intelectuales, pues su contenido difícilmente es reproducible. Un circuito integrado presenta enormes facilidades de portabilidad debido a su tamaño y su consumo energético. Lo cual es el principal objetivo del proyecto: "Sonidos Ilegales".

2.1 Tipos de diseño circuitos integrados digitales

La clasificación del diseño de circuitos integrados se puede apreciar en el esquema de la figura 2.1.

Como puede observarse el diseño de circuitos integrados (de aquí en adelante ICs²) se puede clasificar a grosso modo en tres tipos principales:

2.1.1 ICs totalmente personalizados

En esta metodología de trabajo los diseñadores generan todas las celdas lógicas de acuerdo con las necesidades tiene el diseño que se pretende realizar, lo que contempla la concepción funcional, lógica y física de cada celda. También se desarrollan las máscaras necesarias para la fabricación del chip sobre la oblea de silicio.

¹Entendiendo por dado el espacio que el sistema propuesto ocupará sobre la oblea de silicio, al final del proceso de fabricación

²ICs: acrónimo en inglés para circuitos integrados

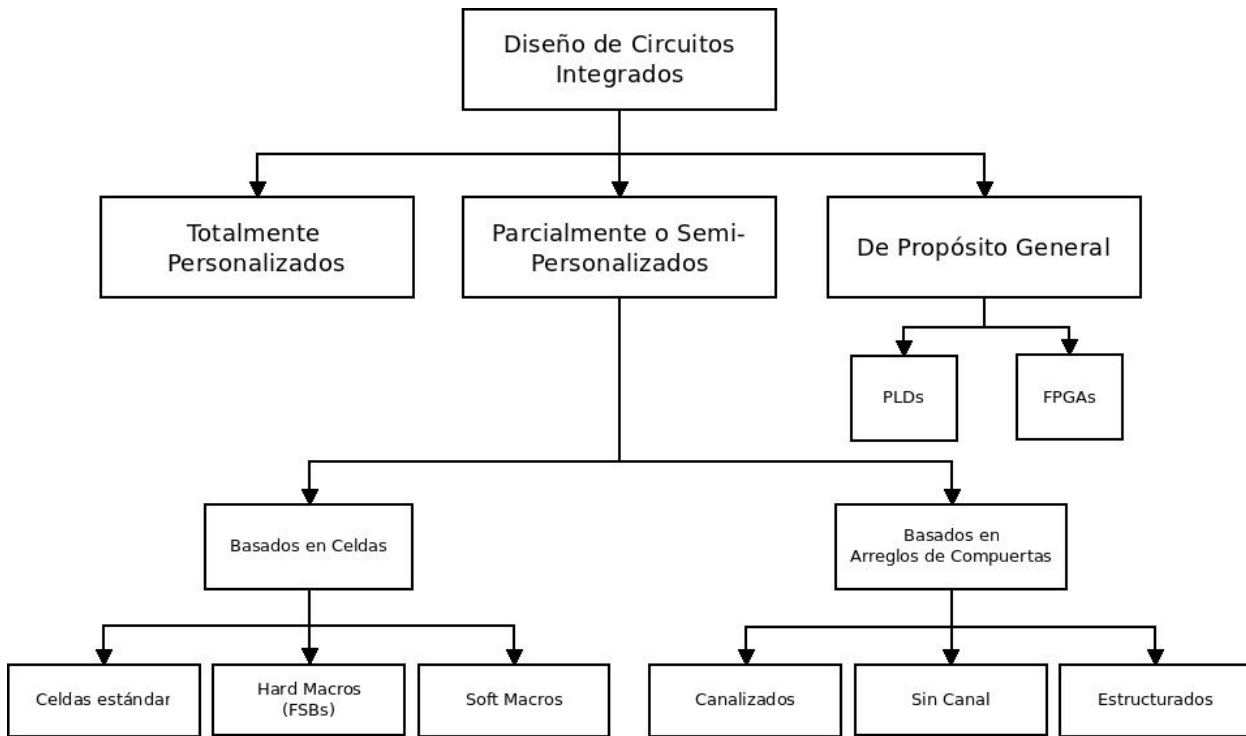


Figura 2.1: Tipos de Diseño de Circuitos Integrados. Figura de autoría propia.

Empresas como Intel son un buen ejemplo de industrias dedicadas al desarrollo de circuitos integrados totalmente personalizados. Aquí los ingenieros invierten enormes cantidades de tiempo maximizando el aprovechamiento de cada micrómetro cuadrado disponible en el dado, lo que les permite incluir circuitos analógicos, optimizar celdas de memoria e incluso contemplar y mejorar la eficiencia mecánica de las estructuras de conexión y ubicación de los componentes del chip.

Esta es la metodología más costosa de todas, no obstante, es la respuesta adecuada si el diseñador necesita de unidades (celdas) más eficientes en términos energéticos y velocidad, de las que es capaz de encontrar en una biblioteca de celdas existentes. Es comúnmente utilizada por empresas dedicadas a la innovación y el desarrollo tecnológico.

2.1.2 ICs programables

Estos consisten en dispositivos lógicos programables (PLDs), que son circuitos integrados con configuraciones estándar predeterminadas. Algunos ASICs para permiten la personalización de algunas funciones puntuales, por lo que aunque sean estructuras prediseñadas y flexibles, son consideradas como una rama del diseño de ICs.

2.1.3 ICs semipersonalizados

Son aquellos ICs para los cuales existe una biblioteca de celdas estándar y posiblemente todas las máscaras de diseño están disponibles. El uso de bibliotecas de celdas prediseñadas hace el diseño más fácil. Esta metodología de diseño se subdivide en dos partes:

Basado en arreglos de compuertas

En este tipo de diseño los transistores se encuentran predefinidos en un patrón dado sobre en la oblea de silicio. Este patrón se conoce como arreglo base y a los elementos más pequeños se los conoce como celdas base o celdas primitivas.

En esta técnica el diseñador únicamente define la interconexión de las celdas usando para ello máscaras personalizadas. Así el diseñador escoge de entre una basta biblioteca de celdas prediseñadas y precharacterizadas. Las celdas lógicas suelen ser denominadas como macros. La razón de esto se debe a que el trazado de la celda base es el mismo para cada celda lógica y únicamente se personaliza la interconexión y el enrutado dentro de las celdas y con las celdas adyacentes, de manera similar a un macro de software.

Existen subcategorías del uso de esta técnica, pero esa información excede el alcance de esta tesis, por lo que no serán expuestas. Se invita al lector a considerar una bibliografía pertinente [26, 27]

Basado en celdas estándar

Un circuito integrado basado en celdas estándar usa celdas lógicas prediseñadas como compuertas “Y”, “O”, “Flip Flops”, “Multiplexores”, etc.. Estas celdas se conocen como celdas estándar.

Suele usarse el término *CBIC* para esta categoría. El área de ubicación de las celdas estándar en un *CBIC* se construye en hileras, de manera similar a una pared de ladrillos. Las celdas estándar pueden y suelen ser utilizadas en conjunto a microcontroladores, o microprocesadores, conocidos como “Mega celdas”. Este término también se emplea para bloques totalmente personalizados, y suelen ser denominados como: “Macros a Nivel de Sistema (SLMs)”, “Mega Funciones”, “Bloques Fijos”, o “Bloques Funcionales Estándar (FSBs)”.

En esta metodología es diseñador define únicamente la colocación y el enrutado (interconexión entre celdas estándar) de las celdas estándar; sin embargo, éstas pueden ser colocadas libremente en el área disponible del dado. Esto implica que las máscaras de fabricación en un *CBIC* son únicas para cada cliente.

Usar *CBICs* tiene enormes ventajas ya que permite hacer diseños más flexibles y optimizarlos en términos de aprovechamiento de área, velocidad o consumo energético. Adicionalmente las bibliotecas de celdas estándar suelen ofrecer las mismas celdas estándar prediseñadas para cumplir distintas las expectativas de desempeño, enfocadas a tamaño, velocidad o energía.

Las principales desventajas consisten en que el tiempo necesario para el diseño suele ser elevado, debe comprarse una biblioteca de celdas estándar, y se está sujeto a las restricciones de ésta, y finalmente las máscaras de fabricación serán nuevas con cada nuevo diseño o versión de diseño, lo cual implica tiempo adicional, además de costos adicionales por parte del fabricante.

La información de esta sección se tomó a partir de las fuentes [28, 26, 28]

2.2 Flujo de diseño digital para el diseño de circuitos integrados digitales

Dada la complejidad en el proceso de diseño de circuitos integrados, es imperativo llevar a cabo un diseño estructurado, que use los principios de jerarquización, modularidad, regularidad y localidad para manejar la complejidad del diseño.

El diseño digital VLSI suele ser particionado en cinco niveles de abstracción: diseño de arquitecturas, diseño de microarquitecturas, diseño lógico, diseño de circuitos y diseño físico. Estas etapas son ejecutadas en paralelo, y en muchas ocasiones son dependientes entre sí.

Una manera alternativa de analizar el diseño estructurado es mediante el “diagrama Y” mostrado en la figura 2.2.

El diagrama de Gajski o diagrama Y, permite entender el concepto de abstracción en el diseño digital, pues se parte desde un concepto general de diseño y al interiorizar en el diagrama se desvelan las etapas que llevan hasta la fabricación correcta del circuito integrado.

El proceso de diseño puede verse como la transformación desde un dominio hacia otro manteniendo la equivalencia de los dominios. Las descripciones por comportamiento se transforman en descripciones estructurales y estas a su vez son transformadas en descripciones físicas. Cada transformación es validada, ya sea manualmente o mediante herramientas automatizadas. La especificación jerárquica de cada dominio y sucesivamente detallando sus niveles de abstracción es lo que permite diseñar grandes sistemas.

La razón para describir en detalle los niveles de abstracción y los respectivos dominios es para definir un proceso de diseño en el cual la función final del sistema es capaz de rastrearse hasta la descripción por comportamiento inicial.

El diagrama Y muestra las transformaciones entre cada dominio y las variaciones entre los niveles de abstracción. El flujo de diseño procede desde los anillos exteriores hacia los interiores, profundizando en niveles de abstracción cada vez más complejos, de acuerdo a una jerarquía establecida.

Para más detalles ir a la sección 1.6 de [29].

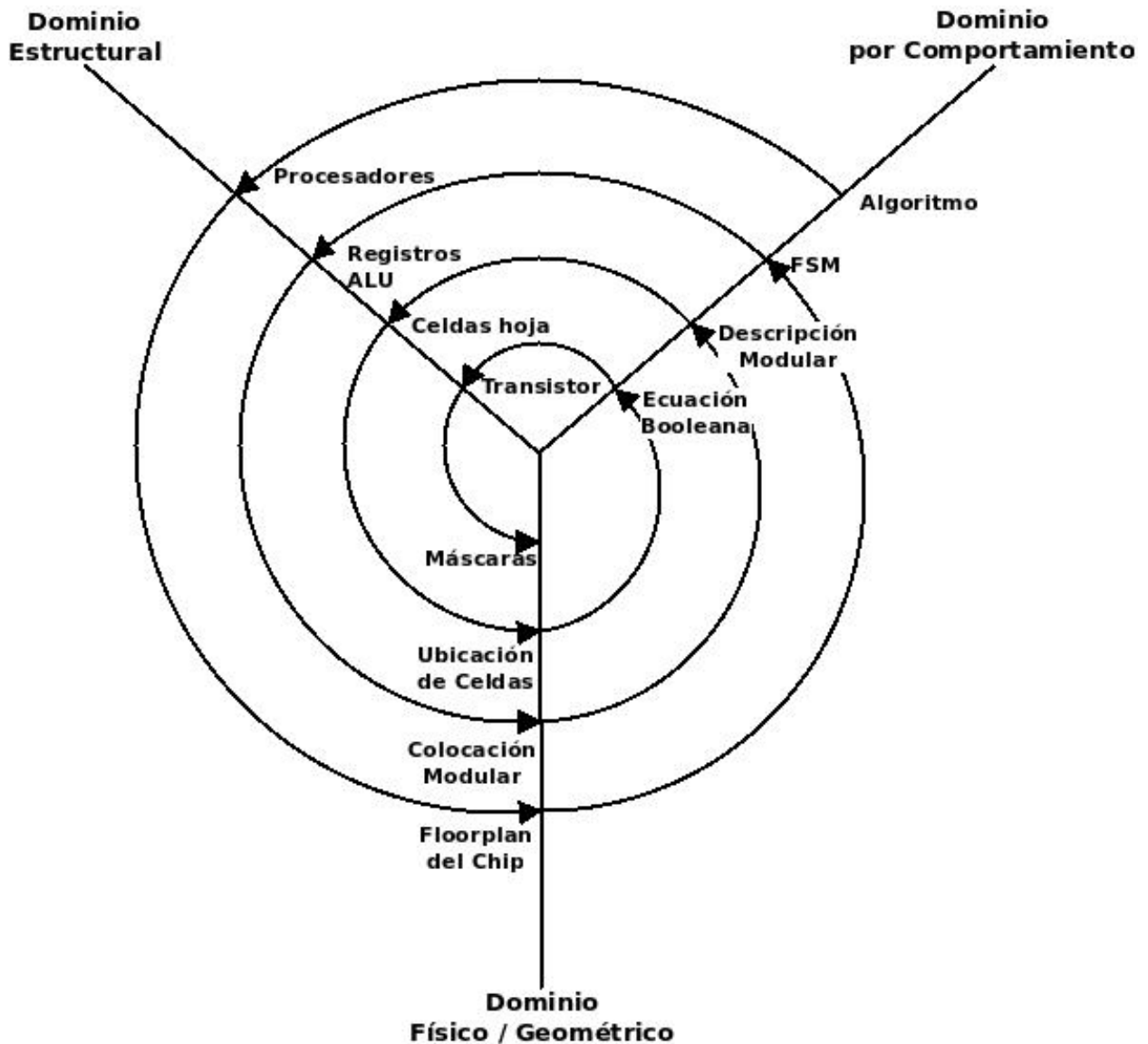


Figura 2.2: Diagrama de Gajski. Espiral que ilustra el flujo digital de diseño de circuitos integrados y cuyas aristas establecen los ámbitos por los que debe atravesar el diseño. Figura de autoría propia basada en lo expuesto en la sección 1.6 de [29].

2.2.1 Generalidades del flujo de diseño digital

Como se mencionó en el apartado anterior, el diseño comienza a dar sus pasos en cuanto es concebida una función o proceso y su comportamiento ha sido modelada en un algoritmo, ya sea secuencial, concurrente o una mezcla de ambos. Este algoritmo se traduce a un lenguaje estandarizado, el cual en el caso de los circuitos integrados digitales corresponde a un *HDL*, que en términos concretos es la descripción digital del circuito integrado o el modelo digital.

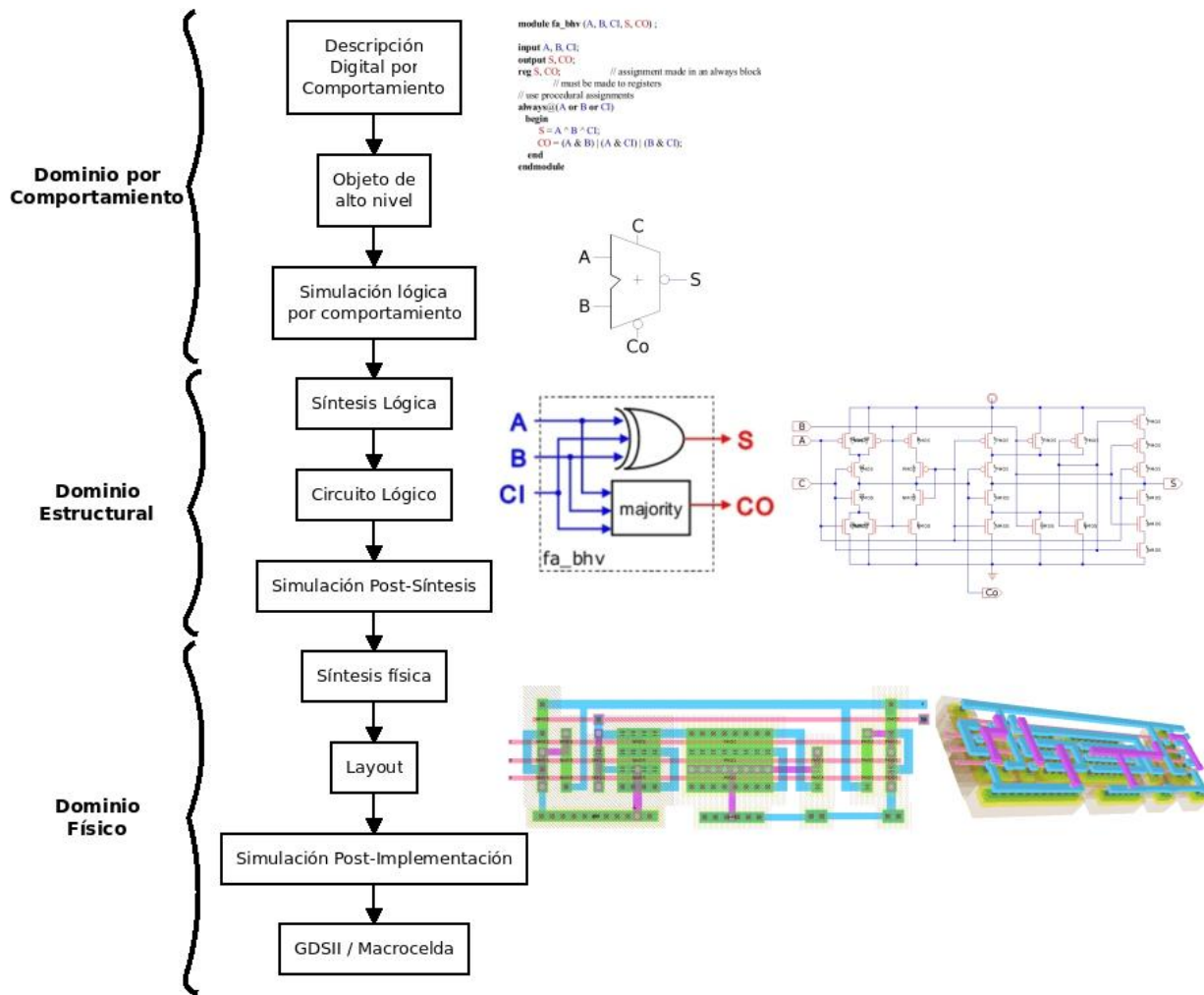


Figura 2.3: Esquema representativo e ilustrativo del flujo de diseño de circuitos integrados digitales. Figura de autoría propia.

Partiendo del modelo digital se puede hacer una verificación de comportamiento, que en términos técnicos se denomina simulación lógica. El modelo digital no es más que un objeto de alto nivel capaz de emular el comportamiento deseado. Su utilidad es poca para el diseñador, ya que no provee datos sobre el desempeño del diseño, únicamente provee información a nivel de comportamiento; sin embargo, es el punto de partida en el flujo de diseño.

Una vez que la simulación lógica es satisfactoria, se procede a usar el modelo por comportamiento y asociar los módulos descritos en él, con unidades lógicas provistas de características de retardos (sincronización de señales), consumo energético, y área, las cuales se encuentran en la biblioteca de la tecnología que se usará en la fabricación. A este proceso se le conoce como síntesis lógica, y genera una base de datos que contiene datos de las celdas estándar y las conexiones entre ellas para ejecutar las funciones abstraídas del modelo de comportamiento.

La base de datos creada en el proceso de síntesis permite generar un nuevo modelo codificado en HDL y una primera aproximación del desempeño de los módulos diseñados, i.e. un modelo con la información sobre los retardos de propagación de las señales en las celdas y entre las mismas, un presupuesto de la energía disipada y el área necesaria para ubicar las celdas usadas. Este nuevo modelo, denominado como modelo post-síntesis nuevamente se simula con el mismo arreglo o banco de pruebas usado en la simulación por comportamiento.

La simulación post-síntesis permite observar el retardo de propagación de las señales, y confirmar si las expectativas de sincronización son alcanzadas. También permite verificar si los datos son generados en los plazos necesarios, y poder garantizar una funcionalidad correcta de los módulos y el diseño en lo concerniente al desempeño de las celdas estándar.

Cuando los resultados obtenidos en la síntesis lógica y la simulación respectiva sean satisfactorios, se toma el modelo post-síntesis y se usa esta información para empezar con el plano de piso (floorplan) del chip. Las celdas estándar son invocadas a un área determinada, y posteriormente se establecen las conexiones que permiten implementar los módulos abstraídos en la primera etapa del diseño (el modelo de comportamiento). Este proceso recibe el nombre de síntesis física o implementación física, en esta tesis se usará la palabra síntesis para referirse a la síntesis lógica, y la palabra implementación será usada para referirse a la síntesis física).

Posteriormente, se genera una nueva base de datos con información más precisa sobre el diseño. Retardos debido a efectos parasíticos, distancia del recorrido de las señales, pérdidas resistivas del alambrado, y otros efectos que serán expuestos más adelante.

Nuevamente se genera una base de datos con la información del colocamiento y el enrutado, que incluye modelos de sincronía y potencia más completos, y otra representación HDL del diseño post implementación. Esta etapa permite obtener una respuesta más precisa de diseño. Al igual que en la etapa anterior, se crean reportes de consumo energético, aprovechamiento de área, calidad de resultados (QOR, que es un compendio de la información más relevante de todos los reportes generados), etc.

Cuando una simulación post-implementación es satisfactoria se procede a dar por concluido el flujo, y a continuación se establecen dos panoramas:

En el primer caso nuestro diseño está completo por lo que se procede a compilar la base de datos en un archivo estándar para condensar la información necesaria para el fabricante. Aquí se valora si el diseño tiene probabilidad alta de ser fabricado con éxito, siendo así la información suministrada será usada para desarrollar las máscaras necesarias para la construcción del chip.

El segundo escenario consiste en que el diseño en el que se ha estado trabajando no es el producto final, sino que forma parte de otro diseño más complejo y grande, el cual aún no haya sido terminado. En ese caso la base de datos servirá para generar una macro celda que será utilizada en diseños posteriores.

En la figura 2.3 se aprecia un resumen gráfico de la explicación anterior.

2.2.2 Flujo de diseño digital en las herramientas de Synopsys

Synopsys es una empresa que brinda herramientas de diseño asistido por computadora (CAD), para la automatización de procesos en el diseño de circuitos integrados. El flujo de diseño digital suele ser similar independientemente del proveedor de las herramientas; sin embargo, esta tesis gira entorno a los softwares de Synopsys, por lo que se expondrá la metodología de diseño usando este proveedor.

2.2.3 Front End

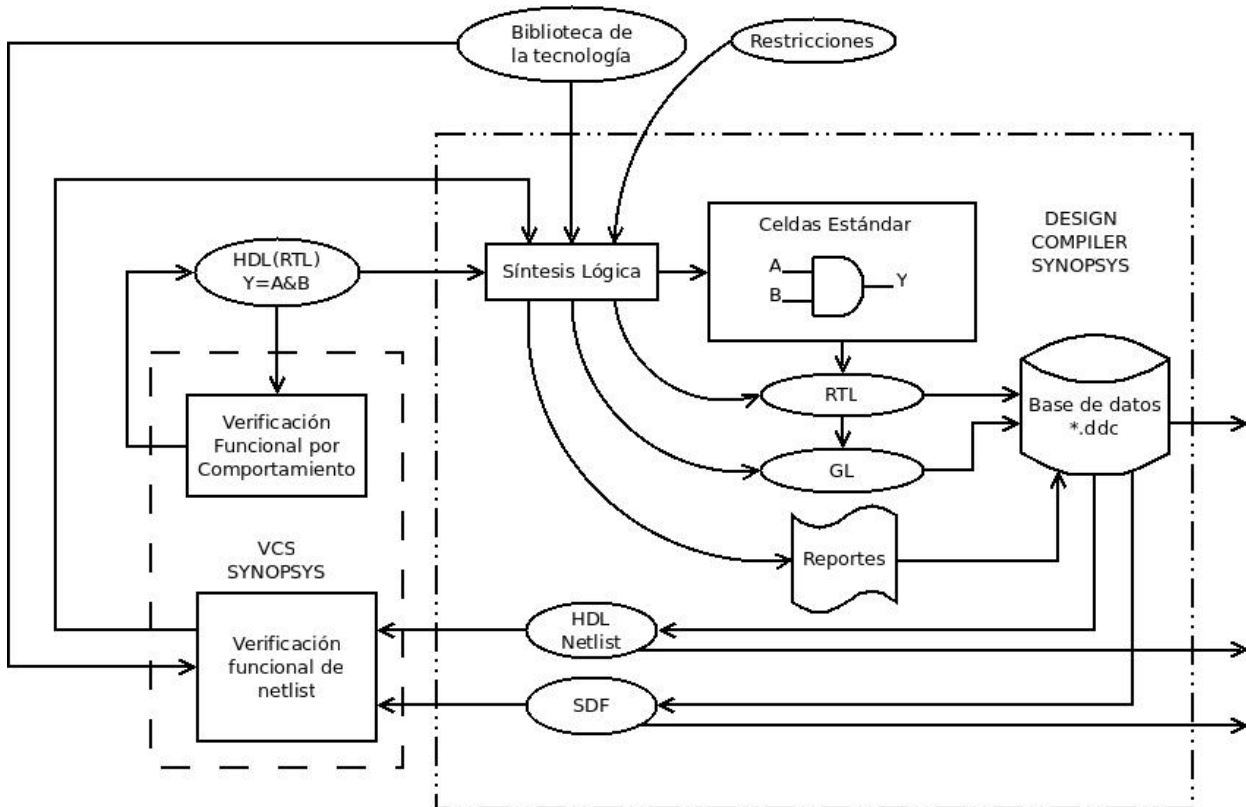


Figura 2.4: Esquema ilustrativo del flujo de diseño de circuitos integrados digitales, en la etapa inicial de síntesis lógica (conocida como etapa de front end). Imagen de autoría propia.

El término *Front End*, está asociado a la jerga del desarrollo de software, y significa interfaz, que no tiene mucho que ver con el diseño de circuitos integrados. Una traducción más literal, y por lo tanto, menos elegante del concepto sería “fachada”, y es en esencia con lo que el diseñador interactúa en la primer etapa del diseño.

Al decir que el diseñador interactúa con una fachada, se debe entender que el diseñador trabaja con el cascarón de un elemento que intrínsecamente no es circuito electrónico como tal, si no un modelo abstracto que presenta un comportamiento de causa efecto, el cual, naturalmente, es acorde a la semántica de la lógica digital, o lógica binaria. Recurriendo nuevamente a la jerga del software, podemos entender el *front end* o fachada como aquello relacionado con un objeto de alto nivel, que ofrece un vector de respuestas de acuerdo con la dinámica de un vector de estímulos.

El diseñador comienza a acercarse al diseño del circuito integrado, considerando y definiendo los aspectos principales de la fachada. Retomando el diagrama de Gajski (figura 2.2), el proceso de "Front-End" consiste en la transición del modelo de comportamiento a un modelo estructurado, que se asocia con las funciones de las celdas estándar ofrecidas por la tecnología y el fabricante.

Condensando lo anterior, las herramientas de *front end* de un proveedor, son aquellas que ejecutan o facilitan el proceso de síntesis lógica en el flujo digital del diseño de circuitos integrados. En el caso de Synopsys, estas herramientas son principalmente Design Compiler y VCS, existen otras herramientas adicionales asociadas a la familia de Design Compiler, VCS y el Shell de Design Compiler son las principales y suelen ser suficientes.

En la figura 2.4 se aprecia un diagrama que ilustra como el proceso expuesto en la sección 2.2.1, se implementa mediante las herramientas de Synopsys todos los procesos en el diseño de circuitos integrados tienen un componente iterativo, y en este diagrama se aprecia como antes de avanzar en el flujo, se atraviesa por etapas a las que se recurre con frecuencia (etapas de simulación de comportamiento y de post síntesis). La depuración de la etapa de síntesis es esencial para garantizar el éxito final del proyecto, por lo que las simulaciones tienen una gran importancia.

Cómo se observa desde el diagrama de Gajski (figura 2.2), las herramientas de *front end* traducen el diseño por comportamiento en modelos de alto nivel de circuitos digitales, aunque como tal no se trabajan con circuitos lógicos reales. Para efectos prácticos el diseñador nunca trabaja realmente con elementos lógicos y digitales: las herramientas proveen la virtualización del comportamiento diseñado, mediante una biblioteca de simulación. Las celdas estándar abstraídas en el proceso de síntesis se comportarán como un elemento de circuito digital real. De ahí que se pueda validar si la etapa de síntesis cumple las expectativas del diseñador.

2.2.4 Back End

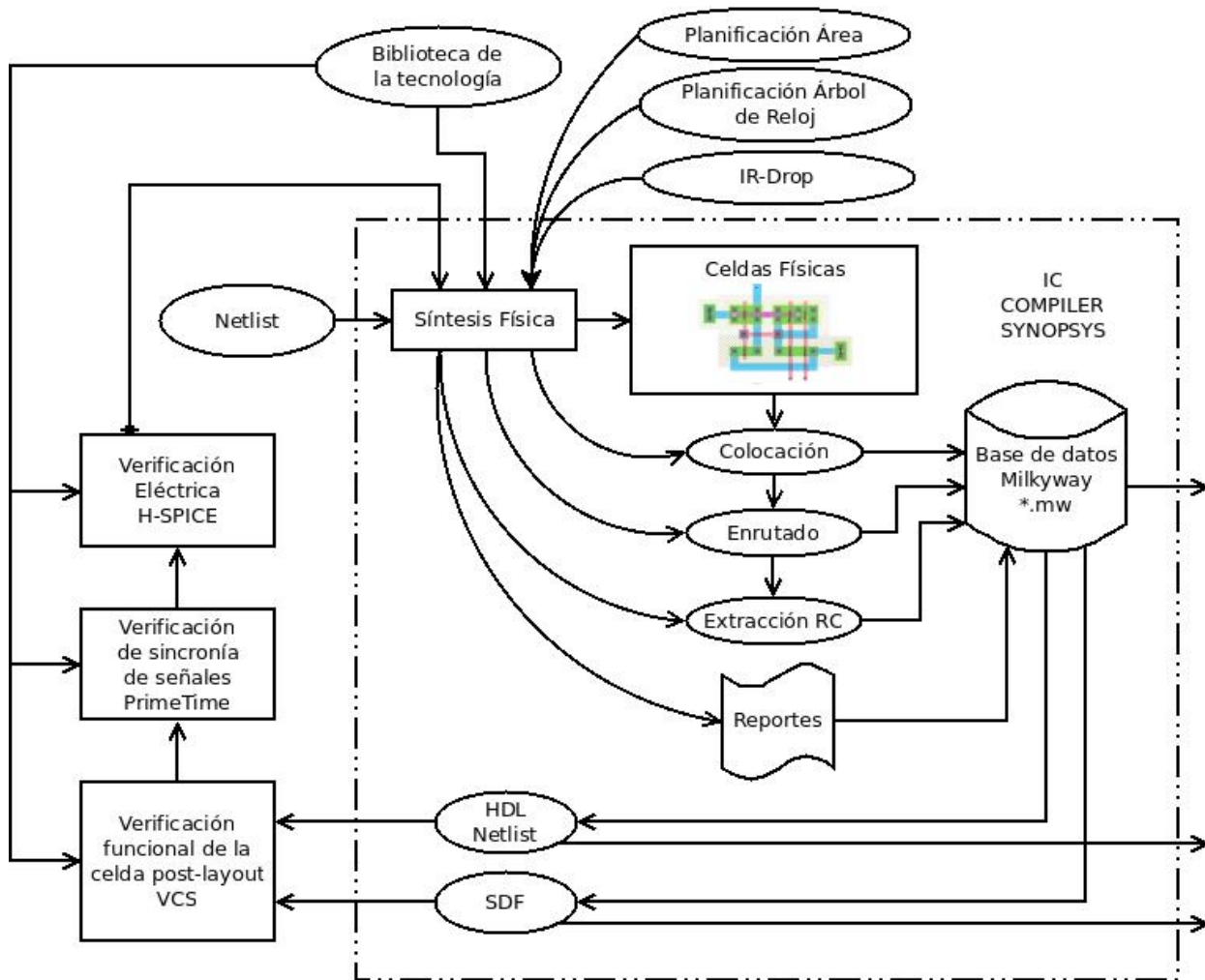


Figura 2.5: Esquema ilustrativo del flujo de diseño de circuitos integrados digitales, enfocado a las herramientas de síntesis física, en esencia las herramientas de *Back End*. Figura de autoría propia.

El *back end* refiere a aquellos procedimientos relacionados con la constitución interna de un proyecto, nuevamente se emplea el anglicismo "Back End" para referirse a dichos procesos. No existe un concepto en español capaz de traducir el significado de esta palabra, pero se podría entender como todos aquellos aspectos ocultos al macro diseñador, entendiendo por oculto, una serie de elementos compuestos por elementos más simples hasta llegar a las unidades fundamentales de la electrónica moderna, i.e. transistores. Una analogía curiosa, y que ilustra bien esta situación es la de la muñeca matryoshka [30]: cada muñeca contiene a otra muñeca más pequeña, y se interioriza así hasta alcanzar una pequeña muñeca que no contiene a ninguna otra.

En el proceso de diseño de circuitos integrados semipersonalizados, y basados en celdas estándar, el diseñador no alcanza a trabajar directamente con transistores, o con los elementos que componen las celdas, sino que da por hecho que las celdas estándar son funcionales y a partir de ellas (las celdas estándar) se generan nuevas macroceldas, las cuales ejecutarán las funciones macromodeladas con el RTL del diseño.

En síntesis de lo anterior, el diseñador de "back end" trabaja con las etapas ocultas para los diseñadores de comportamiento, y síntesis del diseño, donde los primeros establecen la semántica del diseño, y los segundos generan cajas negras que realizan las funciones descritas por sus antecesores. Estas cajas negras manifiestan el comportamiento de una unidad electrónica real, en términos de consumo energético, área, y tiempos de propagación de señales, pero no dejan de ser modelos de alto nivel que no son circuitos electrónicos como tales, y es por ello que esta etapa se le denomina fachada.

En el proceso de *back end* las cajas negras abstraídas del proceso de síntesis, adquieren información más realista y equivalente a la de una celda física real. Esto se aprecia en la figura 2.3, donde en el extremo derecho de la figura se observan los productos de las distintas etapas del flujo:

En el dominio estructural (donde operan las herramientas de síntesis lógica) se muestran los bloques funcionales a nivel lógico.

Al descender en el flujo, se aprecia que la celda estándar es abstraída del diseño estructural, i.e. se asocia con una celda trazada (layout), donde pueden observarse: los metales que interconectan las celdas y los transistores que componen las celdas.

Las herramientas de *back end* se encargan de facilitar el colocado y enrutado de las celdas estándar que se abstraieron de la síntesis lógica. En el *back end* se asocia la base de datos post síntesis con la base de datos de la tecnología, invocando celdas físicas para generar un nuevo layout que implemente el diseño sintetizado. Para esta sección del diseño, se utiliza la herramienta IC Compiler, que permite al diseñador establecer los criterios de optimización del trazado en términos de aprovechamiento de área, colocación selectiva para mejorar la eficiencia del consumo energético, sincronización de señales críticas (tal como la señal de reloj), definir criterios de alambrado (enrutado) para sopesar las pérdidas resistivas, y controlar fenómenos de electromigración y radiación o antena.

Capítulo 3

Flujo de Diseño Digital de Circuitos Integrados con las herramientas de Synopsys. Estructura de scripting

A continuación se detalla la estructura propuesta para implementar un flujo de diseño de circuitos integrados, y se expone como fueron sometidos los módulos de un microprocesador de aplicación específica basado en la plataforma abierta RISC-V. Con el fin de demostrar la efectividad del flujo de diseño digital propuesto. Esta estructura está basada en el manual sobre el uso de las herramientas de Synopsys en el flujo de diseño de ASICs, que ya ha sido desarrollado en el DCILab [31].

La estructura objetivo para el presente proyecto de graduación dista considerablemente de la propuesta en [31] y que a su vez está basada en el manual [32]. En estos trabajos las estructuras se basan en ejemplos simples con diseños *HDL* relativamente pequeños. Para este trabajo se toma como premisa que se trabaja con diseños muy grandes por lo que se requiere de mayor eficiencia en la generación de las bases de datos y archivos necesarios para el diseño.

Dado que las herramientas EDA que se utilizan corren sobre un sistema operativo Linux, muchos de los scripts que se encuentran en esta estructura hacen referencia a particularidades de este sistema operativo, que no se aclaran pues se consideran deberían ser conocimiento básico de los destinatarios de este documento.

La primera propuesta de esta estructura fue desarrollada por el Dr. Juan Agustín Rodríguez y se tomó como referencia para desarrollar la que a continuación será expuesta. La estructura que se propone se basa en tres principios muy relacionados con el mundo de desarrollo de software; sin embargo, apelan más a la experiencia práctica del Dr. Rodríguez y del aspirante que escribió este informe.

- **Regularidad:** este concepto se refiere a la uniformidad con la que los elementos desarrollados se ubican, dentro de la estructura de archivos y directorios, los cuales están basados en un principio o plan.

Esta estructura es "regular" en tanto presenta la misma morfología para los distintos directorios, esto quiere decir que aunque los directorios contienen archivos de distinta naturaleza, la mayoría presentan una jerarquía constante entre sí.

- **Localidad:** se refiere a usar archivos y directorios únicos, para evitar el manejo de múltiples versiones y múltiples rutas de acceso a archivos que en esencia son iguales, lo cual suele ser un problema cuando se debe involucrar a más personas en un proyecto.

La estructura presenta localidad en tanto los archivos fuente y las bases de datos generadas por los procesos de síntesis, únicamente residen en un único directorio y presentan sólo una ruta de acceso, no se generan copias de estos archivos en otros directorios, y la forma en que se accede a ellos es mediante punteros. Existen algunas excepciones a este principio; sin embargo, serán expuestas más adelante y se justificará el porqué de estos archivos.

- **Continuidad:** la continuidad hace referencia a dos conceptos; el primero es sobre la capacidad de poder usar elementos en la estructura del flujo para rastrear el diseño en una etapa dada hasta la etapa de origen que en este caso corresponde al modelo de comportamiento del diseño. El segundo se relaciona con la capacidad de ser permanente, o en otras palabras tener la capacidad de degradarse con lentitud. Es frecuente encontrar que entre diseñadores existan desacuerdos sobre la mejor forma de nombrar los archivos o la mejor manera de ubicarlos y realmente no es posible definir una estructura como la más eficiente o la estructura suprema, pues apela a muchos criterios subjetivos y relativos, tanto por parte de los diseñadores como por la naturaleza de los proyectos.

Finalmente se afirma que la estructura es continua, ya que la navegación dentro de los directorios permite identificar de forma fácil e intuitiva, cada etapa del flujo. Mediante los scripts, y un conocimiento general del flujo descrito en 2.2.1, puede ubicarse cualquier resultado, y rastrear los archivos fuente que los generaron (los resultados). Respecto a la parte de continuidad en el tiempo, se espera ofrecer una estructura lo suficientemente elegante y práctica para que pueda seguir siendo utilizada en futuros proyectos y que los diseñadores que los enfrenten puedan sentirse cómodos con ella.

3.1 Estructura de directorios y archivos

Como puede apreciarse en la figura 3.2, el directorio base del proyecto contiene hasta el momento siete directorios y dos archivos, uno de los cuales es oculto y carece de relevancia para el proyecto. Comenzando por el archivo `bash_synopsys.sh`, tenemos un script del intérprete de comandos `bash`, que es una de las tantas consolas que ofrecen los ambientes Linux. Este script contiene punteros hacia los archivos ejecutables del juego de herramientas de Synopsys, la licencia para que puedan correr, y finalmente cualquier otro software afín al proyecto pueda usarse.

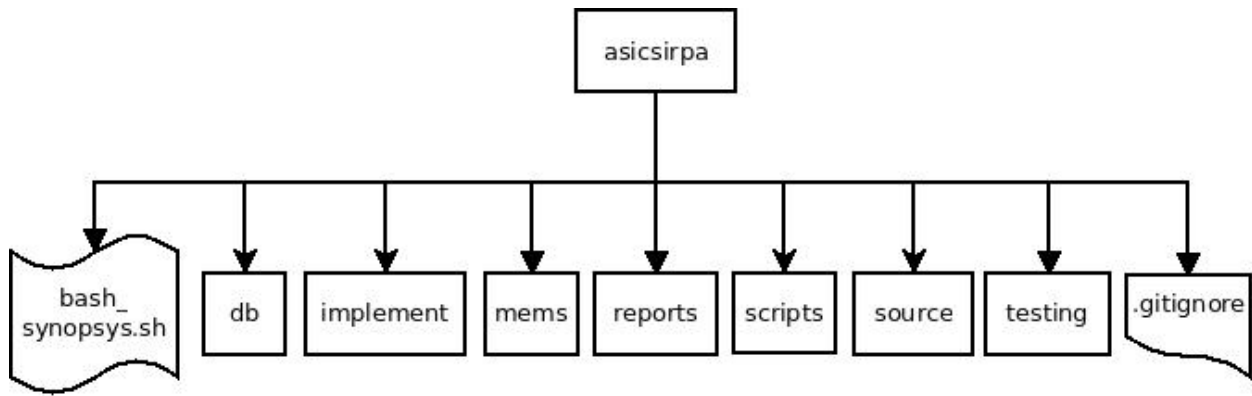


Figura 3.1: Esquema de la jerarquía de directorios y archivos para implementar el flujo de diseño digital. El archivo `.gitignore` es un archivo oculto, que contiene información para que la herramienta de control de versiones usada en el DCILab opere de forma personalizada. Este archivo carece de relevancia para la estructura del flujo; sin embargo, permite mantener un repositorio más ordenado. Figura de autoría propia.

Dentro de la estructura propuesta existe un archivo importante para que los scripts y las herramientas trabajen de forma más eficiente. El archivo en cuestión es el `.bashrc`, que se ubica en el directorio `HOME` del usuario (universal en cualquier sistema linux moderno). Este archivo es ejecutado cada vez que se abre una terminal o consola de tipo `bash`. Le permite al usuario personalizar distintos aspectos de la consola, como:

Contar con la posibilidad de disponer de las herramientas sin tener que ejecutar el script de inicialización (`bash_synopsys.sh`) cada vez que se abre una nueva terminal, invocando por defecto al script de inicialización.

Si existen copias del proyecto en más de un computador, se tendrá una ruta absoluta diferente hacia el directorio del proyecto para cada máquina. Sin embargo, una vez dentro del directorio del proyecto, las rutas hacia directorios y archivos se vuelven universales. Usar variables de entorno permite que en los scripts hayan punteros hacia el directorio principal del proyecto, esto evita la necesidad de editar los scripts, en caso de querer ejecutarlos en computadores diferentes.

En el bloque de código 3.1 podemos observar un ejemplo que ilustra la ejecución del script de `bash` para incluir en la variable de ambiente `PATH`, las rutas a los ejecutables de las herramientas EDA (línea 8), y la creación de 2 variables con la ruta relativa al directorio base del proyecto (variable “`asic`”, línea 3) y al directorio de la biblioteca de la tecnología (variable “`TECH`”, línea 5).

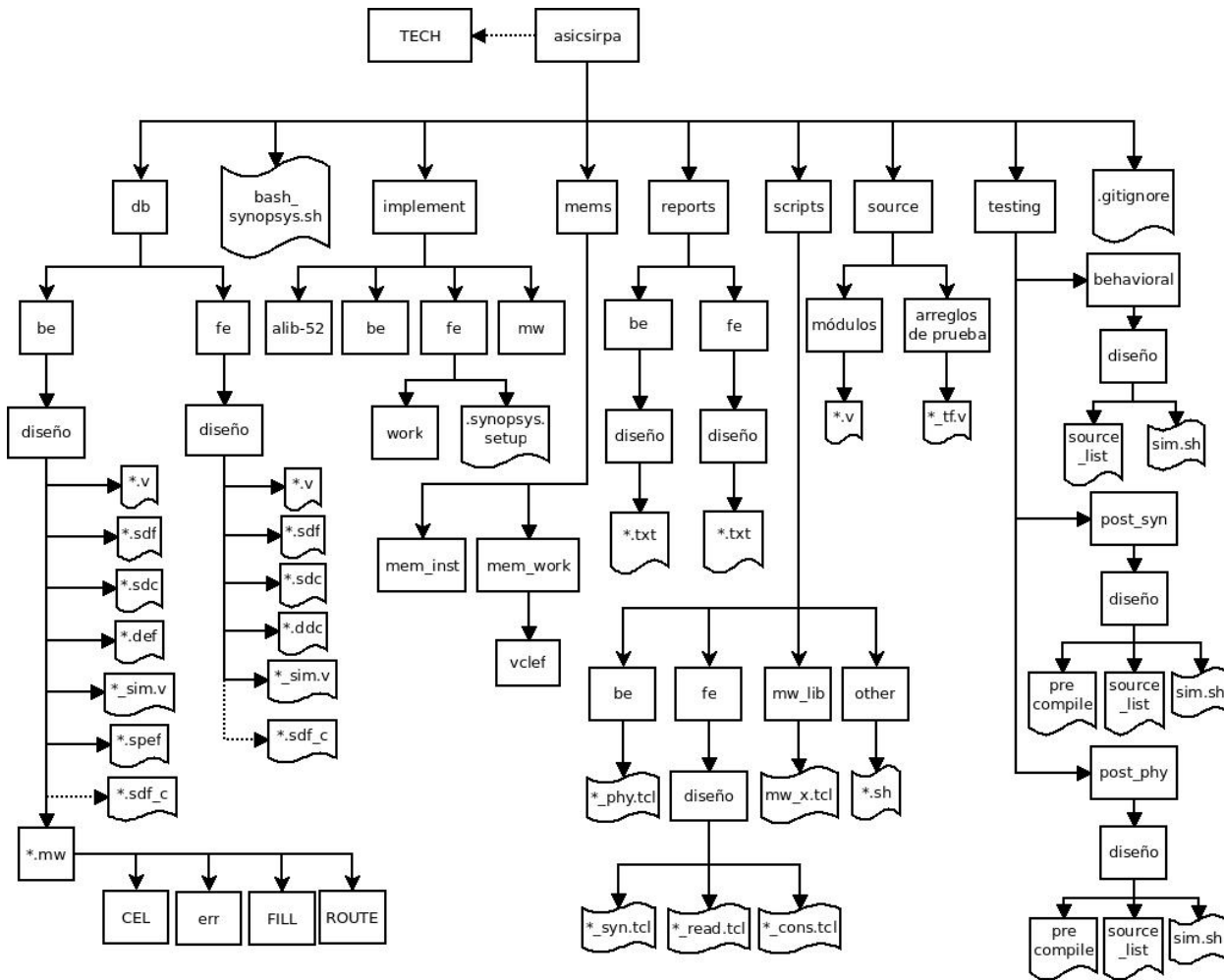


Figura 3.2: Esquema completo de la jerarquía de directorios y archivos que implementan el flujo de diseño digital. El bloque **TECH** hace referencia a la biblioteca de la tecnología. No es una buena idea agregar este directorio en la misma ubicación que el proyecto ya que es un directorio de tamaño considerable y copiarlo en cada proyecto representa un mal uso de los recursos. Figura de autoría propia.

Listado 3.1: Directivas para crear las variables de entorno utilizadas por los scripts de las herramientas e invocar el script de las herramientas de Synopsys.

```

1 # Home for working on the SiRPA ASIC Integration Project
2 asic=/home/rcastrro/asicsirpa
3 export asic
4 # IBM technology files for the Integration Project (Library)
5 tech=/home/IBM/TECH
6 export tech
7 # Load the EDA Synopsys executables
8 . $asic/bash_synopsys.sh
9 # Load a text processor application
10 PATH=$PATH:/home/tools/sublime_text_3
11 export PATH

```

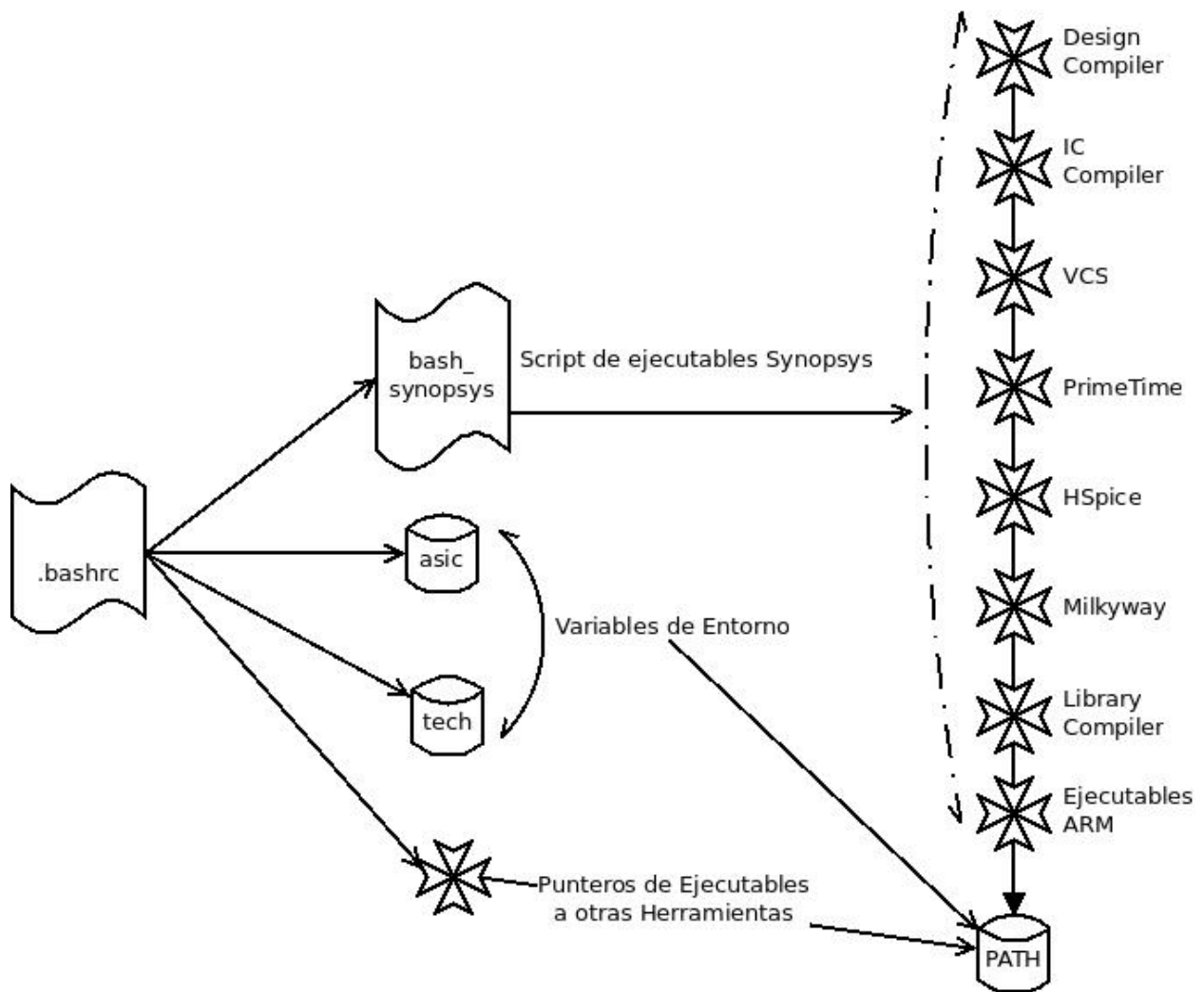


Figura 3.3: Esquema ilustrativo de la relación entre los scripts de bash que inicializan las herramientas EDA y crean las variables que contienen los punteros a los archivos y la biblioteca de la tecnología de integración. Figura de autoría propia.

Retomando el diagrama de la figura 3.2, dentro del directorio del proyecto de integración encontramos siete directorios, de los cuales `db`, `implement`, `reports` y `scripts` presentan la misma estructura regular, pues en cada uno de estos directorios encontramos otros 2 directorios llamados `be` y `fe` (que como podrá intuirse hacen referencia a back end y front end respectivamente). Se decidió usar estas abreviaciones por su simplicidad, pues resumen muy concretamente a qué hacen referencia y son muy intuitivas.

El directorio `db` alberga las bases de datos y demás archivos que son producto de los procesos de síntesis, naturalmente en el subdirectorio `fe` sólo se almacenan los resultados de la síntesis lógica del proyecto.

El directorio `implement` alude al proceso de ejecución de las herramientas y los procesos de síntesis; este directorio no contiene archivos de gran relevancia para el proyecto, salvo el directorio `fe` el cual contiene el archivo oculto, `.synopsys_dc.setup`, que es un script de TCL con comandos de inicialización y preconfiguración de la herramienta Design Compiler. Este script es de vital importancia para que la herramienta opere de forma adecuada, en la sección 3.2 se expondrá un poco más sobre este archivo.

El directorio `implement` concentra los archivos temporales que se producen al ejecutar las herramientas de síntesis. En `fe` se tienen los directorios `work` y `alib-52`, el primero sirve para codificar el RTL y generar información útil para la compilación. El segundo alberga las bibliotecas producto de la compilación del diseño, éstas bibliotecas contiene un compendio de las celdas estándar necesarias para implementar el RTL y sirve para acelerar compilaciones subsecuentes.

Finalmente dentro del directorio `implement` tenemos otro directorio llamado `mw`, este directorio contiene los reportes y anotaciones (`log`) de la ejecución de la herramienta Milkyway, que es una herramienta de back end y podría incluirse en el directorio `be`; sin embargo, se considera adecuado que esta herramienta tenga un directorio particular para sí, pues se concentran las anotaciones de cada herramienta en directorios separados, facilitando la búsqueda de información.

Siguiendo con los directorios que presentan regularidad, encontramos los directorios: `reports` y `scripts`, los cuales contienen reportes y scripts respectivamente. Los reportes generados son distintos de los que genera la herramienta a la hora de estar haciendo el proceso de síntesis. Estos reportes corresponden a la información final de síntesis y proveen información útil en la toma de decisiones de las etapas posteriores. Los scripts, son códigos principalmente de TCL que dirigen a las herramientas para generar los procesos de síntesis. En el directorio `scripts` se observan otros dos directorios, `mw` y `other`, que contienen respectivamente contienen scripts para la herramienta Milkyway, y de otras herramientas como por ejemplo MATLAB y los generadores de memorias SRAM de ARM, los cuales serán expuestos más adelante.

Los tres directorios restantes, `mems`, `source`, y `testing`, no presentan una estructura regular como la que ha sido expuesta hasta el momento. El directorio `testing` podría incluirse en esa categoría; sin embargo, la subdivisión que presenta es bastante puntual e intuitiva, así que se considera más práctica. El directorio `testing` (una palabra en inglés cuya traducción es “pruebas”), aloja los resultados de las distintas simulaciones con las que se valida la funcionalidad de los módulos sometidos al flujo. Las simulaciones se subdividen en tres categorías: por comportamiento, en inglés, `behavioral`, post síntesis lógica (`post_syn`), y post síntesis física (`post_phy`). Estos tres directorios cuentan con tres archivos principales, los punteros de: compilación y los archivos fuente del RTL, y un script en `bash` que permiten invocar la herramienta de simulación y ejecutar las compilaciones y simulaciones respectivas, luego de la simulación puede que sea de interés guardar un vector de resultados en un archivo de texto. Los directorios de simulación presentan cuatro archivos relevantes, los archivos extra que puedan encontrarse en estos directorios son producto de los apuntes y registros que genera la herramienta durante la compilación, y pueden obviarse.

El directorio **source**, que se traduce como fuente, contiene dos categorías de directorios: el primero corresponde a los archivos HDL con la información modular del diseño, es decir los archivos con la descripción por comportamiento del diseño, y también los archivos con los vectores de estímulos y arreglos o bancos de prueba, es decir los *testbenchs* o *testfixtures*. Los primeros usados en el proceso de síntesis lógica, y los últimos necesarios para las simulaciones en los distintos dominios.

Por último tenemos el directorio **mems**, el cual es un directorio bastante particular. Contiene la información producida por los aceleradores de diseño de ARM, los cuales generan celdas de memoria SRAM. Se trata este proceso aparte ya que los programas para generar las celdas de memoria tienen una metodología de trabajo un poco diferente de la del flujo digital que se ha venido exponiendo. La justificación de manejar estos directorios aparte recae en no mezclar las metodologías de trabajo; sin embargo, el producto final de este procedimiento se integra a la estructura que se ha venido describiendo, es decir se generan bases de datos para las etapas de back y front end, posteriormente en la sección 3.4 se ahondará en detalles sobre la síntesis de las SRAM.

En este directorio (**mems**), se encuentran dos directorios principales, **mem_inst**, el cual contiene los ejecutables de las herramientas de ARM para sintetizar las memorias, es importante que sólo exista una única copia de este directorio, ya que es muy pesado (en términos de espacio, cerca de 2 GB). El otro directorio es **mem_work**, el cual contiene el resultado de la síntesis de las celdas SRAM y finalmente **vc1ef** el cual es un archivo con la información de los metales y el enrutado de la celda SRAM física, este último se crea con el fin de facilitar la ejecución de la herramienta Milkyway y generar las bases de datos necesarias para los procesos de back end. Nuevamente esto será discutido en detalle en la sección 3.4.

3.2 Scripts para diseño de front end

3.2.1 Script de síntesis lógica

En la figura 3.4 se observa como la síntesis lógica se ejecuta mediante tres scripts, dos de los cuales son auxiliares al script de síntesis. Los scripts reciben un nombre representativo del diseño con el que se trabaja; aunque suele ser conveniente nombrar los archivos de la misma manera al módulo principal, si este nombre es muy largo, es preferible optar por un nombre representativo.

Es aconsejable separar las instrucciones de síntesis de la herramienta para no generar scripts densos en código y modular las operaciones ejecutadas por la herramienta.

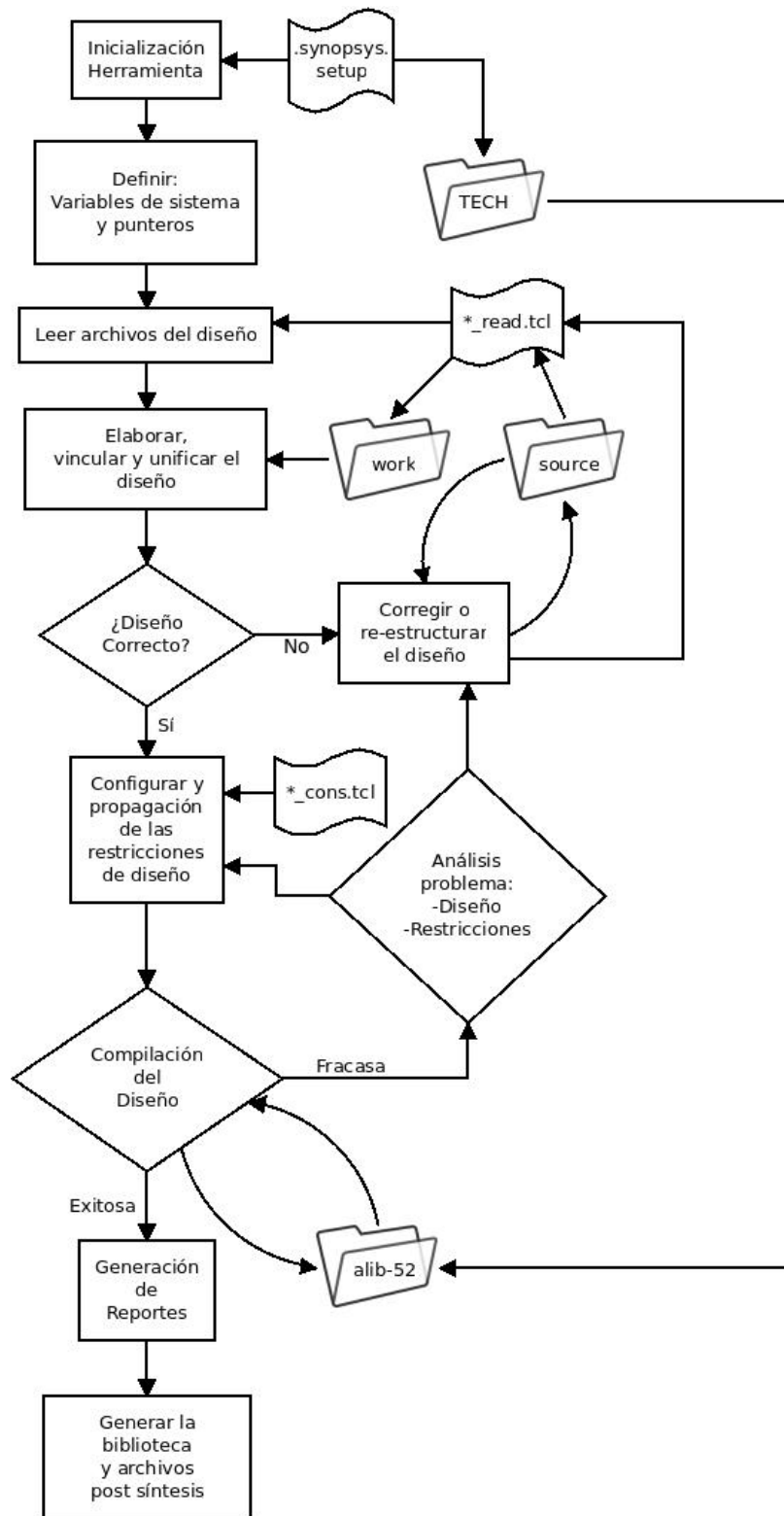


Figura 3.4: Diagrama de flujo del scripting que ejecuta la síntesis lógica. Figura de autoría propia.

Existe un único flujo en la ejecución de los scripts. El script principal, (`*_syn.tcl`) se describe en la figura 3.4, a partir de la etapa en la que se definen los punteros y las variables del sistema, continúa en línea recta hasta la generación de la base de datos y demás archivos post síntesis. En un apartado anterior se mencionó sobre la importancia del script `.synopsys_dc.setup`. Este archivo contiene comandos y procedimientos de TCL que ejecutan tareas configuración, tal como inicialización de parámetros, variables, y declaración de las bibliotecas de diseño.

Cuando se invoca Design Compiler, se busca el archivo configuración (`.synopsys_dc.setup`) en la dirección desde dónde la herramienta fue invocada. Se cargan las rutas a los archivos de interés, la biblioteca de la tecnología, alias y demás variables para inicializar el flujo de síntesis del proyecto. Cabe destacar que no toda la configuración de parámetros y variables se realiza en el archivo `texttt.synopsys_dc.setup`, y aunque es posible, tampoco es eficiente hacerlo. Este archivo tiene un carácter genérico y universal para cualquier diseño sobre la tecnología definida. Su función es poner la herramienta a punto para que este en capacidad de sintetizar correctamente cualquier diseño. Las particularidades de los diseños que se trabajan son contempladas en el script TCL de restricciones (*constraints*).

Una vez que la herramienta ha sido inicializada, se procede a configurar variables y parámetros propios del diseño con el que se trabaja. Estos son principalmente variables de rutas hacia el destino de los archivos de salida tales como reportes, así como punteros hacia la ruta a bibliotecas auxiliares para el diseño en particular. Por ello se crean variables con el nombre del proyecto para que al generar los archivos de salida se usen los comandos de forma paramétrica.

Script de lectura de archivos fuente

Cuando finalmente se han definido todos los parámetros y variables generales para el diseño particular con el que se trabajará, se procede a leer los archivos fuente del diseño (los módulos descritos en HDL). El conjunto de estos archivos HDL suelen denominarse como diseño descrito en RTL (Register Transfer Level).

Con TCL la lectura y análisis de los módulos se puede realizar de forma programática. En el script `*_read.tcl` se usa una variable con el puntero hacia el directorio base de los archivos HDL fuente; luego mediante una subrutina de TCL se crea una colección con los directorios que contienen los distintos módulos. Posteriormente se ejecuta un pequeño algoritmo recursivo que recorre las colección de los directorios, donde para cada directorio se crea otra colección con los módulos, los cuales nuevamente son leídos de manera recursiva. Al recorrer esta última colección se ejecuta el comando de análisis de la herramienta. Este pequeño algoritmo evita repetir la misma instrucción en el `script`.

En la figura 3.5 se observa un diagrama de flujo, con el resumen gráfico del algoritmo descrito en los párrafos anteriores. Cabe destacar que en el bloque que indica la definición de variables y parámetros hace referencia a variables locales para ejecutar las tareas de lectura, aunque en realidad únicamente se crean punteros (a los que también se llama: variables de ruta) hacia las rutas base de los archivos fuente.

Cabe destacar que para algunos diseños, las variables de ruta no necesariamente apuntan a directorios con contenido *HDL*, pues como suele hacerse en proyectos muy grandes, conviene dividir el diseño general en subdiseños más compactos. Ello implica que para algunos proyectos es preferible leer las bases de datos generadas para otros diseños, pues es más práctico.

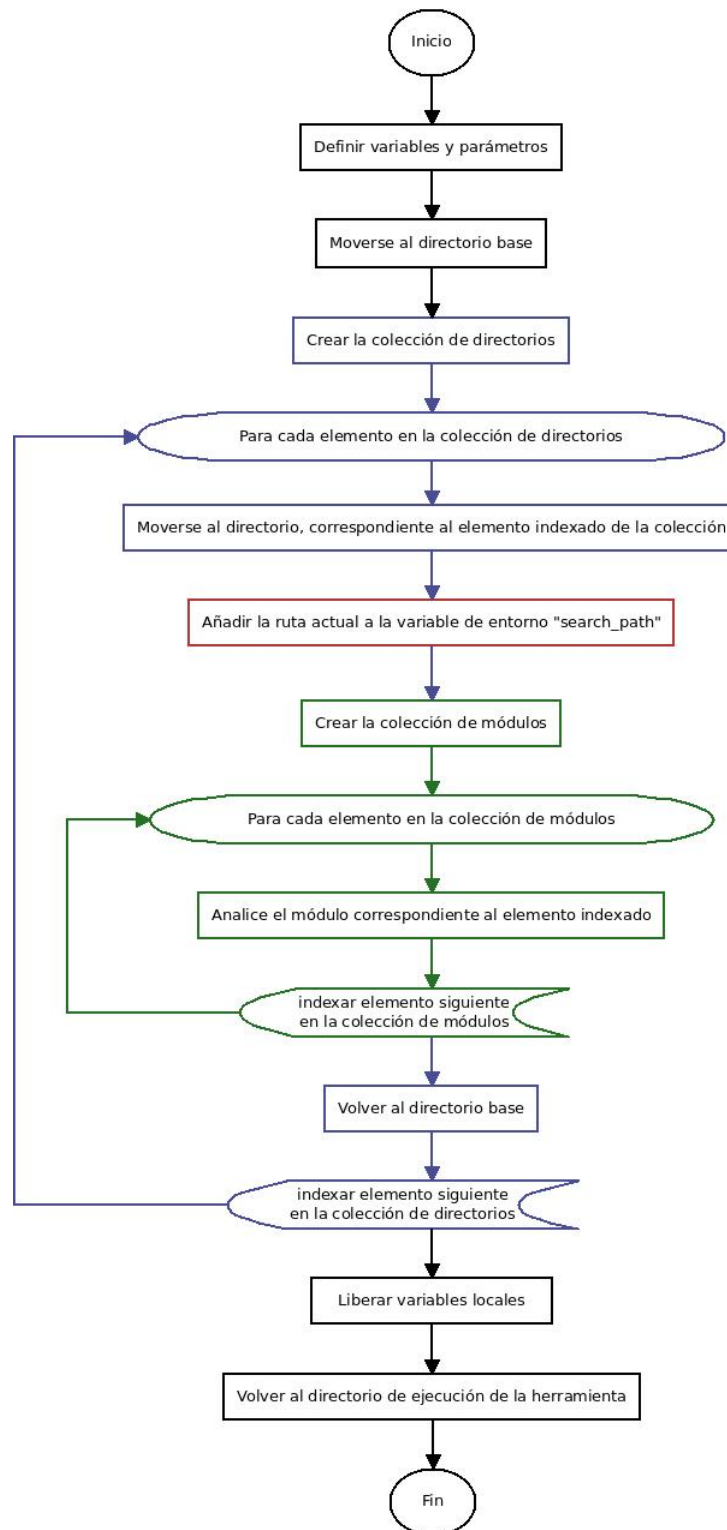


Figura 3.5: Algoritmo recursivo de lectura y análisis de los módulos *HDL* del diseño. Figura de autoría propia.

Otra observación que se debe realizar al diagrama de la figura 3.5 es que no en todos los diseños es necesario crear una colección de directorios, pues si todos los archivos fuente de ese diseño se concentran en un directorio, recorrer la colección de directorios carece de sentido. Consecuentemente puede omitirse el lazo de color púrpura, correspondiente al algoritmo

iterativo de indexación de directorios. El único lazo estrictamente necesario para leer los archivos fuente, es el lazo en color verde. Se incluyen ambos lazos en la figura 3.5, y se expuso el proceso de lectura y análisis con los dos lazos para ilustrar el caso más general de lectura de archivos fuente. Naturalmente los scripts deben ajustarse a la estructura particular del diseño; sin embargo, el algoritmo expuesto es fácilmente adaptable a cada caso.

El bloque en color rojo de la figura 3.5, corresponde a añadir en la variable `search_path`. Si al ejecutar el comando de análisis y la ruta relativa al archivo no ha sido incluido en la variable de búsqueda (`search_path`), la herramienta ofrecerá un error, pues se pretende acceder a una ubicación que no ha sido autorizada para la herramienta

La variable `search_path`, es una variable TCL que define una ruta de búsqueda de archivos fuente y bibliotecas para la herramienta.

Finalmente conviene volver a la ubicación desde donde se ejecutó la herramienta para evitar crear archivos de apuntes, o archivos temporales con poca relevancia para el proyecto, en directorios ajenos a los destinados para esta tarea. Basta decir que liberar todas las variables locales al finalizar una subrutina es una buena práctica de programación, así que es conveniente hacerlo.

Volviendo a la ejecución del `script` principal de síntesis, cuando ya todos los módulos se han leído, en el directorio `work` se crean archivos con información sobre el resultado del análisis de los módulos. Estos archivos serán fundamentales para avanzar en el flujo de síntesis, pero una vez acabado el flujo y habiendo generado la base de datos del diseño, dejan de ser relevantes.

Dicho lo anterior, se debe proceder a elaborar el diseño, que consiste en definirle a la herramienta cuál es el módulo principal del diseño, posteriormente se deben vincular y unificar todos los módulos. La vinculación y unificación establece la jerarquía modular del diseño y resuelve la interdependencia de los módulos.

En la figura 3.4 se observa un bloque condicional (diamante de decisión), evaluando la condición si el diseño es correcto. En otras palabras, la herramienta evalúa entre otras cosas, si la semántica del código es correcta, no existen errores sintácticos y todas las dependencias se cumplen. Este bloque condicional no se asocia a una subrutina programática en el script, de hecho ninguno de los bloques condicionales que se aprecian en este diagrama lo hace, en la semántica del flujo también interviene el criterio del diseñador y este es un claro ejemplo de esta situación.

Muy probablemente cuando se le solicite a la herramienta evaluar el diseño, aparecerán mensajes de información, alarmas, y en el peor de los casos, errores. El diseñador entonces, deberá evaluar el escenario con el que se enfrenta, corregir los errores es imperativo; sin embargo, muchos de las alertas son tolerables o inofensivas, así que de acuerdo con la experiencia del diseñador se deberá mejorar o no el diseño, y solucionar o aprobar los mensajes de alerta. ¡Un ingeniero en diseño VLSI, debe aprender a vivir bajo el warning!

Si el diseñador no se siente satisfecho con el resultado de la comprobación de la herramienta, y los mensajes sugieren que debe hacer correcciones a los archivos fuente del diseño, lo más sensato es ponerse en contacto con el departamento o personal encargado del diseño por comportamiento y exponer la situación. En algunos casos el diseñador de *front end* podrá verse arrastrado a analizar y corregir el código fuente, lo que suele ser muy raro, ya que las tareas en el flujo de integración suelen estar muy bien definidas de acuerdo con departamentos creados para delimitar y definir tareas y responsabilidades (al menos en las empresas grandes de diseño VLSI); sin embargo, para diseños de baja envergadura, e instituciones académicas es común encontrar a una única persona realizando todas las tareas relacionadas al flujo.

Cuando la comprobación del diseño es aprobada, lo siguiente es establecer las cotas del comportamiento (es decir, las restricciones de operación del diseño) y propagarlas por la jerarquía; así todas las dependencias tendrán las mismas condiciones de funcionamiento. Como por ejemplo, se definen las características de señales críticas como el reloj.

Script de restricciones

En esta etapa es donde se invoca el `script` de restricciones, en inglés *constraints*. Este `script` se encuentra en el flujo del diagrama de la figura 3.4 como `*_cons.tcl`. Para este `script` no hace falta mostrar un diagrama de flujo sobre su comportamiento ya que en principio sólo define parámetros y variables de entorno de la herramienta Design Compiler.

A continuación se presenta un pequeño resumen de los parámetros que se consideran suficientes para cumplir a cabalidad las expectativas de desempeño del diseño.

- **Señal de reloj:** esta señal necesita ser atada con el pin adecuado del diseño sintetizado, con esto se refiere a generar un objeto¹ para que la herramienta pueda crear un árbol de reloj hacia los módulos sincrónicos del diseño. Debe definirse un nombre representativo para la señal en el diseño. Hay que recalcar que es conveniente definir una nomenclatura estándar para un proyecto grande que involucre diseños jerarquizados, así cada subdiseño tendrá una señal de reloj con las mismas propiedades y un mismo nombre.

Una vez se crea el objeto “señal de reloj”, se definen sus propiedades, tales como: el periodo, la incertidumbre, el sesgo y la latencia de la señal, siempre asociando cada propiedad al objeto “señal de reloj” que ha sido creado. La herramienta permite definir más propiedades, o las mismas de formas diferentes; sin embargo, se considera que estas son suficientes.

- **Propagación de señales:** cuando se tienen señales universales a todos los módulos secuenciales, como lo suelen ser las señales de reloj, de *reset* (reinicio), de *enable* (habilitación), o de *preset* (preconfiguración), etc, es conveniente definir condiciones de comportamiento para la herramienta de síntesis lógica, ya que esta suele hacer cosas

¹Entendiendo por objeto al concepto de programación de alto nivel asociado al ente programático capaz de presentar una colección de atributos y permitirle a funciones complejas invocarle y utilizar sus atributos en tareas específicas.

indeseables como colocar buffers en el camino de propagación de dichas señales, que serán ruteadas por comoandos especiales del ICC. Es por ello que para estas señales se define un atributo para que la herramienta no realice optimizaciones sobre ellas. Y el diseñador tendrá un mejor control sobre lo que suceda con esas señales. Para este trabajo los diseños únicamente necesitan obviar la optimización de la señal de reloj y reset, que se definen como intocables.

- **Comportamiento de otras señales:** para generar de forma adecuada los modelos de simulación, y hacer estimaciones certeras sobre la sincronización de las señales, es beneficioso definir los retardos de propagación de entrada y salida de todas las señales, excepto las mencionadas anteriormente (reloj y reset). Los estímulos en el cableado no se propagan de forma inmediata, por lo que se debe procurar emular un comportamiento cercano a la realidad, para que el resultado de la síntesis del diseño sea lo más confiable posible. Esta información puede re-anotarse luego (back annotate) para volverla más precisa, con los resultados parciales de las primeras síntesis de la herramienta.
- **Modelo de carga para el cableado:** continuando con la idea de generar un modelo lo más realista posible, se deben considerar las características energéticas del cableado.

Conocer las parasitancias de los materiales que se pretenden usar para alimentar e interconectar los módulos del diseño, permite elaborar modelos de comportamiento más cercanos a la realidad. Las propiedades resistivas ayuda a elaborar estimaciones de consumo y desempeño energético. Los efectos capacitivos e inductivos permiten evaluar los fenómenos de retardo en las señales y consumo de potencia dinámico. Incluso, es posible considerar interferencia por radiación de campos eléctricos y en menor medida los magnéticos. Conocer el área que el enrutado ocupará también es importante para que la herramienta genere una buena optimización del colocado y el enrutamiento. El modelo de cableado es fundamental en sistemas de integración modernos, donde la escala de integración es cada vez más pequeña, a partir de los 300 nm es imperativo contemplar estos efectos, pues las dimensiones y propiedades físicas de los cables también generan pérdidas resistivas, retardos de propagación y efectos de interferencia por acoples capacitivos. Para más información consultar el capítulo 6 de [29].

- **Celdas de manejo:** aquí se define la celda que por defecto maneja la unidad bajo diseño (es decir, quién alimenta las entradas del circuito). El nombre de este atributo, es “driving_cell”, la traducción literal al español no es muy certera; sin embargo, es suficientemente clara. La herramienta necesita que este atributo se defina para calcular de manera eficiente el esfuerzo lógico necesario en las rutas a optimizar (es decir, el tamaño de compuertas a usar en determinada ruta, para minimizar el retardo en la misma). También permite que la herramienta seleccione celdas adecuadas para conectar los módulos del diseño a los puertos de entrada y salida. Esto debe hacerse, en las señales que se propagan hacia puertos de entrada y salida principales e intermodulares.

Otra aplicación de las celdas de manejo, consiste en definir un límite de carga para las señales. La carga excesiva generará problemas de retardo y, por tanto, fallas funcionales. No se entrará en detalles sobre porqué es importante respetar los *fanouts*² de las señales. Sin embargo, es evidente que contemplar las capacidades de conexión de una señal hacia múltiples dependencias, y garantizar la integridad de la información transmitida por dicha señal, es una práctica obligatoria para garantizar la operación exitosa de un diseño. Es por ello que es importante definir un valor máximo de *fanout*

- **Factor de Actividad:** el factor de actividad es un parámetro generalmente estadístico que le indica a la herramienta, como derivar el consumo de potencia dinámico del circuito total. En la literatura este parámetro se denomina con la letra griega alfa (α).

El factor de actividad, α , indica cuantas veces conmuta una señal por ciclo de reloj. En un diseño complejo habrán muchas señales con factores de actividad diversos y este cambiará de acuerdo con la tarea que se ejecuta. El valor adecuado a usar en este parámetro responde más a la evidencia empírica y a la experiencia del diseñador.[29]

El uso de este parámetro ofrece la posibilidad de generar un reporte de estimación de consumo energético más cercano al comportamiento real del diseño.

Existen muchos otros parámetros que permiten personalizar y condicionar de forma más profunda el diseño; no obstante, se expusieron las principales categorías que se deben contemplar y algunas particularidades que se consideran importantes para generar modelos suficientemente cercanos a la realidad. La selección de parámetros suele ser hecha por los ingenieros que se definen los alcances y el desempeño esperado del diseño.

Una vez que las restricciones de diseño han sido definidas y propagadas, se procede a realizar la síntesis en sí. El comando `compile` realiza dicha síntesis. La compilación consiste en traducir el RTL en estructuras booleanas o lógicas, realizar un mapeo de esa estructura en función de la biblioteca de síntesis disponible, y efectuar las optimizaciones necesarias para cumplir con las restricciones de temporizado y área. En la figura 3.6 se observa una analogía gráfica de lo expuesto anteriormente.

Retomando la figura 3.4 recordamos que el bloque de compilación nuevamente es representado por un diamante de decisión. Este diamante no refiere a un evento programático, si no a una etapa en la que el diseñador deberá evaluar si la compilación del diseño es satisfactoria. En esencia la compilación se corrobora mediante la solicitud de reportes a la herramienta; en este punto las bibliotecas compiladas en el directorio *alib-52* y los alias creados en el script de inicialización son bastante útiles, ya que es posible iterar el procedimiento de compilación con mayor facilidad y rapidez.

²El fanout o abanico de salida hace referencia a la máxima cantidad de carga (conexiones) que una señal o puerto es capaz de manejar, hasta que por razones de potencia, esta se empieza a degradar y la señal alcanza niveles de tensión que pueden conducir a la metaestabilidad.

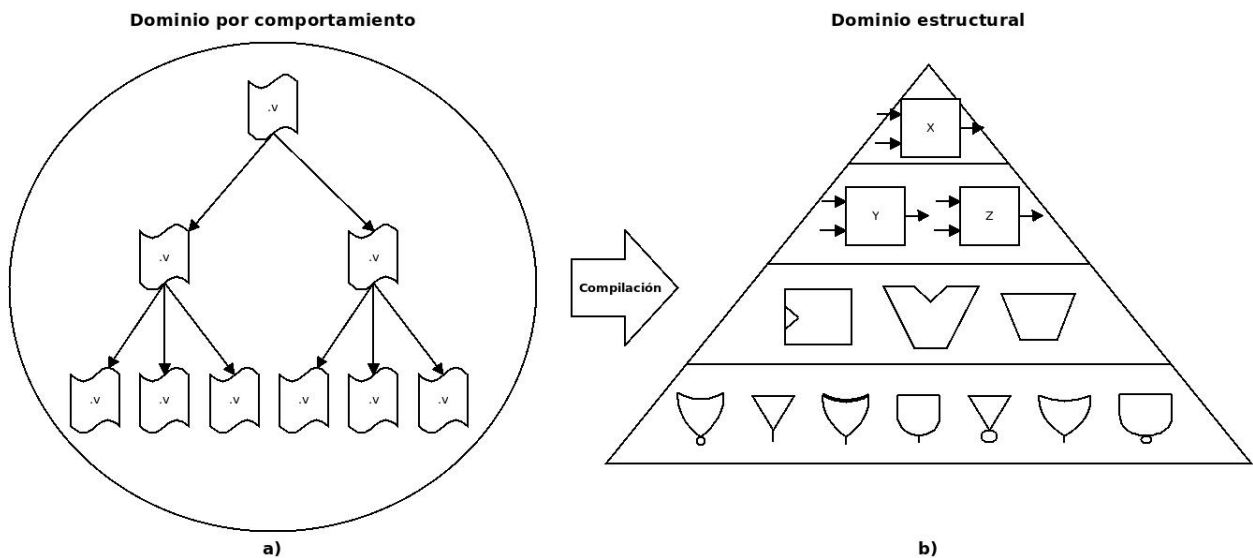


Figura 3.6: Ilustración del proceso de compilación de front end. En figura a) se aprecia la estructura jerárquica de códigos en HDL, y en la figura b) la jerarquía abstraída, implementada mediante distintos niveles de elementos digitales, pasando desde funciones complejas, módulos digitales avanzados, hasta compuertas lógicas fundamentales. Aunque no siempre se llega hasta estas últimas. Figura de autoría propia.

Es conveniente desviar los resultados de la síntesis (compilación) hacia archivos. Esto permite llevar un registro fácilmente accesible y que permita contrastar con otros diseños. En caso de necesitar incluir este trabajo en otro proyecto, podrá evaluarse si cumple las expectativas para el nuevo diseño.

La herramienta permite generar reportes de:

- **Temporizado (timing):** tabulación de la información asociada a la propagación de la señal de reloj del circuito. Es un análisis que estima el retardo en el diseño basándose únicamente en la topología de las rutas de propagación de la señal de reloj, y determinando la ruta crítica.

La herramienta genera entonces un presupuesto teórico de cuanto debe tomarle a la señal, el propagarse a través de la ruta crítica, y evaluar, junto con la información suministrada por las restricciones del diseñador, y la información que aporta la biblioteca de la tecnología, si los datos llegan en el tiempo presupuestado, conocido como “slack MET”³.

- **Potencia:** descripción resumida de las características de alimentación del diseño, referidas a la tensión nominal de operación general de los dispositivos, el comportamiento estático y dinámico de las celdas estándar en términos de disipación energética. La información aparece resumida y tabulada, de forma que se puede encontrar cuanta energía se disipa o consume en un bloque en particular.

³Es posible ahondar más sobre qué es el *slack* y cual es su relevancia; sin embargo, este informe pretende desvelar el flujo de diseño digital. Se parte de la premisa que el lector comprende el uso de términos como: *slack*, *hold time* o *setup time*, hablar más al respecto está fuera del objetivo de este informe.

- **Área:** resumen del volumen o cantidad de celdas usadas (combinacionales y secuenciales), y del área usada. Con este reporte es posible prever el espacio necesario para colocar estos elementos en el espacio del dado de silicio. La información más útil es que permite tener una idea del área necesaria para la colocación cuando deba diseñarse el plano de pizo, y le permite al diseñador efectuar un buen *ansatz*⁴ al definir el tamaño del plano de colocación y enrutado (floorplan) del diseño durante la implementación física.
- **Celdas:** listado de las celdas estándar necesarias para implementar el diseño. Se presenta en forma tabular, describiendo el nombre de referencia de la celda, el bloque de diseño donde es usada, el área que ocupa, y en que biblioteca donde se encuentra. Es útil para generar bibliotecas reducidas que permiten compilaciones más rápidas, y le permite al diseñador saber cuales bibliotecas debe tener en cuenta al hacer la implementación física o deba hacer simulaciones.
- **QOR (Quality of Results):** compendio de varios de los atributos que se exponen en los otros reportes. Tiene información sobre el área, la naturaleza de las celdas usadas, los caminos críticos de retardo, así como violaciones de sincronización, violaciones de reglas de diseño e inclusive presenta estadísticas del desempeño del computador que corrió la herramienta de síntesis. Se puede afirmar que es un resumen muy general del trabajo hecho en la síntesis del diseño.

Finalmente, es necesario guardar el proyecto y generar archivos que permitan evaluar si el proyecto cumple las expectativas de comportamiento (ejecuta las funciones para las que fue concebido), y generar los archivos que permiten continuar el flujo de diseño. En resumen de lo anterior, se deben generar los archivos que tienen la información del resultado del proceso que se acaba de realizar, además de generar los modelos para evaluar si los resultados siguen cumpliendo las expectativas de funcionamiento establecidas.

Los principales archivos generados son los siguientes:

- **DDC:** archivo jerárquico, que define la estructura del diseño, qué celdas son necesarias para implementarla, todas las dependencias entre éstas, además de información de conectividad para implementar el comportamiento abstraído del diseño HDL. Este archivo funciona como una base de datos que contiene toda la información expuesta en los reportes.

Este archivo es el estándar de almacenamiento de la información post síntesis lógica de la herramienta Design Compiler, y se usa si se necesita volver al diseño, usando el modo topográfico de la herramienta ya sea para editarlo y efectuar optimizaciones o recuperar información, como puede ser generar un reporte, etc. A partir de este archivo se puede continuar con la implementación física.

⁴Anzats: término de origen alemán, usado comúnmente por físicos y matemáticos para referirse a una estimación que permite resolver una ecuación o un problema.

- **HDL:** Este código sirve de modelo de simulación anotado post-síntesis, a la vez que ofrece de la jerarquía necesaria para la etapa de back end.

Verilog ofrece la posibilidad de implementar circuitos digitales mediante primitivas. Puede usar referencias a componentes digitales e interconectarlos de acuerdo con la semántica abstraída del código por comportamiento. A esta forma de codificación se le conoce como *Gate Level Description*: Descripción a Nivel de Compuertas (GLN).

Es conveniente generar 2 archivos con la información del GLD, esto con el fin de tener un archivo para iniciar la implementación física, y otro archivo para realizar la simulación post síntesis. La separación se debe a que el archivo usado para simular necesita de una directiva adicional que lee el archivo con la información de los retardos. Esta directiva entra en conflicto con el IC Compiler. En este proyecto se usa la nomenclatura `*_syn_sim.v` para diferenciar al archivo de simulación.

- **SDF:** siglas en inglés de Standar Delay Format (formato estándar de retardos). Consiste en un archivo, con información detallada de retardos y restricciones de temporizado, para uso de otras herramientas de colocado y enrutado o simuladores. En el caso de la simulación se obtienen resultados más cercanos a la realidad y este archivo permite una guía para que las herramientas de colocado y enrutado puedan optimizar el área en función de las restricciones de tiempo de las señales.
- **SDC:** siglas en inglés para Synopsys Design Constraints, que definen las restricciones aplicadas por parte de la herramienta según un estándar propio de Synopsys. Consiste en un compendio de las configuraciones de los retardos que definen el comportamiento de los puertos y de algunas señales.

Es posible exportar algunos archivos adicionales; sin embargo, los archivos anteriormente descritos son los necesarios para avanzar en el flujo.

3.2.2 Simulación Lógica: evaluación pre y post síntesis lógica

Los reportes generados permiten evaluar la calidad de la síntesis efectuada; sin embargo, para garantizar que el diseño funciona de manera adecuada, es necesario efectuar una verificación funcional. La verificación consiste en estimular el diseño de la misma manera que el modelo *RTL* o modelo por comportamiento.

En el directorio `testing` de la figura 3.2 se encuentran 3 directorios, dos de los cuales se asocian al proceso de *front end*, y corresponden a los directorios **behavioral** y **post_syn**. Todos los directorios contenidos en el directorio de testing tienen la misma estructura y archivos muy similares.

Los directorios de simulación contienen un script de precompilación y o simulación, y uno o dos archivos sin extensión que funcionan como punteros al invocar la herramienta de simulación. En el caso de una simulación por comportamiento (`behavioral`) únicamente se necesitan, los archivos RTL del diseño y el archivo de estímulo (banco o arreglo de pruebas), que son indexados con el archivo sin extensión denominado `source_list`. Entonces al ejecutar el script de simulación, se corre el comando de `bash` para invocar la herramienta. Este proceso se ilustra en la figura 3.7.

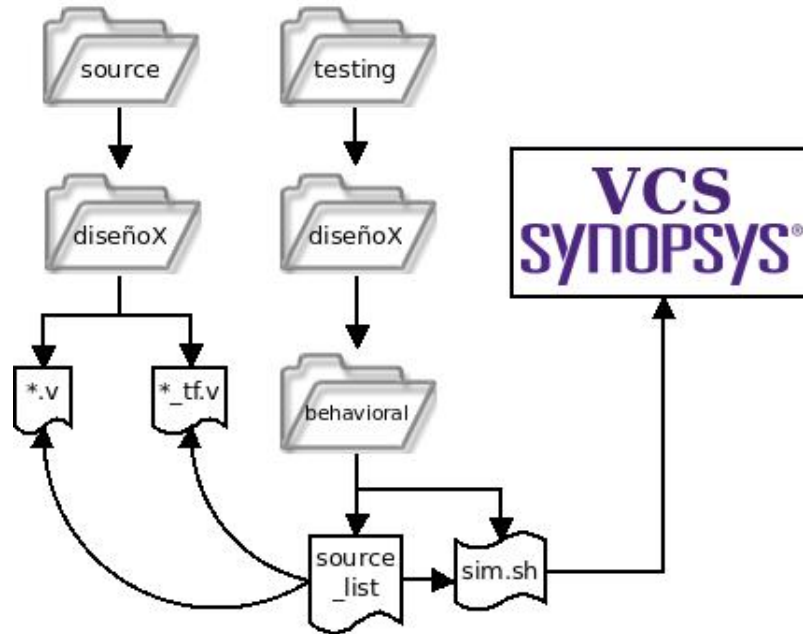


Figura 3.7: Estructura del script de simulación por comportamiento. El archivo `source_list` apunta hacia el RTL, que se ubica en algún subdirectorio de `source`. El script de simulación llama al archivo `source_list` al invocar al simulador VCS. Figura de autoría propia.

Para la simulación post síntesis se tiene una estructura similar, que posee dos archivos sin extensión. El primero le indica a la herramienta cuáles son los archivos necesarios para la precompilación (archivo de la tecnología que contiene la biblioteca de simulación, y el archivo HDL con el netlist generado en la síntesis lógica). El segundo le indica a la herramienta los archivos fuente (HDL con el diseño, y la biblioteca de simulación) y el `testbench`.

La simulación post síntesis requiere que el simulador genere un objeto de alto nivel con las celdas estándar y del comportamiento del retardo de las señales. Por ello que debe ejecutarse la precompilación y la lectura del archivo `*.sdf` con el fin de que en la simulación pueda contemplarse el efecto de los retardos de propagación de las señales. Se genera un nuevo archivo con los retardos `*.sdf_c` y este es el que será usado por el simulador. En la figura 3.8.

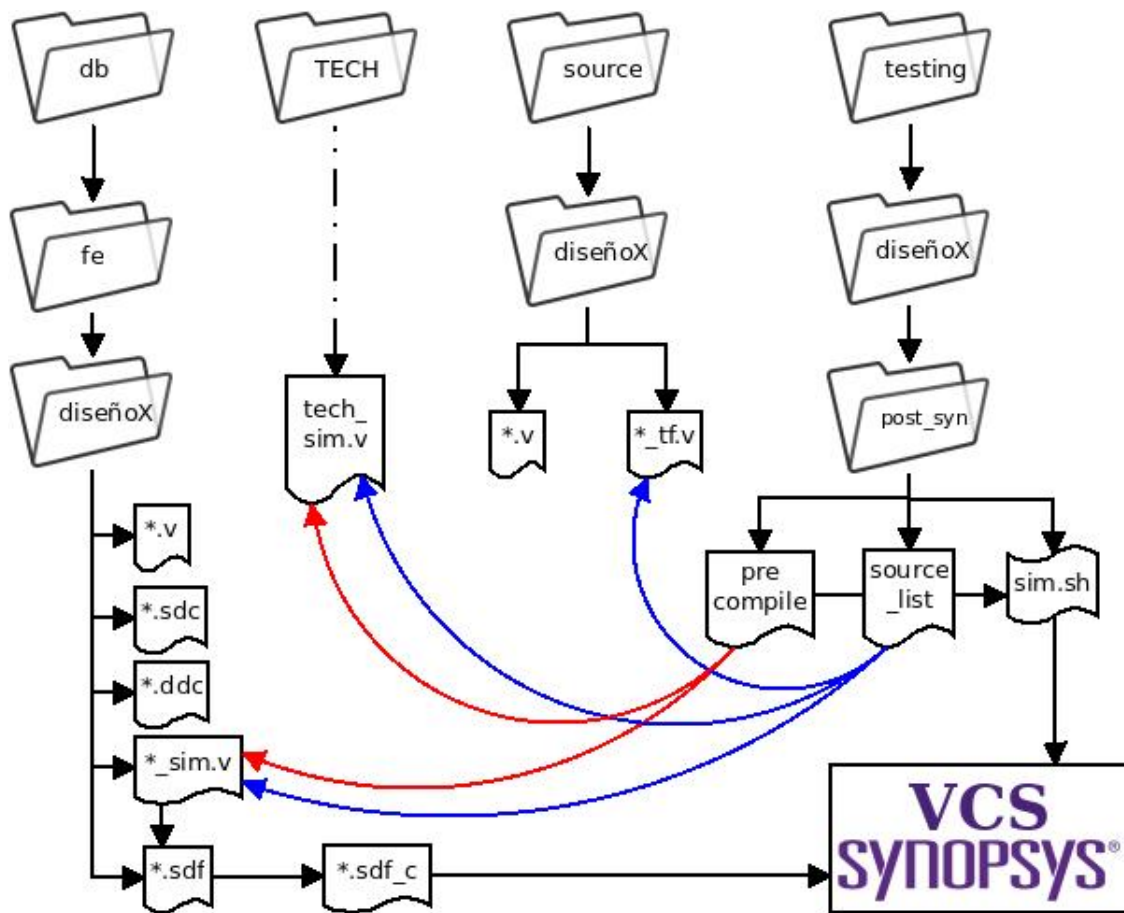


Figura 3.8: Estructura del script de simulación post síntesis. El archivo `precompile`, apunta a la biblioteca de simulación y al netlist. El archivo `source_list` apunta hacia el *netlist*, el código de estímulo, y la biblioteca de simulación. El script usado invoca al simulador, luego precompila y después llama al archivo `source_list` para realizar la simulación. Figura de autoría propia.

3.3 Scripts de diseño back end

3.3.1 Script de implementación o síntesis física

Las herramientas de Synopsys siguen un patrón similar en la ejecución de los flujos de síntesis. Nuevamente, conviene tener variables relativas hacia los directorios clave para que la implementación pueda ser ejecutada sin importar la ruta absoluta hacia los directorios. Además se deben definir las variables nativas de la herramienta que garantizan que el flujo será exitoso, tal como los punteros e identificadores hacia las bibliotecas.

Es posible iniciar la implementación de diversas maneras. En el caso del flujo propuesto en este trabajo comienza definiendo las variables principales para la celda física. En el entorno físico de Synopsys, la base de datos que contienen la información de la celda, o las celdas físicas se llaman Milkyway.

Antes de abrir o crear una base de datos, en esta biblioteca Milkyway se deben definir ciertos parámetros y aportarle a la herramienta información importante para la implementación. Esta información corresponde principalmente al nombre de las redes de alimentación y los coeficientes de tiempo de la tecnología. Al hablar de coeficientes de tiempo, se refiere a los retardos por efectos parasíticos, que en esencia son los efectos capacitivos y resistivos de los puertos de los transistores y por lo tanto los retardos de los puertos entrada y salida de cada celda estándar en la tecnología.

Los archivos que contienen la información de los coeficientes se llaman TLUPlus y su formato consiste en una base de datos binarios tabulados. Esta información es fundamental en la extracción de los efectos de retardo debido a las parasitancias propias de la tecnología CMOS. Si estos no son especificados no se podrá habilitar la extracción de parásitas RC y tener estimaciones de retardo, y consumo dinámico más precisas, necesarias para un análisis más exhaustivo con las herramientas Primetime y HSpice.

Una vez se hayan definido los archivos TLUPlus y demás parámetros para la celda Milkyway, se puede abrir o crear la biblioteca. En este punto es posible tomar diversos caminos para resolver un diseño. Idealmente es posible generar un único diseño jerárquico donde existe una única base de datos, y los subdiseños pueden ser implementados como celdas dentro de la base de datos general del proyecto. Así, es posible resolver el diseño desde una estrategia que en la jerga del diseño de software se conoce como *BottomUp*”, es decir, de abajo hacia arriba.

Existe otra posibilidad en la que se pueden generar bibliotecas Milkyway para cada subdiseño del proyecto principal. Estos subdiseños pueden exportarse luego como macros para diseños posteriores, con la información de dichos macros anotada en archivos del tipo llamado DEF (formato estándar de intercambio). Aún está pendiente el comprender exactamente cómo usar esta metodología de creación de macros (típicamente llamados hard macros).

En proyectos complejos, lo más probable es que no sea posible terminar un diseño en un único intento, es por ello que el `script` que se muestra en la figura 3.10, tiene un bloque condicional el cual se encarga de buscar el archivo Milkyway que contiene la información de la celda o el diseño que interesa trabajar. Este condicional evalúa si la base de datos (`.mw`) existe. Con esa premisa se toma la decisión de abrir la base de datos o en su defecto crearla.

En el caso de que la base de datos exista y se deba abrir alguna celda en particular, conviene dejar el `script` principal o de inicialización del diseño hasta aquí. A partir de este punto el diseñador tiene múltiples caminos para decidir que hacer con su diseño y no hay una única ruta posible. En este punto es posible invocar múltiples para realizar tareas como evaluación de reglas DRC, análisis de sincronía de señales, re-enrutado selectivo, re-estructuración del colocado de las celdas, optimización de enrutado de acuerdo con criterios de área, potencia, o sincronía, entre muchas otras opciones.

Siguiendo el flujo del diagrama, y asumiendo que se trabaja con un proyecto nuevo, lo que procede es ejecutar el script que importa el diseño. Hay dos maneras principales para cargar un diseño, leyendo un archivo "GL" (HDL a nivel de compuertas) o leyendo un archivo "*.ddc", ambos le indican a la herramienta que mediante la base de datos de las celdas estándar de la tecnología se importen las celdas físicas necesarias para iniciar el trazado físico.

La importación del diseño suele involucrar los siguientes procesos: lectura del archivo fuente, vinculación que resuelve las referencias del diseño y establece la jerarquía de los módulos, y remueve las celdas multiplicativas instanciadas de la biblioteca jerárquica. Lo anterior significa que se genera un nuevo bloque jerárquico que abstrae todas las jerarquías de las instancias y las unifica en una celda.

Plano de Piso (floorplan)

Luego de invocar las celdas físicas, la herramienta provee una nueva ventana para visualizar la evolución del trazado del diseño. Se procede a definir el floorplan o plano de piso, que corresponde a las dimensiones especificadas del área disponible de silicio. Existen varias formas para establecer el plano de piso, que siempre será un rectángulo, ya que es la geometría que permite maximizar el aprovechamiento del área al usar celdas con formas de cuadriláteros.

Valiéndose del reporte de área generado en la síntesis lógica y mediante un par de ecuaciones cuadráticas simples, es posible extrapolar las dimensiones del plano de piso. Habrá que considerar que tan simétrico conviene hacer el espacio o si otra relación de aspecto permite una mejor distribución de las celdas y un diseño más compacto. El diseñador deberá contemplar un presupuesto extra en las dimensiones para el enrutado general, y garantizar un espaciado suficiente para evitar problemas de congestión y demás DRCs. Además debe contemplarse un margen para puertos de entrada/salida (I/O) y los anillos de alimentación de los cuales se hablará más adelante.

Generar el plano de piso de forma manual y correcta responde más a la experiencia y la habilidad del diseñador para realizar un buen *ansatz*. Típicamente se deben hacer unas cuantas iteraciones para encontrar el mejor tamaño y relación de aspecto del plano de piso. La herramienta permite usar otras metodología para establecer el plano de piso. La forma más cómoda que ofrece la herramienta es proveer la relación de aspecto, de los márgenes desde los puertos I/O hacia el área de colocado, y finalmente el porcentaje del área utilizable para el colocamiento de las celdas; entre más bajo sea este último valor, menos probable será enfrentar problemas de congestión y por lo tanto más fácil conseguir un enrutado completo y funcional, pero esto implica que se desaprovecha un porcentaje considerable de área.

Una vez que se haya definido el plano de piso se procede a colocar la celdas y a legalizar el colocado, que corresponde a que la herramienta efectúe un análisis de posibles conflictos debido a la ubicación de las celdas. Así se reubican las celdas estándar del diseño, para evitar cualquier conflicto de congestión que se pueda encontrar debido a la colocación de las celdas.

Si se establece un diseño jerárquico, y ya se ha logrado conseguir la síntesis física de los subdiseños que componen el proyecto actual, es posible cargar la información de estas macroceldas ya generadas y tenerlas como premisa para el colocado y enrutado de la nueva macrocelda. Esta estrategia se logra al crear `plan_groups`.

Los `plan_groups` (grupos de planos) son áreas restringidas para que las celdas asociadas a una macrocelda se coloquen en una región específica; la colocación de las celdas se hacen a groso modo. Si existen celdas o diseños ya implementados la herramienta puede abstraer el colocamiento que se definió previamente para ese subdiseño y usarlo como punto de partida para agilizar el colocamiento del nuevo diseño. Suele complementarse con la lectura de archivos de restricciones que definen el colocado usado para los subdiseños.

Plan de potencia (power plan)

Una vez que se define la ubicación de las celdas estándar del diseño, se procede a generar el esquema de distribución de la red de alimentación del circuito. Existen para ello varias metodologías, exponerlas todas, pero se expone sólo una por razones del tamaño para este documento.

La metodología de distribución de potencia escogida corresponde a un arreglo tipo malla, que define anillos para las redes de alimentación. Mediante un arreglo simétrico de rieles paralelos vertical y horizontalmente, se distribuyen las señales de alimentación en toda el área del dado. Definir una distribución adecuada de la malla de potencia es laborioso e iterativo. Nuevamente depende de que tan bueno sea un ingeniero realizando *ansatz*. Posteriormente cuando se realice la extracción y simulación eléctrica, se desvelará si la distribución de potencia cumple con el presupuesto de potencia establecido.

Una distribución de potencia puede diseñarse de forma minuciosa considerando ecuaciones asociadas a la caída de tensión por pérdidas resistivas en el alambrado, o por fenómenos como la electromigración. Ello representa una tarea ardua y laboriosa a nivel matemático, y usualmente se deberá recurrir a los manuales de descripción física del proceso de integración. Además el diseñador deberá tener conocimientos profundos en el área de electromagnetismo a microescala.

Tener un departamento consagrado a la caracterización y diseño para la distribución de potencia en un chip, es propio de una metodología de diseño *full-custom*. Para un diseño *semicustom* carece de sentido realizar un proceso tan laborioso. Típicamente se usan las herramientas de estimación de alambrado en función de la caída en tensión debida a la impedancia del conductor (abreviada en inglés como **IRDrop**). esta abreviación se asocia a las pérdidas de potencia, debido a la disipación resistiva del conductor y electromigración principalmente.

En tecnologías relativamente antiguas (32nm y superiores), que ya han sido altamente depuradas, suele ser recomendable usar las aplicaciones para acelerar el diseño, provista en las herramientas de síntesis. En este caso la aplicación realiza un análisis de la potencia consumida por cada instancia, usando el mismo motor que se usa para generar los reportes de potencia. De acuerdo a las restricciones de diseño definidas en la síntesis lógica, como el factor de actividad o la probabilidad de conmutación, se define un valor estimado del consumo del diseño actual.

Con el valor de potencia necesaria, que ha sido estimado por la herramienta, algunas especificaciones dadas por el usuario, y la información que la biblioteca de la tecnología, la herramienta elabora una propuesta para distribuir las redes de alimentación en el dado.

Dentro de las especificaciones, el usuario define la tensión global del diseño, identificar los metales que serán usados en la red de alimentación, además de sus respectivas propiedades (ancho máximo y mínimo) y define la cantidad máxima y mínima de rieles y/o correas. Cabe destacar que el diseñador deberá proveerle a la herramienta algún medio de conexión a puertos virtuales “ V_{DD} ” y “ V_{SS} ”. Sin ellos la herramienta no ejecutará adecuadamente el acelerador de diseño.

La herramienta construye a partir de lo anterior el arreglo tipo malla (un ejemplo puede verse en la figura 3.9). Esta propuesta le permite al diseñador evaluar varias circunstancias relevantes para decidir como proceder a continuación. Por ejemplo, identificar un área donde se concentra un consumo excesivo, podría implicar una reestructuración del colocamiento de las celdas estándar, y así evitar posibles focos calóricos⁵ que puedan comprometer el desempeño del chip una vez fabricado. Las bibliotecas de las tecnologías profesionales manejan condiciones de esquina, que modelan condiciones del funcionamiento de las celdas a diferentes temperaturas, no se profundizará en este tema; sin embargo, hay múltiples fenómenos físicos que deben ser considerados. La herramienta le facilita al diseñador, tomar de decisiones de acuerdo con los escenarios que definen las condiciones de esquina y las especificaciones definidas para el diseño.

Una vez que el diseñador se siente satisfecho con el plan de potencia, se encomienda a la herramienta su implementación, y posteriormente se deben remover los *pads* virtuales que fueron usados para crear el plan de potencia. Conviene entonces pre-enrutar la alimentación de las celdas estándar, y generar *pads* virtuales para efectuar el análisis de potencia a posteriori. La creación e implementación de la distribución de potencia del chip es iterativa y aunque la herramienta facilita su desarrollo, es posible que cuando se efectuó un análisis eléctrico, se encuentren deficiencias en la misma, y consecuentemente deba reformarse el plan de potencia.

⁵En la termodinámica se define un foco calórico como un ente que es capaz de intercambiar calor, idealmente, sin que su temperatura se vea afectada, que posee una disipación térmica continua o constante [33]. En el contexto de este proyecto de graduación, si un chip presenta una temperatura muy elevada en un sector, podría causar daños a los elementos de la zona, y en casos muy extremos alterar el funcionamiento de los dispositivos, causando problemas como metaestabilidad. Ello pues debido a la naturaleza de los semiconductores, una variaciones en la temperatura causa variación en la movilidad de los portadores, y altera las tensiones de umbral de los dispositivos, lo que introduce cambios en las relaciones V-I de los transistores con respecto a las típicamente medidas o modeladas.

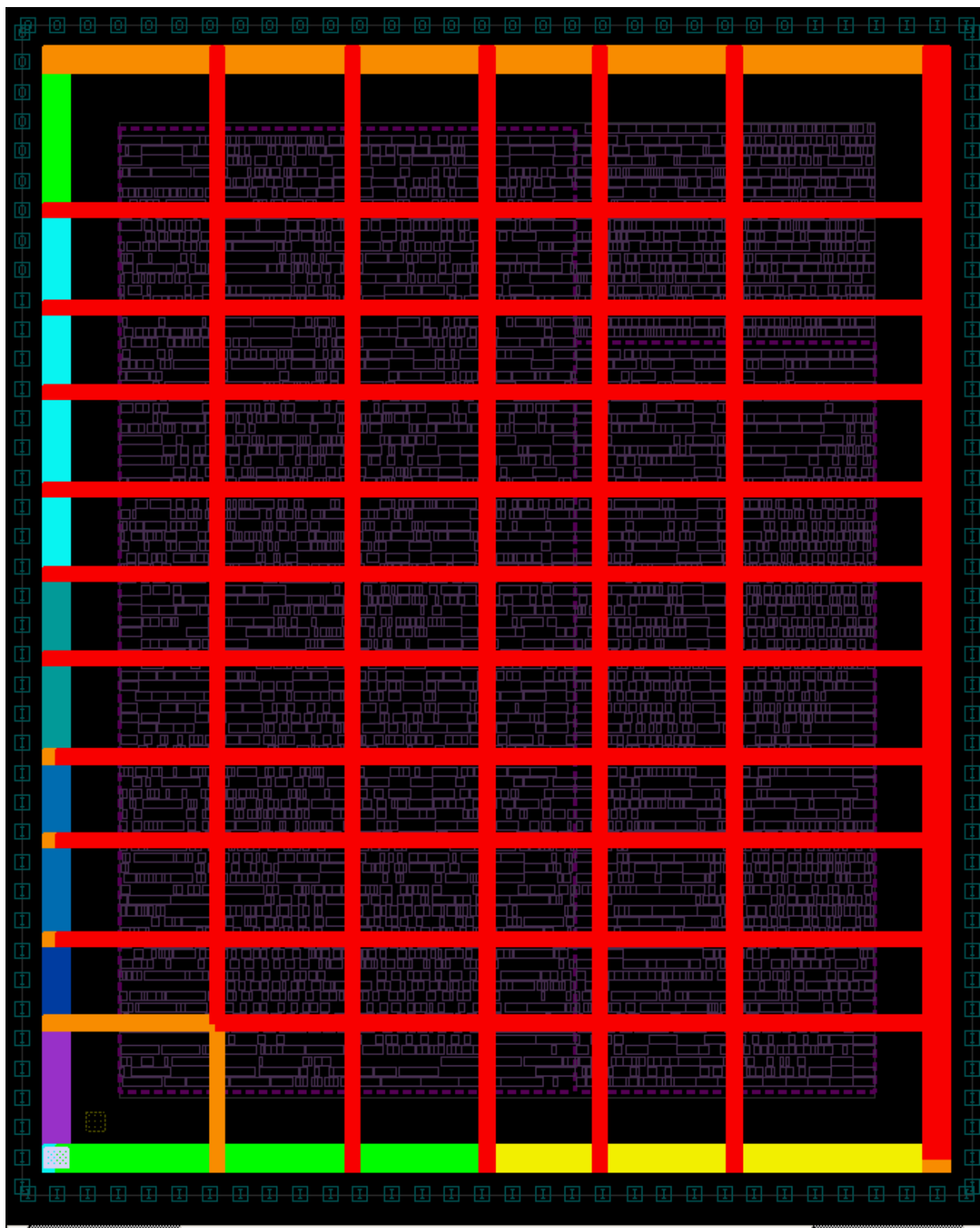


Figura 3.9: Primer propuesta de distribución de potencia para el trazado de la unidad aritmética de punto flotante discutida en el capítulo 4.

Enrutamiento y conexión global

Una vez fueron definidos e implementados el plano de piso y el plan de potencia faltaría conectar el resto de dispositivos para obtener el funcionamiento esperado del diseño. A este proceso se le conoce como enrutado (*Routing*). Antes de iniciar el enrutamiento deben definirse algunas restricciones de diseño importantes, como las restricciones sobre el árbol de reloj, las reglas de antena, y las restricciones sobre el enrutado global.

Árbol de reloj

Respecto al árbol de reloj deben considerarse los siguientes aspectos:

- Identificador del nombre del reloj.
- Inserción de celdas frontera cercanas a los puertos de reloj para el diseño basado en jerarquía de bloques.
- Atributo de apiñado de OCV (OCV Clustering). Este atributo permite mejorar la distribución del reloj a través de ramas de registros.
- Reglas para la reubicación y redimensionamiento de de las compuertas y búfers para optimizar la síntesis del árbol de reloj.

Muchos de estos atributos funcionan en conjunto con las restricciones definidas en el proceso de síntesis lógica, como lo son la incertidumbre, o la latencia del reloj. También el *netlist* generado es de importancia pues con base en éste se le encomienda a la herramienta implementar el árbol de reloj.

Reglas de antena

Las reglas de antena permiten especificar atributos relacionados con este fenómeno físico de fabricación, que se almacenan en la biblioteca Milkyway, y que permite la inserción de diodos entre el enrutado.

Las reglas de antena básicamente definen el área máxima de conexión a metal en la compuerta de un transistor. En los procesos de múltiples metales, los cables entre capas suelen acumular carga electrostática y esta acumulación de carga electrostática recibe el nombre de “antena de acumulación de carga” o simplemente “antena”.

El problema únicamente se presenta en la fabricación del chip cuando las conexiones entre capas están incompletas y no existen caminos de descarga disponibles a través de las terminales de fuente, y drenaje del transistor. Tiene una mayor probabilidad de presentarse cuando un cable con un área considerable se conecta a un transistor cuya compuerta tiene un área menor.

La herramienta previene los problemas de antena, verificando la entrada de cada celda⁶. Si se determina una relación de áreas propensa a un efecto de antena entonces la herramienta colocará un diodo de protección para desviar una posible descarga electrostática.

La definición de las reglas de antena está intrínsecamente relacionada al proceso de fabricación; la razón entre el área del cable y la compuerta la define el proveedor de la tecnología.

Para considerar inserción de diodos y evitar los efectos de antena se establecen los siguientes criterios:

- Definición de especificaciones particulares para cada capa de metal y uso de área poligonal.
- Protección de diodo limitada, esto considerando que si más de un diodo es necesario para conectar a una antena, se colocará un diodo que abarque el efecto total de todas las relaciones de área asociadas a la razón de área entre esa antena y las compuertas conectadas a ella. Es decir se colocará un único diodo capaz de manejar el peor escenario entre la antena y todas la compuertas asociadas a ella.
- Razón de metal: especifica la máxima relación permitida entre las áreas del metal y la compuerta. Esto puede considerarse como una condición para considerar si la relación entre un metal y una compuerta amerita colocar un diodo; relaciones excesivas indican un enrutado deficiente y por lo tanto deberá iterarse en este proceso.
- Razón o tasa de corte: especifica la máxima relación permitida entre las vías entre capas (metales) y las compuertas de los transistores. Al igual que el parámetro anterior permite evaluar la calidad del enrutado.
- Razón o tasa de diodo: se relaciona con la protección de diodo limitada; aquí se definen las posibles relaciones entre una antena y todas las compuertas asociadas a ella.

Los atributos de las reglas de antena los establece el fabricante y la mayoría de los atributos pueden encontrarse en el manual de la tecnología. Los atributos presentados, responden a la información que aporta el manual y al criterio técnico del Dr. Rodríguez.

Enrutado global

La configuración de la herramienta de enrutado (Zroute) de ICC-Compiler se divide en cuatro categorías, configuraciones comunes, globales, detalladas y de rieles.

En las configuraciones comunes tenemos: obedecer las restricciones de enrutado, que son propias de los planes de grupo definidos en la etapa de plano de piso, y que en el enrutado global sean respetadas.

⁶En esencia la herramienta comprueba las conexiones entre el metal de los cables y las compuertas de los transistores de entrada de las celdas.

Entre las configuraciones detalladas, tenemos la inclusión de las reglas de antena, la referencia a los diodos que se utilizarán, y que estos sean considerados durante el enrutado, así también como configuraciones por defecto del tamaño de las compuertas (de los transistores), la razón por defecto de la protección de los diodos y si debe emplearse un esfuerzo alto para que el enrutado sea converja con las reglas de diseño.

Dentro de las configuraciones globales se establecen condiciones que afectan a todos los comandos que realizan el enrutamiento global. Las configuraciones principales corresponden a enfocar el enrutado para cumplir las especificaciones de sincronía y usar un esfuerzo alto en ello. Finalmente se tienen las configuraciones de enrutamiento asociadas a los rieles ue básicamente usan dos métodos de optimización: o enfocado a la sincronización, o enfocado a disminuir la diafonía (crosstalk). En el caso de este proyecto, se prefirió la opción correspondiente a la sincronización.

Existe una configuración adicional que corresponde a la estrategia de optimización de los buffers; un esfuerzo medio a bajo se considera adecuado para el enrutamiento, ya que entre más alta sea configurada esta restricción, mayor será la optimización de los buffers, que podría afectar el *fanout* de las funciones. Un esfuerzo alto permite mejorar la sincronía, pero baja el *fanout*

De todos los procesos descritos hasta el momento, podría decirse que el enrutado es en el que los resultados se obtienen de manera más empírica. No obstante a que los algoritmos de enrutamiento que presenta la herramienta ICC son altamente eficientes, en diseños complejos la herramienta puede no ser capaz de implementar todas las interconexiones y cumplir a la vez con las expectativas de sincronización y reglas de diseño en una única iteración. Por ello el enrutado es el que más ciclos iterativos conlleva, debido al gran número de restricciones que debe cumplir.

Tener un enrutado que satisfaga todas las consideraciones de DRC, antena y de especificaciones propias del diseñador, no es fácil. No hay una secuencia definida para lograr conseguir un enrutado satisfactorio, y cada vez que se realiza un proceso completo de enrutado con miras a resolver algún problema puntual, el diseñador debe comprobar que todas las consideraciones del diseño se cumplen y son satisfactorias.

Cuando se tiene un enrutado satisfactorio, se procede a insertar las celdas de relleno y el metalizado de relleno (celdas sin metales). Las primeras rellenan los espacios vacíos en las filas de colocación de celdas, usando para ello celdas estándar de relleno. El metalizado de relleno elimina los espacios causados por una densidad insuficiente de metal en las capas de la interconexión. Para ello se usan cables de relleno, sin conexiones o conectados a tierra para alcanzar las reglas y especificaciones.

Finalmente, deben exportarse los reportes y generarse los archivos finales de salida. Respecto a los reportes, se genera información congruente con lo expuesto en la sección anterior sobre la síntesis lógica (sección 3.2.1).

Los archivos de salida generados son similares a los de la síntesis lógica: un archivo HDL (para simulaciones) que contiene el diseño a nivel de red de compuertas (GLN) implementado, y un archivo con extensión `*.sdf` con la información detallada de retardos tanto de cableado como de las compuertas, basándose en la información física final de enrutado y colocación, que es más preciso al generado en la síntesis lógica, que usaba estimaciones aproximadas.

Un tercer archivo, de extensión `*.spef`, ofrece información de parásitas RC más detallada, que permite a la herramientas como PrimeTime realizar análisis de temporizado más exactos, basados en los modelos de retardo de Elmore [29]. Para lo anterior es necesario haber definido los archivos TLUPlus que se mencionaron anteriormente (inicio de esta sección).

Es posible generar archivos `*.spef` para condiciones mínimas, máximas, y de esquina (corners)⁷ relacionadas con los mejores y peores casos de retardo intrínseco de los transistores y celdas del proceso.

Los últimos archivos de salida relevantes para el flujo corresponde al archivo `.def` y el **GDS**, que consisten en compendios de la información física del diseño, incluyendo información de trazado, *netlist* y restricciones de diseño. La herramienta permite exportar un archivo con toda o alguna de la información del diseño; por ejemplo, podría generarse un archivo `.def` únicamente con la información de las vías. El archivo **GDS** es corresponde al formato estándar con el que los que se intercambia información con el fabricante. Este archivo contiene toda la información necesaria para el desarrollo de las máscaras del proceso de diseño, así también como detalles particulares del proceso de fabricación. En este proyecto no fue necesario generar este archivo ya que el trabajo de integración no está completo. La figura 3.10 resume lo expuesto en este apartado mediante un diagrama de flujo.

3.3.2 Script de simulación física: evaluación post implementación

Al igual que en el proceso de síntesis, es posible determinar la calidad de la implementación mediante el análisis de los reportes, y aunque hacer esto es importante, es necesario validar la funcionalidad del circuito mediante simulaciones de post colocación y ruteo..

El proceso no se detallará pues sigue la misma estructura que lo expuesto para la simulación post síntesis de la sección 3.2.2 y las diferencias fundamentales consisten en los punteros hacia los archivos fuente para la simulación. Lo anterior puede apreciarse claramente en la figura 3.11.

3.3.3 Script de análisis de temporizado estático STA

En esta sección se expone como se utiliza la herramienta PrimeTime para evaluar si el diseño cumple las expectativas de temporizado de señales a la velocidad establecida en el diseño.

⁷Siempre y cuando se hayan incluido las bibliotecas correspondientes.

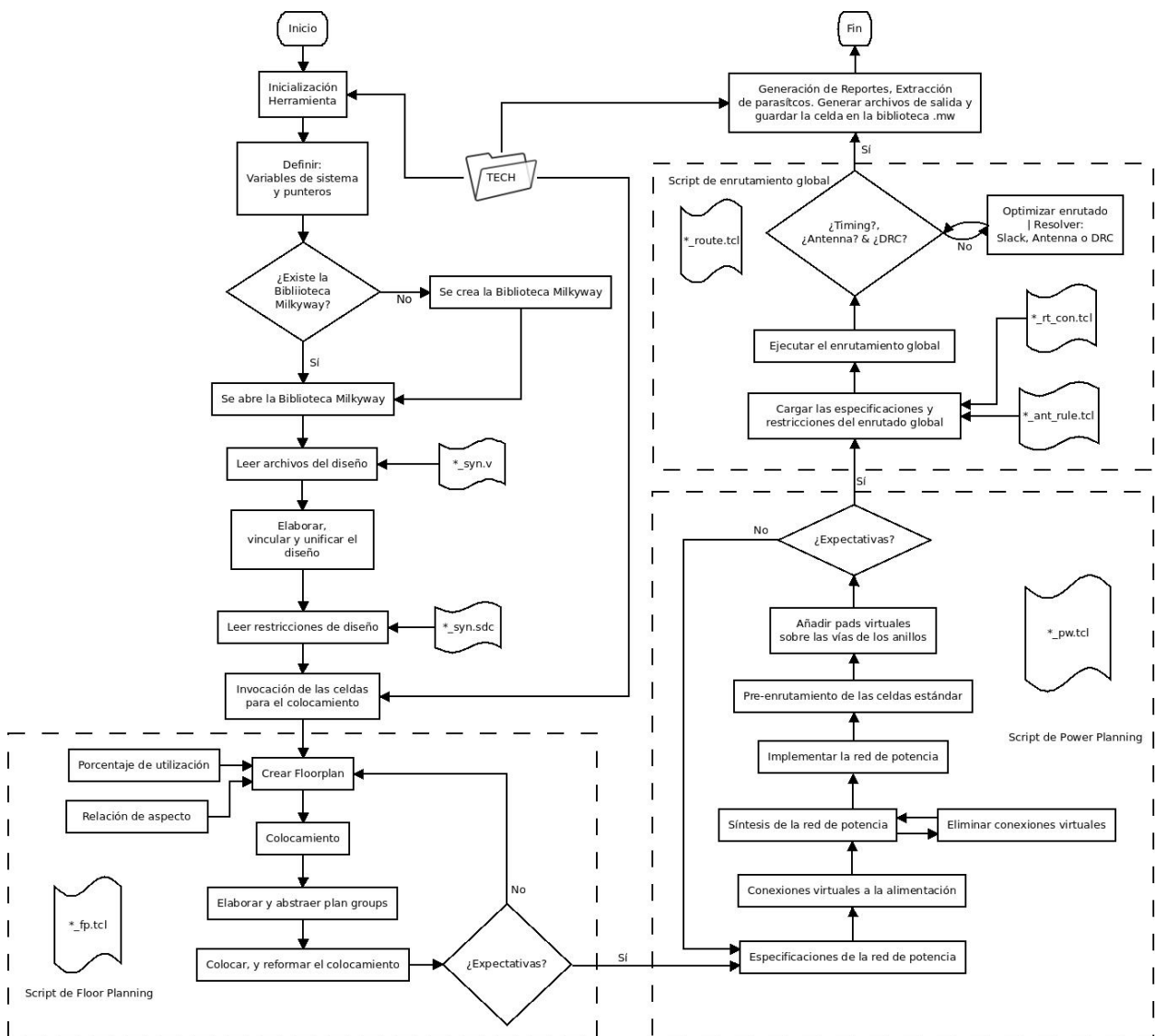


Figura 3.10: Diagrama de flujo que expone el proceso genérico de la implementación física en la herramienta *IC Compiler*. *Figura de autoría propia.*

PrimeTime es una herramienta que se usa tanto para evaluar un diseño post síntesis lógica como uno post implementación física. Prime Time puede considerarse una herramienta de back end, pues sirve para verificar el cumplimiento de las especificaciones de temporizado luego del traslado del diseño a las herramientas físicas. No obstante, sus resultados pueden utilizarse para hacer iteraciones más precisas de síntesis lógica, cuando las restricciones de tiempo no se cumplen a cabalidad (a este proceso se le conoce en inglés como back annotation).

Como se puede apreciar en la figura 3.12, el flujo de operación de esta herramienta es bastante simple, para la misma se ha desarrollado un pequeño script, que inicializa la herramienta y define los punteros hacia los archivos fuente y la biblioteca de la tecnología.

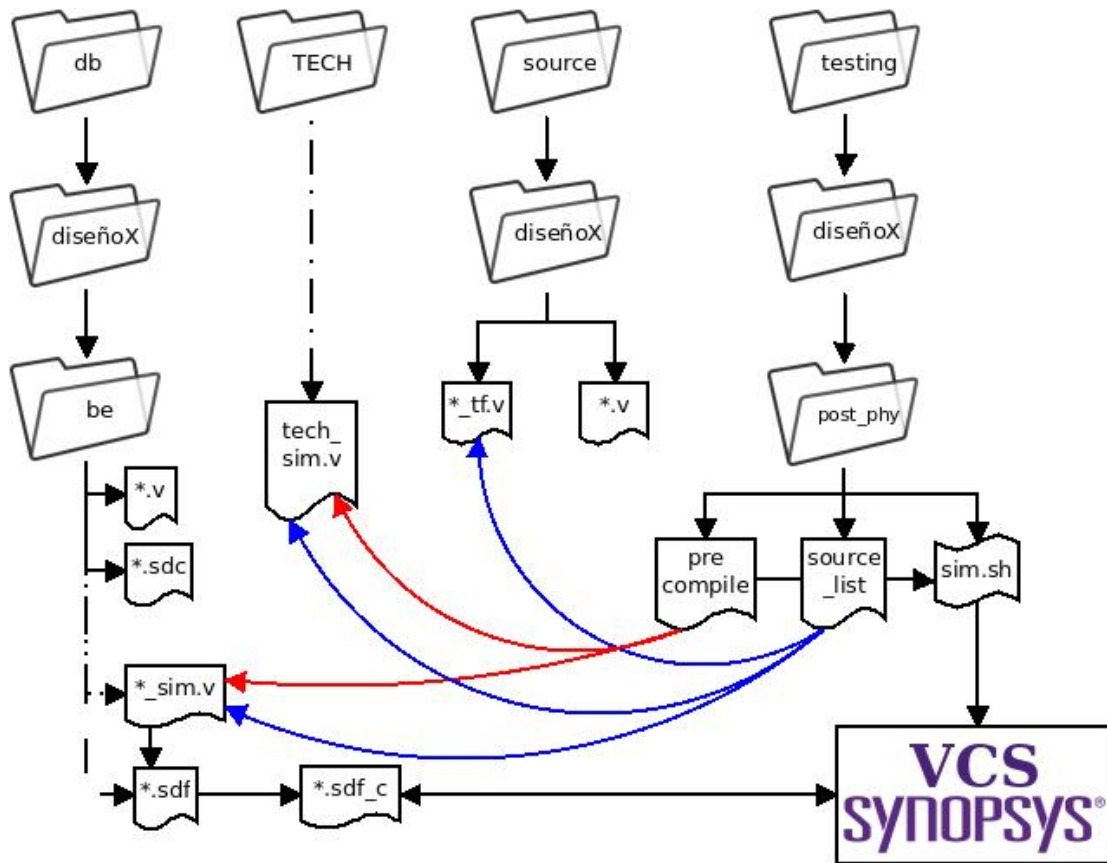


Figura 3.11: Estructura del script de bash para la simulación post implementación. Figura de autoría propia.

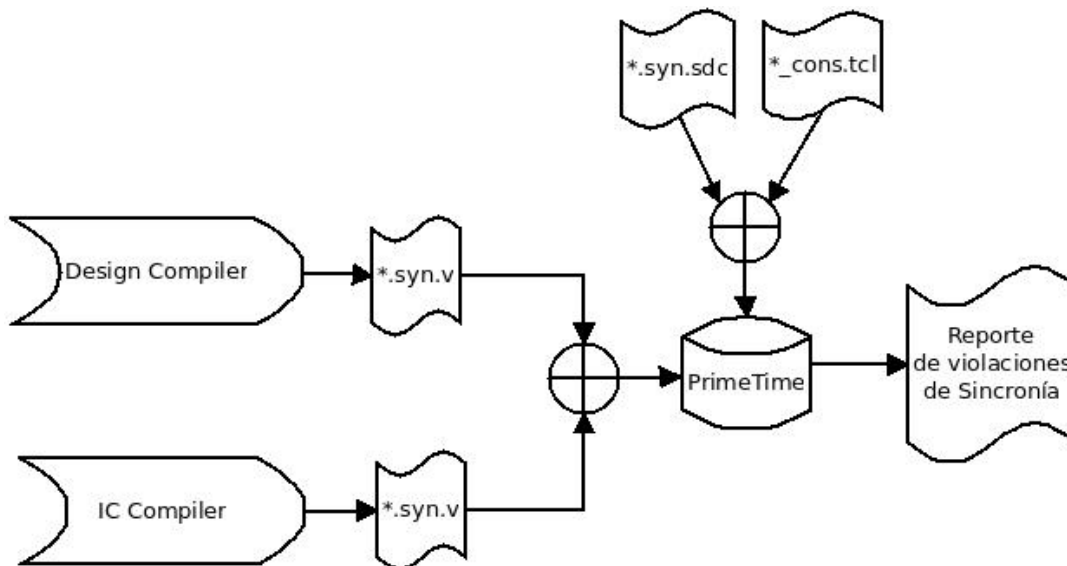


Figura 3.12: Diagrama de flujo del análisis de temporizado estático (STA). Figura de autoría propia.

Luego de inicializar la herramienta, se cargan los archivos fuente que consisten en: el archivo verilog con el GLN implementado, el archivo SDF con la información de los retardos de propagación a través del GLN, el archivo SDC con las restricciones de diseño de Synopsys (principalmente las de temporizado), y finalmente el archivo SPEF para incluir el efecto

de las parasitancias en el análisis. También se pueden incluir otros archivos para realizar otros tipos de análisis como son archivos SAIF, que indican el factor de actividad de las celdas. De igual manera se puede estimular el diseño con otro conjunto de restricciones, que corresponden con la misma sintaxis usada en el script de TCL para definir las restricciones en la síntesis lógica (`*_cons.tcl`).

Debido a que los diseños grandes y complejos presentan una enorme cantidad de restricciones, es necesario efectuar análisis multimodales y multiesquina⁸. PrimeTime permite generar reportes enfocados a escenarios particulares de temporizado. Ya sea para evaluar si el diseño cumple con las restricciones de diseño especificadas en una esquina o esquinas determinadas, PrimeTime también permite hacer un análisis gráfico, y generar recomendaciones para alcanzar las restricciones de sincronización definidas.

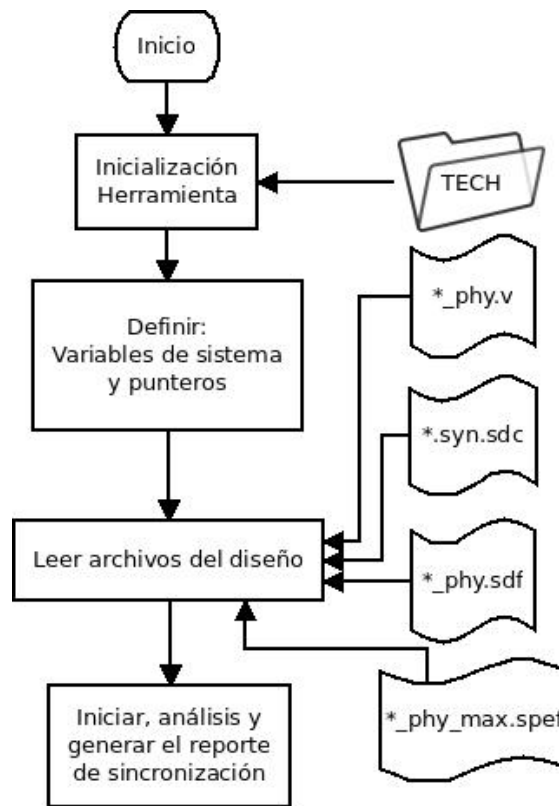


Figura 3.13: Diagrama de flujo del script para un análisis básico de STA en la herramienta PrimeTime de Synopsys. Figura de autoría propia.

El flujo de análisis expuesto en la figura 3.13, expone un uso relativamente simple de la herramienta donde se genera un análisis sobre los tres criterios más relevantes: retardos de entradas a registros, retardos entre registros y retardos de registros a salidas. Para los diseños trabajados en este proyecto, se considera una única condición de esquina, la cual corresponde a los valores típicos de retraso en las señales, una de tensión de 1.2 V para las celdas estándar de la tecnología y a temperatura ambiente (25°C). Estas condiciones se definen al inicializar la herramienta y cargar la biblioteca de la tecnología.

⁸Multiesquina: Traducción literal de multicorner para referirse a que se evaluarán todas las condiciones esquina individualmente

Los reportes generados usan motores de cálculo de temporizado similares a los de las herramientas de síntesis; sin embargo, son más robustos en términos de profundidad y variedad en el análisis. Herramientas como Design Compiler y IC Compiler usan un modelado dinámico del diseño, valiéndose del peor escenario de propagación de señales, determinado mediante el modelo de Elmore. PrimeTime permite usar modelos estáticos y es capaz de analizar distintos escenarios y estimular el diseño de diversas formas, para así evaluar el desempeño de la propagación de señales en distintos bloques de lógica: puramente combinacional, registros, puertos y/o distintas combinaciones de todos ellos.

3.4 Script de síntesis de IP Cores y bibliotecas autónomas

Este apartado corresponde a una particularidad en el flujo de diseño digital propuesto, ya que no todos los diseños requieren incluir IP Cores⁹. La inclusión de los programas necesarios para su inclusión, obliga a variar la regularidad de la estructura de directorios definida anteriormente (ver inicio del capítulo 3).

El flujo del diseño de los IP Cores usados en este proyecto requirió de tres herramientas extra. La primera y principal corresponde al sintetizador de memorias de la tecnología usada: Artisan ARM Physical IP. Esta herramienta es un generador de IP Cores para celdas de memorias (SRAM y ROM) sobre el proceso IBM CMRF8SF-RVT. La herramienta puede invocarse a través de scripts, aunque también cuenta con una interfaz gráfica bastante amigable.

Las dos herramientas extra mencionadas pertenecen a Synopsys, y son el Library_Compiler y Milkyway. El primero es necesario para definir una biblioteca lógica dedicada a los IP Cores de forma que el bloque SRAM generado se pueda instanciar en el diseño de manera que sea visto como un bloque duro *hardblock* en los procesos de síntesis. *Milkyway* sirve como complemento para generar las bibliotecas físicas de las celdas.

No se expondrá mucho sobre el Library_Compiler, pues su uso es muy simple, y muy puntual. Esta herramienta es necesaria ya que al ejecutar el generador de memorias, el resultado de la síntesis de un IP Core son los archivos:

- Un modelo Verilog GLN para instanciar la celda generada. Algo equivalente a lo que se obtendría efectuando una síntesis lógica en *DesignCompiler*.
- Archivo *.dat, que es una tabla *ASCII* con un resumen de la información geométrica y eléctrica de la celda.
- Un archivo tipo PostScript, que es una pequeña hoja de datos con diagramas de sincronía e información sobre los puertos en formato “.ps”.
- Archivo *.vclef, que contiene la información de la distribución de pads, diodos y metales para el enrutado (interconexión) y conexión externa en la celda.

⁹Un IP Core, o bloque de propiedad intelectual consiste en un módulo que no ofrece información sobre como es su arquitectura interna. Su representación consiste en una caja negra.

- Archivo `*.clf`, necesario para las herramientas de *Cadence* que es otro proveedor de herramientas para diseño VLSI. Sin embargo, los *foundries* suelen solicitar este archivo por motivos de estandarización, así que conviene conservarlo. Contiene información relacionada con la antenas y los diodos de protección.
- Un archivo en formato Liberty (`*.lib`), que es un estándar para definir los parámetros de sincronía y potencia de la celda, así también como la conectividad con el exterior. Este archivo en conjunto al `*.vclef` permiten definir la celda física del IP Core.
- Un archivo `*.data`, que es necesario para el STA con PrimeTime; este archivo no obstante requiere precompilarse con un script de PERL para obtener el archivo `*.sdf` que necesita PrimeTime

A partir de estos archivos, Library Compiler traduce la biblioteca en formato liberty (`*.lib`) al formato de base de datos de Synopsys (`*.db`), con el fin de poder incluir la celda de memoria generada en el flujo de front end. Milkyway compila la información de la biblioteca de la tecnología, el archivo `*.vclef` y la biblioteca `*.db`, para así desarrollar una nueva biblioteca `*.mw` que permita incluir la celda de memoria en el flujo de back end.

Las instrucciones necesarias para compilar la biblioteca en formato *liberty* a una en formato *database* son las siguientes, observe que la palabra `library_name` indica que se debe colocar el nombre de la biblioteca que se definió al usar el generador de SRAMs.

```
1 read_lib library.lib
2 write_lib <library_name> -format db -output library.db
```

El script para ejecutar la herramienta Milkyway sigue un proceso similar al de la síntesis física. Primero se configuran los punteros y las bibliotecas de la tecnología. Luego se crea o se abre la biblioteca `*.mw`, para incluir las nuevas celdas. Posteriormente al inicializar la biblioteca Milkyway se leen los archivos `*.vclef`, aquí es importante destacar que la referencia un archivo particular de la tecnología (`lefin_layer_map.txt`) debe hacerse con cuidado. Este archivo contiene alias de los nombres de los metales de la tecnología y sin ellos es imposible generar de forma exitosa la biblioteca física. Los alias definidos en el archivo `lefin_layer_map.txt` deben ser congruentes con los nombres de los metales de la tecnología. De no ser así, deben ser editados. Este archivo debe incluirse ya que la herramienta de generación de SRAM nombra de forma diferente a los metales cuando genera los archivos `*.vclef`. Finalmente se leen los archivos `*.db` para actualizar la información de los puertos, en términos de la temporización de señales.

Como una consideración final, para mantener la regularidad, localidad y continuidad en los directorios y archivos, conviene tener una única biblioteca Milkyways para los IP Cores. Así cada nueva macro celda se abstraerá de una única biblioteca auxiliar.

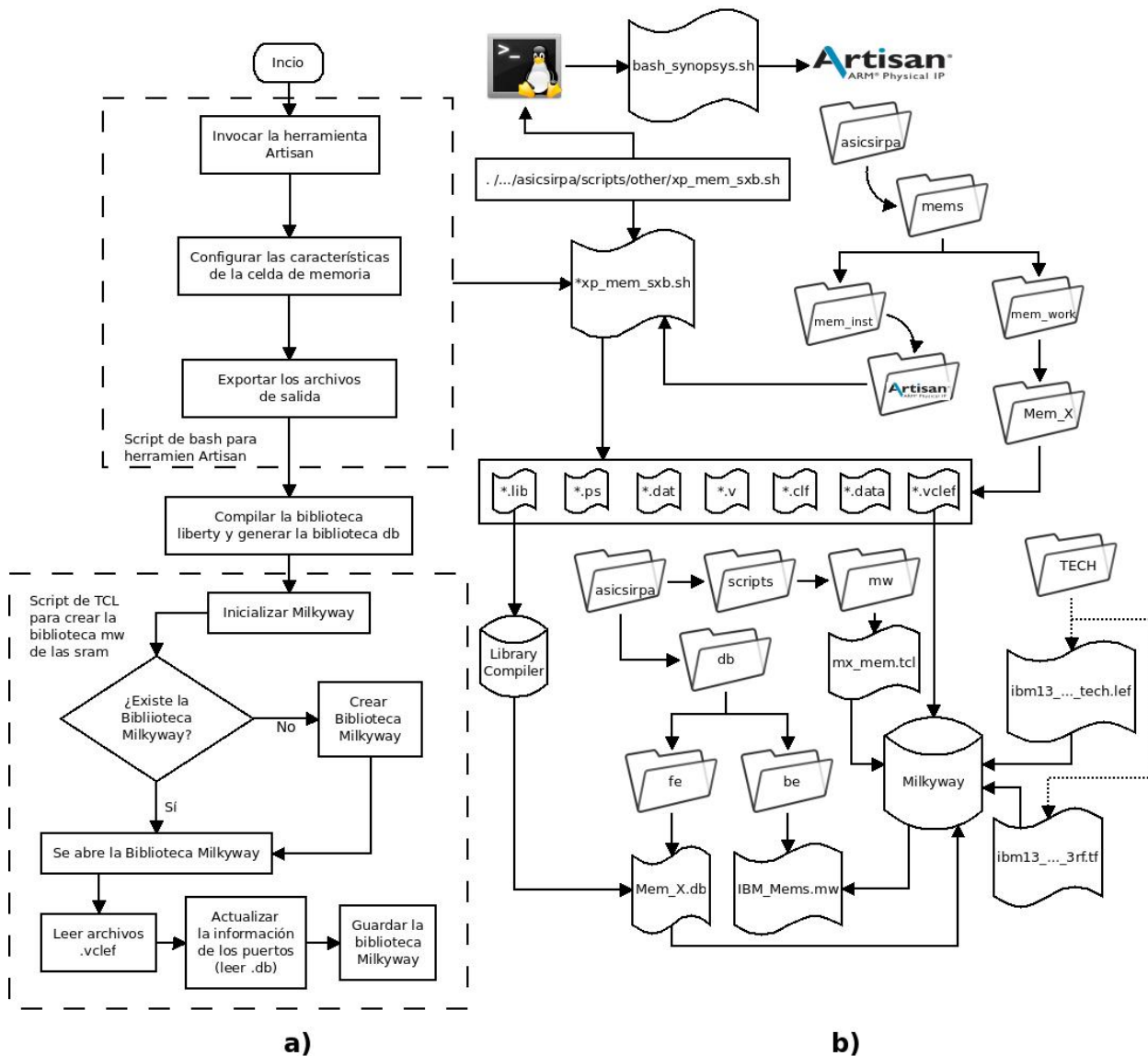


Figura 3.14: a) Diagrama de flujo de la síntesis de los *IP Cores* de memorias SRAM y generación de las bibliotecas autónomas.

b) Estructura de directorios y archivos que intervienen en la síntesis de los *IP Cores* de las memorias, herramientas involucradas y los archivos generados. Figuras de autoría propia.

3.5 Análisis de resultados: Scripting

Cabe destacar que determinar la eficiencia de una estructura de directorios y archivos que pretenden realizar una tarea programática particular, depende de los paradigmas de trabajo individuales o de un equipo de trabajo. Por lo tanto no se puede establecer de forma cuantitativa que tan eficiente es la estructura de scripting que se planteó en este capítulo. Es por ello que este análisis de resultados es cualitativo, a partir de la obtenido en la experimentación con las herramientas de Synopsys y la integración de un microprocesador RISC-V para el proyecto SiRPA.

Como se mencionó al inicio del capítulo, la forma en que los directorios y archivos fueron dispuestos responde a tres criterios: regularidad, localidad y continuidad. La estructura mostrada en la figura 3.2 presenta una alta regularidad pues la mayoría de directorios tienen la misma jerarquía, también continua, pues como puede observarse en las figuras 3.7, 3.8 y 3.11, es posible rastrear los productos de los distintos procesos que afectan a los archivos. Finalmente, cuando un archivo es generado no es necesario copiarlo en ningún otro directorio. Los procesos que lo necesiten a posteriori podrán acceder a él mediante un puntero, definible en los scripts de los procesos que lo requieran.

Los fundamentos de este capítulo se han extraído de los manuales de Synopsys, las páginas *man* de sus herramientas y la experiencia del Dr. Rodríguez.

Capítulo 4

Síntesis lógica de un microprocesador RISC-V

En este capítulo se explica el proceso seguido para la síntesis lógica del código RTL del microprocesador RISC-V de aplicación específica (ASP, del inglés Application Specific Processor), desarrollado por el M.Sc. Carlos Salazar [34, 35]. En esencia la tarea encomendada consistía en efectuar la síntesis lógica del microprocesador y comprobar mediante simulaciones post síntesis lógica su funcionabilidad.

Este ASP diseñado por el M.Sc Salazar, tiene como propósito ejecutar un algoritmo de modelos ocultos de Markov (HMM), para la detección de patrones acústicos de disparos y motosierras dentro de ambientes boscosos. Explicar este algoritmo está más allá del objetivo de este trabajo, por lo que se invita al lector a referirse al trabajo del M.Sc Salazar, [8], si se desea profundizar más en el tema. Exponer el funcionamiento del RTL del ASP también queda por fuera del alcance de este trabajo. En consecuencia a lo anterior cabe mencionar que se parte de un código considerado correcto y con base en él se procede con la integración según el flujo de diseño de circuitos digitales expuesto en el capítulo anterior.

En la figura 4.1, observamos la estructura del microprocesador que debía ser sintetizado, el RTL de este dispositivo mantiene una estructura equivalente a la de la figura 4.1; sin embargo, a primera impresión es posible encontrar una gran deficiencia en el diagrama de bloques mostrado. Esta deficiencia consiste en la ausencia de un hardware que permita cargarle información en las memorias. Al analizar el código del M.Sc. Salazar se encontró que las memorias utilizadas fueron definidas desde macros para una tarjeta de desarrollo FPGA.

La estructura del RTL de las memorias consiste en desarrollar pequeños bloques de memorias de ocho bits y generar un bloque de memoria total de treinta y dos bits. La técnica de codificación usada corresponde a un arreglo bidimensional en Verilog. En el caso de la memoria de programa o instrucciones, se crea un bloque de memoria de 32 Kbytes (kilo bytes), y se replica cuatro veces para tener una memoria de 32 Kbytes con un tamaño de palabra de treinta y dos bits. Lo mismo debe hacerse con la memoria de datos.

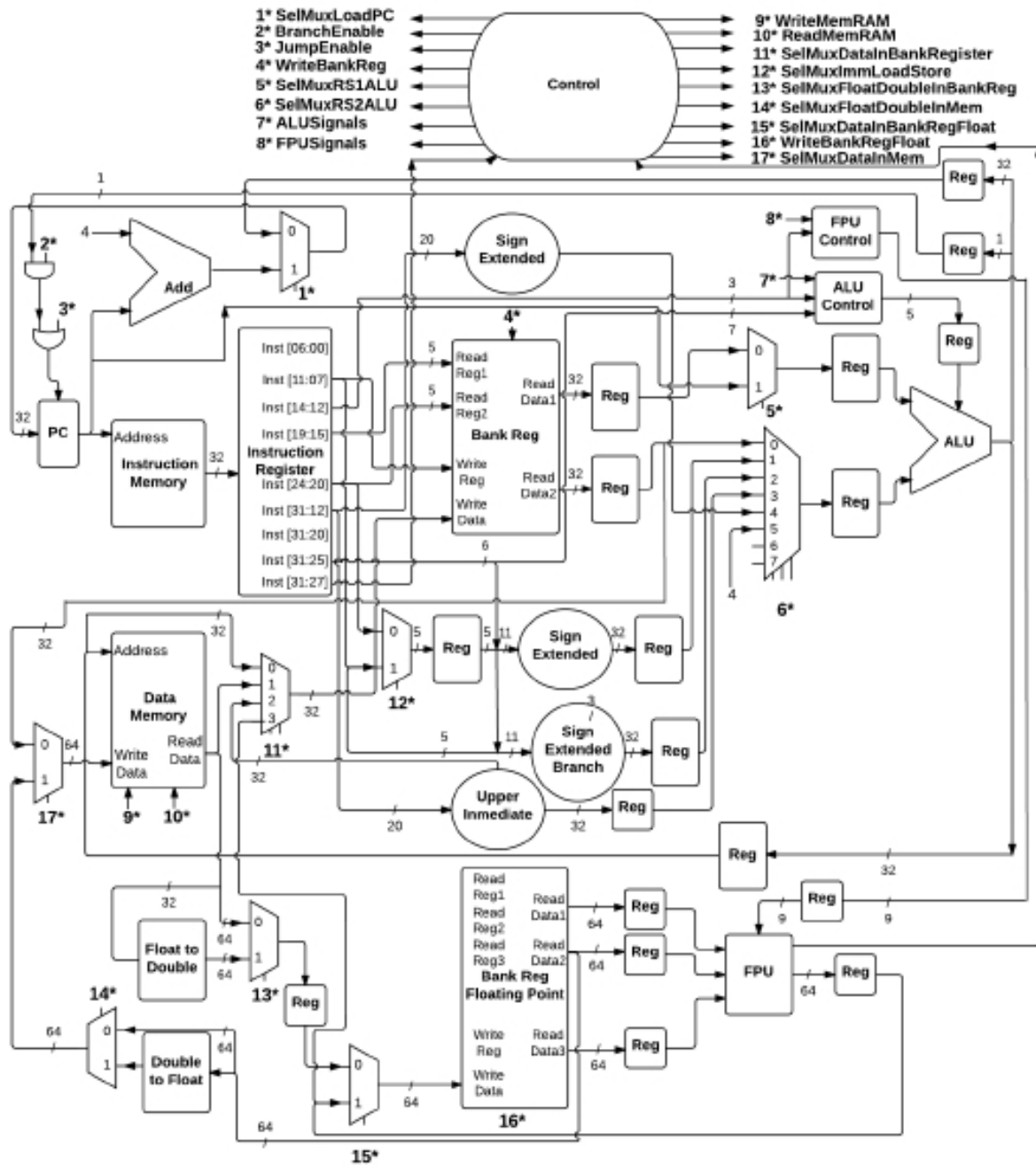


Figura 4.1: Diagrama funcional del microprocesador ASP. RISC-V. Imagen tomada del trabajo [8], con la autorización del M.Sc Salazar.

Sintetizar esta estructura de memorias directamente en la herramienta Design Compiler no es viable pues la misma no está diseñada para sintetizar memorias. Es por ello que conviene incluir bloques de memoria (SRAM en este caso), deben generarse por aparte y luego incluirse como IP Cores dentro del proceso de síntesis física.

Un detalle a considerar es que, al estudiar la estructura del diseño, se descubrió que no se ha generado una estructura en el ASP capaz de inicializar estas memorias (no existe tampoco un esquema adecuado de boot-strap). Esta situación deberá resolverse en otro proyecto, pero para este trae como consecuencia que no será posible hacer una validación funcional completa del circuito una vez terminado el flujo de implementación.

Existe una particularidad extra a este proyecto, y consiste en que la unidad de punto flotante original fue desarrollada por el Ing. Diego Rodríguez [24]; sin embargo, el diseño tenía muchas oportunidades de mejora, por lo que fue optimizada por el Ing. Francis López [25]. Esta última no fue probada junto al microprocesador, aunque se demostró su funcionalidad de forma individual. Al igual que con el caso anterior, tampoco será posible hacer una validación de esta etapa pues no ha sido integrada correctamente al ASP. Se procederá con la implementación de la FPU de [24] simplemente para completar el flujo correcto. Una vez resueltos los problemas de boot-strap, inicialización de memorias e incorporación de la FPU mejorada, será cuestión de corregir brevemente los scripts.

4.1 Estrategia de síntesis y validación

El primer objetivo específico de este trabajo, comprende el someter al ASP a todo el proceso de síntesis lógica. El flujo ya fue expuesto en el capítulo anterior, así que no se entrará en detalles sobre como fue sometido el código, pero se hablará de las configuraciones y consideraciones que se establecieron para que la implementación sea correcta.

4.1.1 Restricciones del diseño

Lo expuesto a continuación corresponde a los criterios usados en el script de TCL de constraints (restricciones) expuesto en la sección 3.2.1. Este es uno de los puntos de partida fundamentales para la síntesis lógica del ASP.

Modelo de carga en el cableado (`wire_load_model`)

El modelo de cableado utilizado en este diseño corresponde al `ibm13_w110` de la biblioteca `scx3_cmos8rf_lpvttt_1p2v_25c`. La forma para determinar los modelos de cableado se hace mediante la herramienta Design Compiler y solicitando el reporte de la biblioteca; sin embargo, se debe tener funcionando de forma correcta el Library Compiler. Existen varias modalidades de configuración para el modelo de cableado, en los diseños trabajados en este proyecto se utiliza el modelo “top”.

En el modo superior “top”, el compilador de diseño modela las redes como si el diseño no tuviese jerarquía y utiliza el modelo de carga de cable especificado para el nivel superior de la jerarquía de diseño de todas las redes en un diseño y sus subdiseños. La herramienta ignora todos los modelos de carga de cable configurados en subdiseños con el comando `set_wire_load_model`.

Reloj y factor de actividad

Los diseños se diseñan previendo una frecuencia de operación general de 100 MHz por lo que se crea un reloj de con un periodo de 10 ns. Con el fin de modelar un reloj más realista se especifica una transición de 0.5 ns, lo cual se traduce como una pendiente en los flancos de subida y bajada del reloj.

También se especifica una incertidumbre con márgenes de setup y de hold de 0.5 ns, que corresponden al lapso entre dos flancos sucesivos respecto a la variación fuera de los tiempos de llegada nominales. Finalmente se especifica una latencia de 0.5 ns, y es básicamente un parámetro que le indica a la herramienta el lapso que se da desde que la señal conmuta en su fuente hasta que conmuta en la entrada de una celda, y su utilidad es sólo para fines de análisis de sincronía y simulación dinámica.

El factor de conmutación ofrece una aproximación probabilística al número de veces totales que un circuito conmuta. Así es posible estimar preeliminarmente el consumo dinámico. Este parámetro se configura con una razón de conmutación del 25% y una probabilidad estática del 50% pues el reloj es simétrico.

Puertos y propagación de señales

Las señales de reloj y reset se configuran para que en ellas se coloquen la mínima cantidad de búfers, y que en general las señales se intervengan de forma mínima en la síntesis.

Se configura un intervalo de retraso en la propagación de los puertos, con el fin de que los análisis y la simulación sean más realistas, en el caso de las entradas el retardo es 1 ns como mínimo y de 3.5 ns como máximo, y de 1 a 2 ns para las salidas.

Los modelos de la carga asociada a las salidas y entradas se establecen en función de la celda de conducción (`driving_cell`), esta configuración le permite a la herramienta de análisis de sincronía estimar de forma precisa el retraso causado desde la conmutación en un puerto, y sobre toda la ruta de propagación consecuente.

Se establece un límite de 10 al máximo fanout para cada celda. El restringir desde la entrada (`driving_cell`) hasta la salida (fanout) de cada camino lógico es necesario como punto de partida de las rutinas de optimización de retardo (cálculos de esfuerzo lógico).

4.1.2 Comprobación de la síntesis

Como ya se mencionó el diseño de [8], presenta algunas deficiencias a la hora de considerar someterlo al flujo de diseño digital. En respuesta a lo anterior se efectuó el siguiente proceso.

1. En primer lugar se modificó el RTL haciendo un *bypass* en la conexión de la memorias, creando puertos para conectar de forma externa las memorias con los módulos que las instanciaban originalmente.

2. Se tomó el RTL original con la FPU diseñado por el Ing. Rodríguez, y se estimuló de acuerdo con el banco de pruebas (*testbench*) proporcionado junto al RTL del ASP. Esta simulación por comportamiento se define como la referencia dorada para validar el diseño.
3. Luego se sustituye la FPU por la diseñada por el Ing. López y se estimula el ASP de nuevo en una simulación por comportamiento para valorar la congruencia con el diseño original.
4. Se sintetiza entonces el RTL sin el banco de memorias.
5. Se realiza una simulación post síntesis lógica del ASP usando un estrategia de simulación híbrida, en el sentido de que se combinan elementos de RTL por comportamiento y post síntesis. Así se pretende modelar las memorias como elementos ideales y poder comprobar la funcionalidad del ASP.

4.2 Análisis de resultado: síntesis lógica del ASP

4.2.1 Evaluación pre síntesis

Se realizó un análisis a nivel de comportamiento del código del ASP que incorpora la FPU desarrollada en [24].

Los datos observados en las figuras 4.2, 4.3 4.4 correspondientes a los vectores de salida: ProbBosque, ProbMotosierra, y ProbDisparo son datos en hexadecimal, que representan un número en punto flotante de precisión simple, según el estándar IEEE574. Por lo tanto tenemos que para la figura 4.2, según el estímulo usado (definido en el *testbench*, aportado junto al *RTL*) los valores corresponden a -14,779346 para ProbBosque, -9,637644 para ProbMotosierra, y -308,25473 para ProbDisparo.

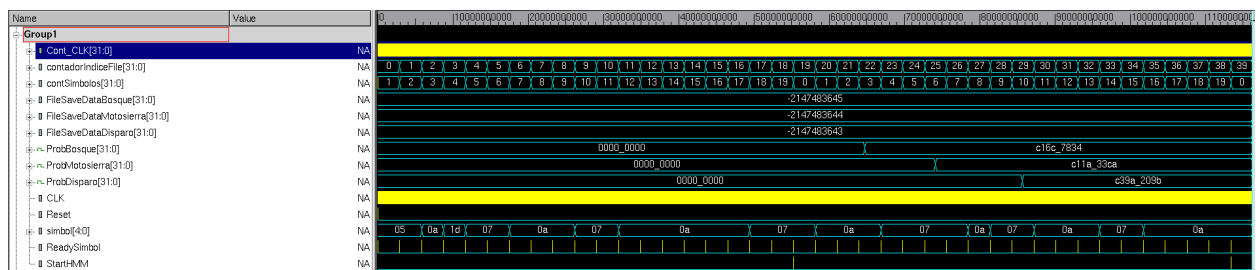


Figura 4.2: Resultado de la simulación por comportamiento del ASP con la FPU diseñada en [24]. Según lo estipulado por el M.Sc Salazar en [8]. Esta es la referencia dorada para este trabajo.

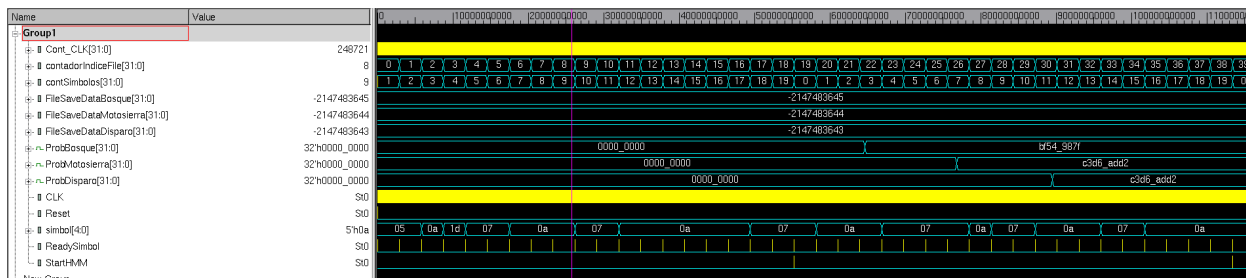


Figura 4.3: Verificación del funcionamiento del ASP al cambiar la FPU por la versión mejorada de [25, 35]. El resultado final del cálculo de probabilidad es erróneo.

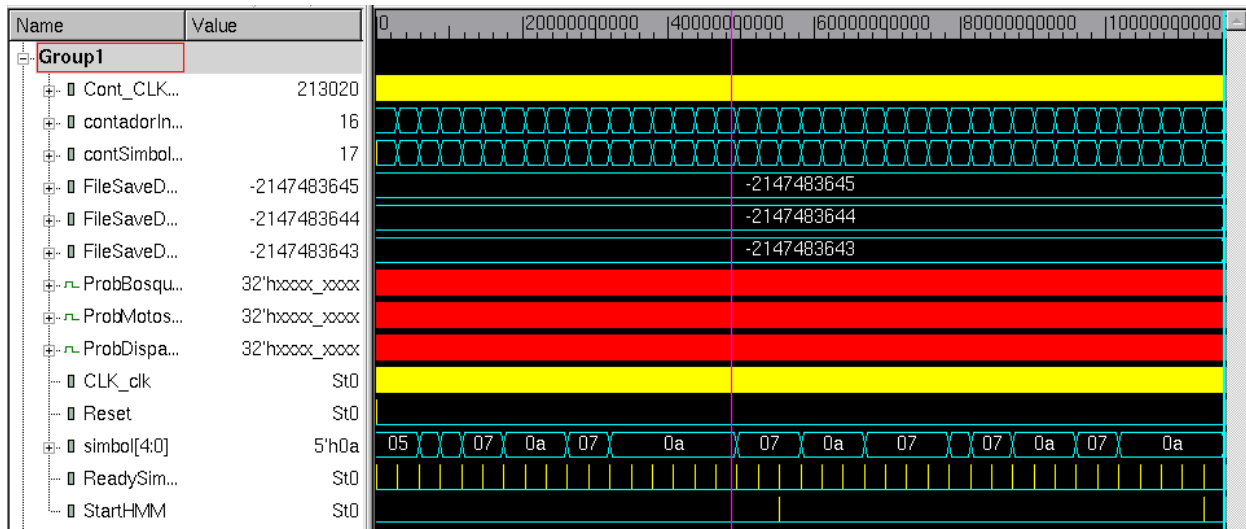


Figura 4.4: Resultado de simulación post síntesis lógica del ASP, con la FPU de [25, 35]. Nótese la existencia de señales indefinidas. Un análisis más detallado mostrará violaciones en la restricción de tiempo de mantenimiento en el resultado de la síntesis lógica.

Algoritmo HMM ejecutado en el ASP

Según se expone en [8] los valores puntualmente no corresponden a una estadística, pues no existen probabilidades negativas, por lo que estos valores son más una heurística que permite determinar que tan cercano se encuentra un dato a pertenecer a uno de los tres conjuntos mencionados; específicamente, qué tan cercanos se encuentren estos valores a cero. Observando que para la figura 4.2, se aprecia que el valor más cercano a cero (en otras palabras, con una probabilidad más cerca de uno), corresponde al vector **ProbMotosierra**, por lo tanto, el estímulo tiene una probabilidad más alta de pertenecer a un patrón acústico tipo motosierra; sin embargo, existe una probabilidad modesta de asociar el estímulo al patrón acústico del bosque, lo cual es esperable de acuerdo con las condiciones en las que fueron adquiridos los sonidos que se usaron como estímulos. Finalmente se observa que el estímulo definitivamente no puede considerarse como un disparo, pues su valor se encuentra dos ordenes de magnitud lejos del cero (más detalles del funcionamiento del sistema de clasificador completo puede hallarse en [8, 34, 35]).

Partiendo de la premisa de que el *testbench* contiene un estímulo que corresponde al patrón acústico de una motosierra, se procede a efectuar un contraste entre las figuras 4.2, 4.3. En este se puede encontrar que *ProbBosque* tiene un valor -0,8304519 (0xbf54987f), mientras que *ProbMotosierra* y *ProbDisparo* tienen un valor de -429,35797 (0xc3d6add2). Lo cual según el criterio expuesto anteriormente conduce a considerar que el estímulo es definitivamente sonido de bosque, pues los otros valores están muy lejos del cero. Conociendo que el estímulo en principio es un sonido asociado a una motosierra, y que la respuesta observada en la figura 4.2 es correcta, se pudo establecer que la FPU propuesta en [25] funciona erróneamente.

4.2.2 Evaluación post síntesis

Ya que el objetivo de este proyecto es el dejar establecido el flujo de diseño completo, se decidió proseguir con el mismo pese a los errores físicos reportados por las herramientas. Quedará para otro trabajo el corregir los errores en la unidad FPU de [25]. Como puede observarse en la tabla 4.1, la red de reloj no consume energía, como tampoco los bloques de lógica secuencial. Ello se debe a que al experimentar con el RTL del microprocesador se encontraron las deficiencias la codificación, comentadas anteriormente, por lo que se efectuó la compilación del diseño considerando una red de reloj ideal. Sin embargo, el reporte de potencia permite identificar el grado de consumo de los distintos bloques en el diseño, y dado que se extrajeron los bloques de memoria de este diseño, además de obligar a la herramienta a considerar la red de reloj como una red ideal, puede observarse que el grueso del consumo lo tienen las categorías de registros y lógica combinacional.

El reporte de temporizado, apreciado en el listado 4.1, permite observar el comportamiento de la red con el camino crítico, que como ya se mencionó es determinada dinámicamente por la herramienta durante el proceso de compilación en el flujo de la síntesis lógica. Nuevamente el reporte de *timing* permite apreciar que la red de reloj tiene un comportamiento ideal. En este mismo reporte, se ofrece al final una descripción completa de la ruta que sigue la señal más lenta en el diseño. En la primera columna, se va indicando la celda por la que atraviesa la señal (dato) en su camino, la segunda columna indica cuánto retraso aporta a la señal la celda susodicha, y la tercera ofrece un acumulado del retardo hasta ese punto. La suma completa al final de la ruta ofrece el tiempo total de arribo de la señal a su destino (*data arrival time*).

Listado 4.1: Reporte de temporizado del microprocesador ASP post síntesis lógica.

```
*****
Report : timing -path full -delay max -max_paths 1
Design : MainHMM.2 Version: L-2016.03-SP3
*****
                Path Group: clk | Path Type: max
-----
Des/Clust/Port      Wire Load Model      Library
MainHMM.2           ibm13-wl10           scx3-cmos8rf-lpvt-tt-1p2v-25c
-----
Point              Incr                Path
```

clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	1.00	1.00
\$A/State_reg[3]/CK (DFFQX1TS)	0.00 #	1.00 r
\$A/State_reg[3]/Q (DFFQX1TS)	1.05	2.05 f
CPUMain/U367/Y (INVX4TS)	0.53	2.59 r
CPUMain/U2567/Y (INVX4TS)	0.49	3.08 f
CPUMain/U235/Y (OR3X2TS)	0.68	3.76 f
CPUMain/U972/Y (CLKIN VX3TS)	0.57	4.32 r
CPUMain/U180/Y (NOR2X1TS)	0.47	4.79 f
CPUMain/U16337/Y (OAI222X4TS)	0.86	5.65 r
CPUMain/U147/Y (NOR3BX1TS)	0.49	6.14 r
CPUMain/U134/Y (BUFX3TS)	0.76	6.90 r
CPUMain/U5970/Y (NOR2X4TS)	0.51	7.41 f
CPUMain/U20520/Y (NOR2X2TS)	0.72	8.13 r
CPUMain/U9888/Y (BUFX3TS)	0.88	9.01 r
CPUMain/U6270/Y (BUFX3TS)	0.83	9.83 r
CPUMain/U20522/Y (OAI22X1TS)	0.48	10.31 f
\$B/DataOutput_reg[0]/D (DFFQX1TS)	0.00	10.31 f
data arrival time		10.31
clock clk (rise edge)	10.00	10.00
clock network delay (ideal)	1.00	11.00
clock uncertainty	-0.50	10.50
\$B/DataOutput_reg[0]/CK (DFFQX1TS)	0.00	10.50 r
library setup time	-0.19	10.31
data required time		0.31
<hr/>		
data required time		10.31
data arrival time		-10.31
<hr/>		
slack (MET)		0.00
<hr/>		

Notes:

\$A = CPUMain/ExtensionFloat.FSMControlCPUFloatPoint

\$B = CPUMain/BankRegisterInteger/Registro5_t0

Finalmente en el reporte de timing se consideran los atributos de la red de reloj que se definieron en las restricciones de síntesis, y se define un tiempo límite en el que el dato debería llegar (*data required time*), la diferencia entre el tiempo requerido y el tiempo de arribo del dato establecen el *slack*, que indica si un diseño funcionaría correctamente a la velocidad requerida. El slack debe ser mayor a cero, es deseable que ofrezca un margen de seguridad. Un resultado negativo del slack aquí sería reportado como una violación de setup por la herramienta. No obstante, es conveniente mencionar que, a estas alturas del diseño, estas consideraciones son preliminares (en principio porque puede que la red de reloj no está definida aún de manera muy exacta), y que bien pueden aceptarse violaciones menores en los requerimientos. Es muchas veces posible corregir estas violaciones conforme

se traslada el diseño al back end. Por otro lado, una violación significativa (de algunos nanosegundos en relación con los tiempos usados en este diseño), implica un código RTL defectuoso que debe corregirse antes de proseguir. Como se puede observar este diseño cumple justamente el *slack*, lo cual es aceptable, aunque suelen ser deseables *slacks* positivos.

Como puede observarse en la simulación post síntesis, las señales correspondientes a los vectores a: `ProbBosque`, `ProbMotosierra`, y `ProbDisparo`, se encuentran en estado indeterminado (“x”), lo que significa que el modelo de síntesis lógica no está funcionando de la misma manera en que lo hace el RTL; sin embargo, no quiere decir que el producto de la síntesis lógica sea erróneo. Esto quiere decir que la herramienta Design Compiler es capaz de abstraer el diseño de forma correcta en lo que respecta a celdas estándar; la correspondencia entre el comportamiento también depende de las restricciones definidas por el usuario. Al inicio de la sección 4.1.1 se definió un factor de transición de 0.5 ns, que implica una pendiente lenta, por lo que el consumo de potencia se eleva, y es muy probable que fallen algunos *flipflops*.

Considerando el escenario de establecido al principio de este capítulo donde se definió que para poder efectuar la síntesis lógica, se iban a extraer los bloques de memoria del microprocesador, y como se planteó posteriormente en la sección 4.1.2, el proceso de síntesis iba a culminar con una simulación mixta, en el sentido de que se iba a usar el RTL original de los bloques de memorias, junto al GLN generado por la herramienta de síntesis, es de esperar que el diseño no funcione adecuadamente. Una simulación post síntesis trata precisamente de incluir efectos de retardo de las señales. Esta información se obtiene al realizar la síntesis lógica. Un RTL descrito por comportamiento no ofrece información de retardos o temporizado. La herramienta de simulación opera con información incompleta, obteniendo resultados incorrectos.

En la figura 4.5 se observa una captura de los mensajes que ofrece la herramienta al compilar la simulación deseada. Como puede apreciarse en la ventana de mensajes, que ha sido resaltada con un recuadro rojo, para este diseño se observa una enorme cantidad de errores de “hold”, es decir que no se cuplen las restricciones físicas de mantenimiento (hold) para un registro dado, según indica la herramienta en la ventana de dependencias. En la imagen se resalta uno de los registros en cuestión mediante un recuadro azul. Con esta información es posible buscar los registros ofensores y tratar de hacer cumplir la restricción mediante la inserción manual de retardos en el mismo archivo GLN, o de forma automática, que puede tratar de hacer la misma herramienta con el comando `set_fix_hold`. Esta estrategia fue tomada en cuenta; durante la experimentación con el comando `set_fix_hold` no se logró determinar su uso correcto, y las violaciones de *hold* no fueron corregidas.

Este tipo de errores pueden tener distintas fuentes; sin embargo, el reporte de timing indica que el slack se cumple para el camino crítico, podría analizarse si alguno de los registros ofensores pertenece al camino crítico y realizar las correcciones correspondientes con la herramienta de síntesis. También es posible usar la herramienta para STA y generar un panorama

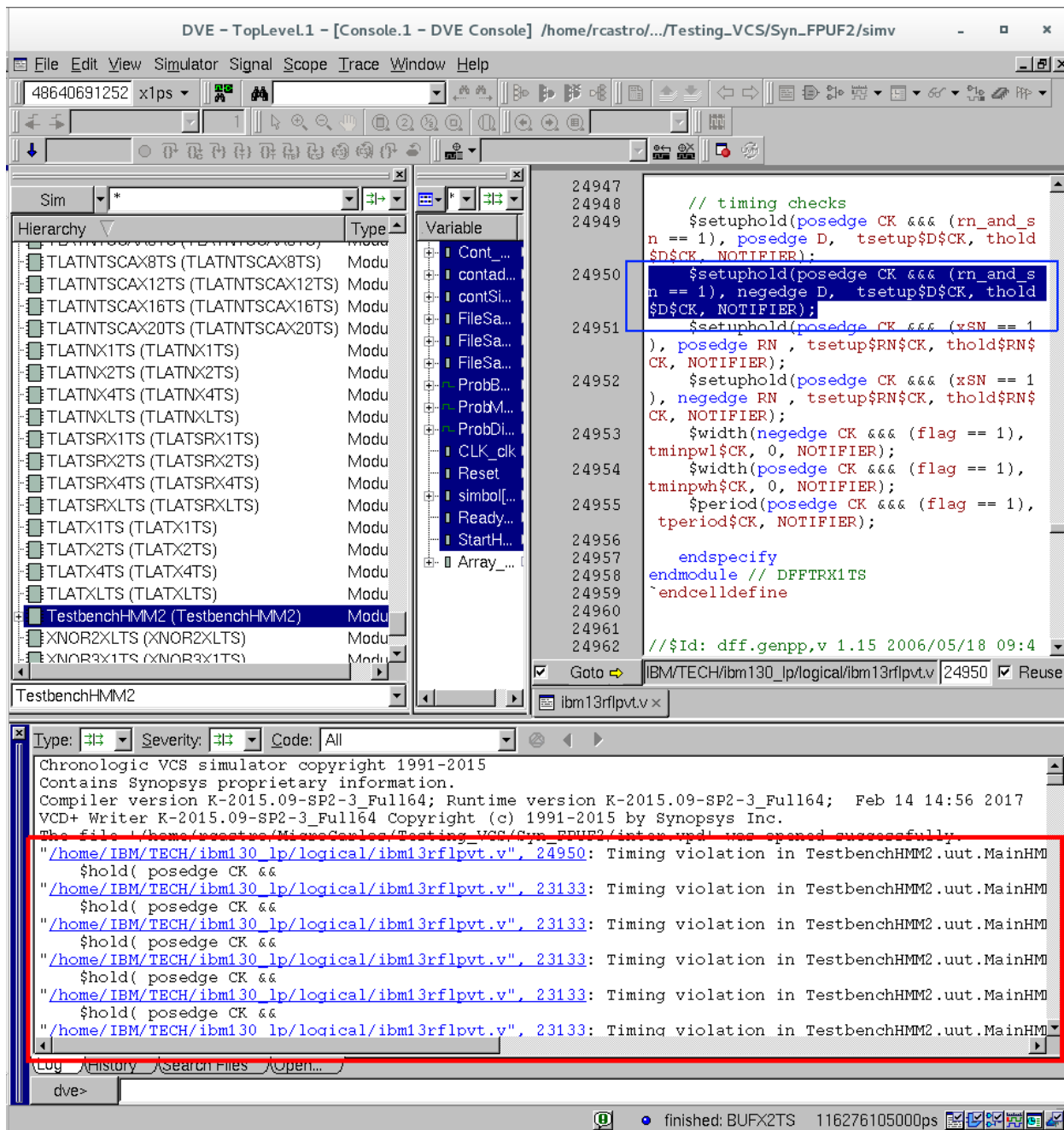


Figura 4.5: Detalle de los resultados de la verificación post-síntesis, proporcionados por VCS. Nótese la violación de hold. Esto implica que hay que corregir el problema, ya sea desde DesignCompiler, o modificando el RTL si el DC no logra su cometido.

más amplio sobre que sucede puntualmente con el diseño. Pero partiendo del hecho de que se está realizando una simulación enfocada a un código en GLN, en la que se ha mezclado RTL sin información de retardos y que el diseño se probó erróneo mediante las simulaciones por comportamiento, no es viable intentar optimizarlo.

Finalmente en el listado 4.2 se observa el reporte de área, que es un compendio de la cantidad de elementos y su categoría dentro de la biblioteca, abstrayendo su información en términos de espacio. Este reporte es apenas un estimado de área, pero que sirve al diseñador como punto de partida para cuando deba proponer el plan de piso inicial para su diseño en la etapa de back end.

Listado 4.2: Reporte de área del microprocesador ASP post síntesis lógica.

```

*****
Report : area
Design : MainHMM_2
Version: L-2016.03-SP3
Date   : Wed Dec 14 13:21:38 2016
*****

Library(s) Used: scx3_cmos8rf_lpvt_tt_1p2v_25c

Number of ports:          920
Number of nets:          40225
Number of cells:         37979
Number of combinational cells: 32684
Number of sequential cells:  5293
Number of macros/black boxes:  0
Number of buf/inv:        8740
Number of references:     40

Combinational area:      318738.243849
Buf/Inv area:            49842.720788
Noncombinational area:   139687.197126
Macro/Black Box area:    0.000000
Net Interconnect area:   4646623.786163

Total cell area:         458425.440975
Total area:              5105049.227139

```


Capítulo 5

Implementación física de la FPU del ASP y bloques de memoria

Se describe en este capítulo el proceso completo de back end, usando como ejemplo la FPU descrita en [25, 35], y la incorporación de un macro de SRAM, desarrollado con la herramienta de generación de memorias provista por el kit de ARM.

Con respecto a la FPU, se partió del RTL, haciendo una integración jerárquica. La primera corresponde a síntesis lógica de los dos módulos principales (suma y multiplicación) y la evaluación post síntesis de las mismas, la segunda etapa consiste en la integración física de cada unidad por separado, y posteriormente se efectúa una evaluación post implementación. Por último se integra el siguiente nivel en la jerarquía, que sirve como módulo principal que instancia los otros dos módulos mencionados.

Tanto la síntesis lógica como física siguen el proceso descrito en el capítulo anterior, según las secciones 3.2 y 3.3 respectivamente. Las restricciones consideradas para el proceso de front end y consecuentemente los scripts para la síntesis lógica son las mismas que las mostradas en la sección 4.1.1, por lo que no se redundará en esos aspectos.

Respecto a la síntesis física, se consideraron las siguientes configuraciones para la síntesis del árbol de reloj:

- Inserción de celdas frontera cercanas a los puertos de reloj para el diseño basado en jerarquía de bloques.
- `OCV_Clustering`, para mejorar la distribución del reloj a través de ramas de registros.
- Permitir la re-ubicación y el re-dimensionamiento de las compuertas y búfers para optimizar la síntesis del árbol de reloj.

Las reglas de antena tienen los siguientes atributos:

- Modalidad de protección de diodo limitada.

- Una razón de metal de 150, un tamaño de compuerta por defecto de 0,1, con una protección por defecto de 0,5.
- Un radio de corte de 20, y una razón de diodo comprendida en el siguiente vector $\{0,09$
 0 123 $16880\}$ para los metales de las capas bajas(cuatro primeras capas de metal), mientras que para las vías de estos primeros cuatro metales el vector sería $\{0,09$ 0 110
 $500\}$

Respecto al enrutado global no hay una serie de restricciones o configuraciones específicas para que la herramienta logre enrutar de forma adecuada toda la celda. Este proceso es algo engorroso, y responde más a una metodología iterativa y de optimización selectiva de acuerdo con los resultados que se van dando al trabajar con el trazado del diseño. Sin embargo, se especifican parámetros como la reducción del efecto de diafonía (*Crosstalk*) para que en el enrutado se eviten colocar cables largos adyacentes a rieles paralelos, y tener un esfuerzo alto al controlar la ejecución de enrutamiento detallado e iterativo para determinar si el DRC diverge.

En relación con los *IP Cores* de las memorias, se efectuó un flujo equivalente. No obstante los *IP Cores* no pueden optimizarse ni trabajarse con mucho detalle, pues precisamente son cajas negras que buscan la protección de la propiedad intelectual. Así que para corroborar que la implementación de estos bloques fuese correcta se evaluaron diseños que hacen de envoltorio (wrapper) para estas celdas, en inglés a esta técnica se le conoce como "*Wrapper*" y en esencia consiste en crear un código que instancie *IP Core* y se sintetiza con base en él. Esto no es estrictamente necesario ya que el código que genera la herramienta tiene una estructura GLN y permite hacer simulaciones post síntesis, si se generan los archivos *.sdf mediante un *script* de PERL. La alternativa del *wrapper* permite usar el *IP Core* como una instancia y usar la metodología que ha sido definida para el flujo.

5.0.1 Análisis de los resultados obtenidos en la implementación de la FPU

En la figura 5.1 se muestra el resultado de la implementación de la unidad aritmética de punto flotante en la tecnología IBM 0.13 según el flujo de diseño establecido en este trabajo.

Potencia

Comenzando por el análisis de los reportes de potencia post síntesis lógica y post síntesis física (tablas 5.1 y 5.2) se aprecia que respecto a la disipación de energía debido al consumo interno de las celdas y la perdidas por corrientes de fuga (leakage) se mantienen en valores muy similares, pues estos son consecuencias de las celdas propiamente y estos parámetros son difícilmente controlables; sin embargo, la herramienta de IC Compiler de Synopsys permite efectuar una optimización de las pérdidas por corrientes de fuga. Como puede apreciarse

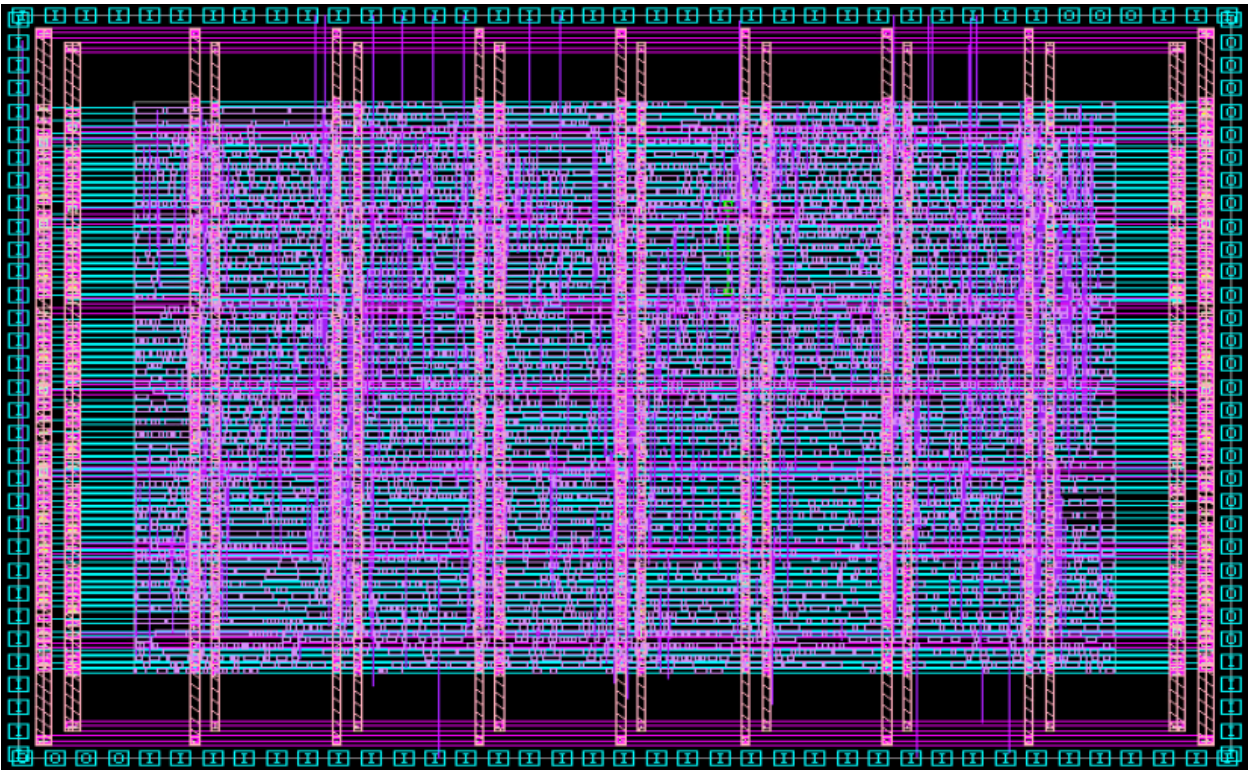


Figura 5.1: Foto captura del layout de la celda física generada para la unidad de punto flotante del ASP

este parámetro no varía significativamente. No fue optimizado debido a que su valor es considerablemente bajo y en el proceso (130 nm), no es tan crítico para tomarlo en cuenta, aunque es un posible aspecto a considerar en aplicaciones a escalas más pequeñas y procesos modernos.

Tabla 5.1: Resumen del reporte de potencia de la FPU post síntesis lógica.

Group	Internal	Switching	Leakage	Total	% Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	0.00%
memory	0.0000	0.0000	0.0000	0.0000	0.00%
black_box	0.0000	0.0000	0.0000	0.0000	0.00%
clock_network	0.0000	0.0000	0.0000	0.0000	0.00%
register	1.2521	0.1130	2.4467e+04	1.3651	60.92%
sequential	0.0000	0.0000	0.0000	0.0000	0.00%
combinational	0.2211	0.6546	5.0897e+04	0.8757	39.08%
Total	1.4732 mW	0.7675 mW	75.364e+04 nW	2.2408 mW	100%

Analizando estos reportes observamos que no existe disipación por elementos secuenciales, esto es esperable ya que los módulos aritméticos usados consisten principalmente en lógica combinatorial y registros gobernados por una pequeña máquina de estados, que respecto al volumen de los demás elementos no aporta un consumo significativo.

Tabla 5.2: Resumen del reporte de potencia de la FPU post síntesis física.

Group	Internal	Switching	Leakage	Total	% Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	0.00%
memory	0.0000	0.0000	0.0000	0.0000	0.00%
black_box	0.0000	0.0000	0.0000	0.0000	0.00%
clock_network	0.0000	0.0000	0.0000	0.0000	0.00%
register	1.2530	4.7273e-02	2.4587e+04	1.3003	78.51%
sequential	0.0000	0.0000	0.0000	0.0000	0.00%
combinational	0.1274	0.2285	5.1105e+04	0.3560	21.49%
Total	1.3804 mW	0.2758 mW	75.692e nW	1.6563 mW	100%

Área

Respecto a los reportes de área, códigos: 5.1 y 5.2, se observa que en el primero (reporte de síntesis lógica) se tienen una cantidad considerablemente superior de todos los elementos, respecto al segundo reporte que es el de síntesis física. Esto se debe a la metodología de diseño jerárquico, recordando lo expuesto en la sección 3.3 el diseño jerárquico del layout corresponde a generar una única biblioteca milkyway y cada subdiseño se sintetizará en esta como una celda, posteriormente conforme se avanza en diseños más complejos que usan estas celdas como instancias en el nuevo diseño, siguiendo una metodología *BottomUp*.

Listado 5.1: Reporte de área de la unidad de punto flotante de ASP post síntesis lógica.

```

*****
Report : area
Design : FloatingPointUnit
Version: L-2016.03-SP3
Date   : Tue Jan  3 18:53:58 2017
*****

Library(s) Used: scx3_cmos8rf_lpvt_tt_1p2v_25c

Number of ports:                355
Number of nets:                  5590
Number of cells:                 4909
Number of combinational cells:   4227
Number of sequential cells:      680
Number of macros/black boxes:    0
Number of buf/inv:               792
Number of references:            54

Combinational area:              41843.520412
Buf/Inv area:                    3906.720098
Noncombinational area:          20897.279408

```

Macro/Black Box area:	0.000000
Net Interconnect area:	588843.014587
Total cell area:	62740.799820
Total area:	651583.814407

Listado 5.2: Reporte de área de la unidad de punto flotante de ASP post síntesis lógica.

```

*****
Report : area
Design : FloatingPointUnit
Version: L-2016.03-SP3
Date   : Tue Jan  3 18:53:58 2017
*****

Library(s) Used: scx3_cmos8rf_lpvt_tt_1p2v_25c

Number of ports:          144
Number of nets:           825
Number of cells:          646
Number of combinational cells: 513
Number of sequential cells: 131
Number of macros/black boxes: 0
Number of buf/inv:        136
Number of references:      63

Combinational area:       42115.680429
Buf/Inv area:             4190.400109
Noncombinational area:    20897.279408
Macro/Black Box area:     0.000000
Net Interconnect area:    undefined
                          (No wire load specified)

Total cell area:          63012.959836
Total area:               undefined

```

Listado 5.3: Reporte de temporizado del microprocesador ASP post síntesis lógica.

```

*****
Report : timing -path full -delay max -max_paths 1
Design : FloatingPointUnit Version: L-2016.03-SP3
*****
Operating Conditions: tt_1p2v_25c
Library: scx3_cmos8rf_lpvt_tt_1p2v_25c
Parasitic source      : LPE
Parasitic mode        : RealRC

```

Extraction mode : MIN_MAX
 Extraction derating : 25/25/25

Startpoint: FPU_Multiplication_Function/Operands_load_reg/
 XMRegister/Q_reg[10](rising edge-triggered flip-flop clocked
 by CLK)

Endpoint: FPU_Multiplication_Function/Sgf_operation/
 SgfM_mul_0.middle/pdt_int_reg[15](rising edge-triggered
 flipflop clocked by CLK)

Path Group: clk | Path Type: max

Des/Clust/Port	Wire Load Model	Library		
FloatingPointUnit	ibm13_wl10	scx3_cmos8rf_lpvt_tt_1p2v_25c		
Point			Incr	Path
clock CLK (rise edge)			0.00	0.00
clock network delay (ideal)			1.00	1.00
FPU_Multiplication_Function/Operands_load_reg/ XMRegister/Q_reg[10]/CK (DFFRX4TS)			0.00	1.00 r
FPU_Multiplication_Function/Operands_load_reg/ XMRegister/Q_reg[10]/QN (DFFRX4TS)			0.88	1.88 f
FPU_Multiplication_Function/U1092/Y (INVX2TS)			0.44 &	2.32 r
FPU_Multiplication_Function/U1766/Y (CLKXOR2X2TS)			0.58 &	2.90 f
FPU_Multiplication_Function/U1767/Y (NOR2X2TS)			0.18 &	3.08 r
FPU_Multiplication_Function/U1211/Y (XOR2X1TS)			0.49 &	3.56 r
FPU_Multiplication_Function/U1191/Y (NAND2X6TS)			0.35 &	3.92 f
FPU_Multiplication_Function/U2505/Y (OAI22X1TS)			0.59 &	4.51 r
FPU_Multiplication_Function/U1066/S (CMPR32X2TS)			1.06 &	5.57 f
FPU_Multiplication_Function/DP_OP_110J1_122_5487/ U307/CO (CMPR42X1TS)			1.15 &	6.73 f
FPU_Multiplication_Function/DP_OP_110J1_122_5487/ U303/S (CMPR42X2TS)			1.01 &	7.74 f
FPU_Multiplication_Function/U1593/Y (NOR2X4TS)			0.18 &	7.92 r
FPU_Multiplication_Function/U1594/Y (NOR2X2TS)			0.14 &	8.06 f
FPU_Multiplication_Function/U1596/Y (AOI21X4TS)			0.33 &	8.39 r
FPU_Multiplication_Function/U408/Y (INVX2TS)			0.22 &	8.61 f
FPU_Multiplication_Function/U1607/Y (AOI21X1TS)			0.22 &	8.82 r
FPU_Multiplication_Function/U1051/Y (XOR2X1TS)			0.46 &	9.28 r
FPU_Multiplication_Function/Sgf_operation/ SgfM_mul_0.middle/pdt_int_reg[15]/D (DFFQX1TS)			0.00 &	9.28 r
data arrival time				9.28
clock CLK (rise edge)			10.00	10.00
clock network delay (ideal)			1.00	11.00

clock uncertainty	-0.50	10.50
FPU_Multiplication_Function/Sgf_operation/ SgfM_mul_0.middle/pdt_int_reg[15]/CK (DFFQX1TS)	0.00	10.50 r
library setup time	-0.27	10.23
data required time		10.23
<hr/>		
data required time		10.23
data arrival time		-9.28
<hr/>		
slack (MET)		0.94

Startpoint: FPU_Add_Subtract_Function/Oper_Start_in_module/
YRegister/Q_reg[31](rising edge-triggered flip-flop clocked by clk)

Endpoint: FPU_Add_Subtract_Function/Add_Subt_Sgf_module/
Add_Subt_Result/Q_reg[24](rising edge-triggered flip-flop
clocked by clk)

Path Group: clk | Path Type: max

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	1.00	1.00
FPU_Add_Subtract_Function/Oper_Start_in_module/ YRegister/Q_reg[31]/CK (DFFRX1TS)	0.00	1.00 r
FPU_Add_Subtract_Function/Oper_Start_in_module/ YRegister/Q_reg[31]/Q (DFFRX1TS)	1.10	2.10 f
FPU_Add_Subtract_Function/U1634/Y (CLKXOR2X4TS)	0.45 &	2.55 f
FPU_Add_Subtract_Function/U1102/Y (XOR2X2TS)	0.35 &	2.91 r
FPU_Add_Subtract_Function/U1130/Y (NAND2X4TS)	0.34 &	3.25 f
FPU_Add_Subtract_Function/U1129/Y (INVX8TS)	0.17 @	3.42 r
FPU_Add_Subtract_Function/U1885/Y (XOR2X1TS)	0.43 @	3.85 r
FPU_Add_Subtract_Function/U1901/Y (NAND2X1TS)	0.43 &	4.28 f
FPU_Add_Subtract_Function/U880/Y (OAI21XLTS)	0.45 &	4.73 r
FPU_Add_Subtract_Function/U1903/Y (AOI21X1TS)	0.25 &	4.98 f
FPU_Add_Subtract_Function/U1904/Y (OAI21X2TS)	0.15 &	5.13 r
FPU_Add_Subtract_Function/U1436/Y (AOI21X2TS)	0.16 &	5.29 f
FPU_Add_Subtract_Function/U1913/Y (OAI21X4TS)	0.21 &	5.50 r
FPU_Add_Subtract_Function/U1917/Y (AOI21X4TS)	0.15 &	5.65 f
FPU_Add_Subtract_Function/U1921/Y (OAI21X4TS)	0.17 &	5.82 r
FPU_Add_Subtract_Function/U1925/Y (AOI21X4TS)	0.13 &	5.95 f
FPU_Add_Subtract_Function/U1929/Y (OAI21X1TS)	0.41 &	6.36 r
FPU_Add_Subtract_Function/U1933/Y (AOI21X4TS)	0.27 &	6.63 f
FPU_Add_Subtract_Function/U1937/Y (OAI21X1TS)	0.46 &	7.09 r
FPU_Add_Subtract_Function/U1941/Y (AOI21X4TS)	0.30 &	7.39 f
FPU_Add_Subtract_Function/U2006/Y (XOR2X1TS)	0.58 &	7.97 r

FPU_Add_Subtract_Function/U905/Y (CLKMX2X2TS)	0.60	&	8.57	r
FPU_Add_Subtract_Function/Add_Subt_Sgf_module/ Add_Subt_Result/Q_reg[24]/D (DFFRX2TS)	0.00	&	8.57	r
data arrival time			8.57	
clock clk (rise edge)	10.00		10.00	
clock network delay (ideal)	1.00		11.00	
clock uncertainty	-0.50		10.50	
FPU_Add_Subtract_Function/Add_Subt_Sgf_module/ Add_Subt_Result/Q_reg[24]/CK (DFFRX2TS)	0.00		10.50	r
library setup time	-0.28		10.22	
data required time			10.22	
<hr/>				
data required time			10.22	
data arrival time			-8.57	
<hr/>				
slack (MET)			1.65	

El reporte 5.2 (síntesis física) presenta una menor densidad de elementos pues reconoce a los bloques de suma/resta y multiplicación como una celda estándar, en el caso del reporte 5.1 (síntesis lógica) la herramienta considera todas las dependencias de los módulos mencionados. También se puede observar que el reporte 5.2 (síntesis física) no presenta información sobre el área total que ocupa el diseño, y la información que aporta sólo indica que el diseño presenta una caja negra. No se puede ofrecer una respuesta puntual a esta situación, es probable que esto se deba a una mala configuración de algún parámetro al crear la biblioteca o importar los archivos fuente para generar el layout.

Temporizado

Los reportes de síntesis lógica (5.4) no aportan mucha información relevante pues el modelo de estimación es bastante impreciso debido a la ausencia de un layout formal; sin embargo, sirven como precedente para saber si el diseño abstraído del RTL es coherente con la tecnología y podría ser funcional en la misma. Los reportes de síntesis física (5.5, 5.7 y 5.6), permiten observar un mejor comportamiento de la propagación de un dato sobre la ruta crítica.

Listado 5.4: Reporte de temporizado de la FPU post síntesis lógica.

```
*****
Report : timing -path full -delay max -max_paths 1
Design : FloatingPointUnit Version: L-2016.03-SP3
*****
          Path Group: clk | Path Type: max

Startpoint: FPU_Multiplication_Function/Operands_load_reg/
```

XMRegister/Q_reg[21](rising edge-triggered flip-flop
clocked by CLK)

Endpoint: FPU_Multiplication_Function/Sgf_operation/SgfM_mul_0.left/
pdt_int_reg[23](rising edge-triggered flip-flop clocked by CLK)

Des/Clust/Port	Wire Load Model	Library	
FloatingPointUnit	ibm13_wl10	scx3_cmos8rf_lpvttt_1p2v_25c	
Point		Incr	Path
clock CLK (rise edge)		0.00	0.00
clock network delay (ideal)		1.00	1.00
FPU_Multiplication_Function/Operands_load_reg/ XMRegister/Q_reg[21]/CK (DFFRX4TS)		0.00	1.00 r
FPU_Multiplication_Function/Operands_load_reg/ XMRegister/Q_reg[21]/QN (DFFRX4TS)		1.18	2.18 r
FPU_Multiplication_Function/U856/Y(INVX2TS)		0.27	2.45 f
FPU_Multiplication_Function/U1080/Y(CLKXOR2X2TS)		0.63	3.08 f
FPU_Multiplication_Function/U524/Y (INVX2TS)		0.46	3.54 r
FPU_Multiplication_Function/U1071/Y (AND2X2TS)		0.48	4.02 r
FPU_Multiplication_Function/U554/Y (INVX3TS)		0.25	4.27 f
FPU_Multiplication_Function/U2192/Y (OAI22X1TS)		0.39	4.66 r
FPU_Multiplication_Function/U2193/S (CMPR32X2TS)		0.75	5.42 f
FPU_Multiplication_Function/mult_x_22/U195/ S(CMPR42X2TS)		0.98	6.39 f
FPU_Multiplication_Function/mult_x_22/U194/ S(CMPR42X1TS)		1.19	7.58 f
FPU_Multiplication_Function/U2107/Y(NAND2X1TS)		0.59	8.17 r
FPU_Multiplication_Function/U2109/Y(OAI21X1TS)		0.51	8.68 f
FPU_Multiplication_Function/U2110/Y(AOI21X2TS)		0.33	9.02 r
FPU_Multiplication_Function/U2111/Y(OAI21X4TS)		0.24	9.26 f
FPU_Multiplication_Function/U2121/Y(AOI21X4TS)		0.29	9.55 r
FPU_Multiplication_Function/U726/Y(OAI21X4TS)		0.22	9.77 f
FPU_Multiplication_Function/U725/Y(AOI21X2TS)		0.29	10.06 r
FPU_Multiplication_Function/U1062/Y(XOR2X2TS)		0.28	10.34 r
FPU_Multiplication_Function/Sgf_operation/ SgfM_mul_0.left/pdt_int_reg[23]/D(DFFQX1TS)		0.00	10.34 r
data arrival time			10.34
clock CLK (rise edge)		10.00	10.00
clock network delay (ideal)		1.00	11.00
clock uncertainty		-0.50	10.50
FPU_Multiplication_Function/Sgf_operation/ SgfM_mul_0.left/pdt_int_reg[23]/CK(DFFQX1TS)		0.00	10.50 r
library setup time		-0.16	10.34
data required time			10.34

data required time	10.34
data arrival time	-10.34
<hr/>	
slack (MET)	0.00

Startpoint : FPU_Add_Subtract_Function/Oper_Start_in_module/
ASRegister/Q_reg[0] (rising edge-triggered flip-flop
clocked by clk)

Endpoint : FPU_Add_Subtract_Function/Add_Subt_Sgf_module/
Add_overflow_Result/Q_reg[0]
(rising edge-triggered flip-flop clocked by clk)

Des/Clust/Port	Wire Load Model	Library		
FloatingPointUnit	ibm13_wl10	scx3_cmos8rf_lpvt_tt_1p2v_25c		
Point			Incr	Path
clock CLK (rise edge)			0.00	0.00
clock network delay (ideal)			1.00	1.00
FPU_Add_Subtract_Function/Oper_Start_in_module/ ASRegister/Q_reg[0]/CK (DFFRX1TS)			0.00	1.00 r
FPU_Add_Subtract_Function/Oper_Start_in_module/ ASRegister/Q_reg[0]/Q (DFFRX1TS)			1.17	2.17 f
FPU_Add_Subtract_Function/U1634/Y (CLKXOR2X4TS)			0.60	2.77 f
FPU_Add_Subtract_Function/U1102/Y (XOR2X2TS)			0.40	3.17 r
FPU_Add_Subtract_Function/U1130/Y (NAND2X4TS)			0.43	3.59 f
FPU_Add_Subtract_Function/U1835/Y (INVX4TS)			0.46	4.05 r
FPU_Add_Subtract_Function/U1839/Y (XOR2X1TS)			0.68	4.73 r
FPU_Add_Subtract_Function/U1848/Y (INVX2TS)			0.39	5.11 f
FPU_Add_Subtract_Function/U1849/Y (NOR2X1TS)			0.61	5.72 r
FPU_Add_Subtract_Function/U1852/Y (AOI21X2TS)			0.50	6.23 f
FPU_Add_Subtract_Function/U871/Y (OAI21X2TS)			0.43	6.65 r
FPU_Add_Subtract_Function/U1436/Y (AOI21X2TS)			0.38	7.04 f
FPU_Add_Subtract_Function/U1913/Y (OAI21X4TS)			0.31	7.34 r
FPU_Add_Subtract_Function/U1917/Y (AOI21X4TS)			0.22	7.57 f
FPU_Add_Subtract_Function/U1921/Y (OAI21X4TS)			0.27	7.83 r
FPU_Add_Subtract_Function/U1925/Y (AOI21X4TS)			0.22	8.06 f
FPU_Add_Subtract_Function/U1929/Y (OAI21X4TS)			0.27	8.32 r
FPU_Add_Subtract_Function/U1933/Y (AOI21X4TS)			0.22	8.55 f
FPU_Add_Subtract_Function/U1937/Y (OAI21X4TS)			0.27	8.82 r
FPU_Add_Subtract_Function/U1941/Y (AOI21X4TS)			0.22	9.04 f
FPU_Add_Subtract_Function/U1945/Y (OAI21X4TS)			0.25	9.29 r
FPU_Add_Subtract_Function/U1962/Y (AOI21X2TS)			0.21	9.50 f
FPU_Add_Subtract_Function/U895/Y (XOR2X2TS)			0.28	9.78 r

FPU_Add_Subtract_Function/U1963/Y (CLKMX2X2TS)	0.41	10.20	r
FPU_Add_Subtract_Function/Add_Subt_Sgf_module/ Add_overflow_Result/Q_reg[0]/D (DFFRX2TS)	0.00	10.20	r
data arrival time		10.20	
clock clk (rise edge)	10.00	10.00	
clock network delay (ideal)	1.00	11.00	
clock uncertainty	-0.50	10.50	
FPU_Add_Subtract_Function/Add_Subt_Sgf_module/ Add_overflow_Result/Q_reg[0]/CK (DFFRX2TS)	0.00	10.50	r
library setup time	-0.30	10.20	
data required time		10.20	
data required time		10.34	
data arrival time		-10.34	
slack (MET)		0.00	

Listado 5.5: Reporte de temporizado del microprocesador ASP post síntesis lógica.

```

*****
Report : timing -path full -delay max -max_paths 1
Design : FloatingPointUnit Version: L-2016.03-SP3
*****
Operating Conditions: tt_1p2v_25c
Library: scx3_cmos8rf_lpvt_tt_1p2v_25c
    Parasitic source      : LPE
    Parasitic mode       : RealRC
    Extraction mode      : MIN_MAX
    Extraction derating  : 25/25/25

Startpoint: FPU_Multiplication_Function/Operands_load_reg/
XMRegister/Q_reg[10](rising edge-triggered flip-flop clocked
by CLK)
Endpoint: FPU_Multiplication_Function/Sgf_operation/
SgfM_mul_0.middle/pdt_int_reg[15](rising edge-triggered
flipflop clocked by CLK)

Path Group: clk | Path Type: max
-----
Des/Clust/Port      Wire Load Model      Library
FloatingPointUnit  ibm13_wl10           scx3_cmos8rf_lpvt_tt_1p2v_25c
-----
Point              Incr      Path
clock CLK (rise edge)  0.00     0.00

```

clock network delay (ideal)	1.00	1.00	
FPU_Multiplication_Function/Operands_load_reg/ XMRegister/Q_reg[10]/CK (DFFRX4TS)	0.00	1.00	r
FPU_Multiplication_Function/Operands_load_reg/ XMRegister/Q_reg[10]/QN (DFFRX4TS)	0.88	1.88	f
FPU_Multiplication_Function/U1092/Y (INVX2TS)	0.44	& 2.32	r
FPU_Multiplication_Function/U1766/Y (CLKXOR2X2TS)	0.58	& 2.90	f
FPU_Multiplication_Function/U1767/Y (NOR2X2TS)	0.18	& 3.08	r
FPU_Multiplication_Function/U1211/Y (XOR2X1TS)	0.49	& 3.56	r
FPU_Multiplication_Function/U1191/Y (NAND2X6TS)	0.35	& 3.92	f
FPU_Multiplication_Function/U2505/Y (OAI22X1TS)	0.59	& 4.51	r
FPU_Multiplication_Function/U1066/S (CMPR32X2TS)	1.06	& 5.57	f
FPU_Multiplication_Function/DP_OP_110J1_122_5487/ U307/CO (CMPR42X1TS)	1.15	& 6.73	f
FPU_Multiplication_Function/DP_OP_110J1_122_5487/ U303/S (CMPR42X2TS)	1.01	& 7.74	f
FPU_Multiplication_Function/U1593/Y (NOR2X4TS)	0.18	& 7.92	r
FPU_Multiplication_Function/U1594/Y (NOR2X2TS)	0.14	& 8.06	f
FPU_Multiplication_Function/U1596/Y (AOI21X4TS)	0.33	& 8.39	r
FPU_Multiplication_Function/U408/Y (INVX2TS)	0.22	& 8.61	f
FPU_Multiplication_Function/U1607/Y (AOI21X1TS)	0.22	& 8.82	r
FPU_Multiplication_Function/U1051/Y (XOR2X1TS)	0.46	& 9.28	r
FPU_Multiplication_Function/Sgf_operation/ SgfM_mul_0.middle/pdt_int_reg[15]/D (DFFQX1TS)	0.00	& 9.28	r
data arrival time		9.28	
clock CLK (rise edge)	10.00	10.00	
clock network delay (ideal)	1.00	11.00	
clock uncertainty	-0.50	10.50	
FPU_Multiplication_Function/Sgf_operation/ SgfM_mul_0.middle/pdt_int_reg[15]/CK (DFFQX1TS)	0.00	10.50	r
library setup time	-0.27	10.23	
data required time		10.23	
data required time		10.23	
data arrival time		-9.28	
slack (MET)		0.94	
Startpoint : FPU_Add_Subtract_Function/Oper_Start_in_module/ YRegister/Q_reg[31](rising edge-triggered flip-flop clocked by clk)			
Endpoint : FPU_Add_Subtract_Function/Add_Subt_Sgf_module/ Add_Subt_Result/Q_reg[24](rising edge-triggered flip-flop clocked by clk)			

Path Group: clk Path Type: max		
Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	1.00	1.00
FPU_Add_Subtract_Function/Oper_Start_in_module/ YRegister/Q_reg[31]/CK (DFFRX1TS)	0.00	1.00 r
FPU_Add_Subtract_Function/Oper_Start_in_module/ YRegister/Q_reg[31]/Q (DFFRX1TS)	1.10	2.10 f
FPU_Add_Subtract_Function/U1634/Y (CLKXOR2X4TS)	0.45 &	2.55 f
FPU_Add_Subtract_Function/U1102/Y (XOR2X2TS)	0.35 &	2.91 r
FPU_Add_Subtract_Function/U1130/Y (NAND2X4TS)	0.34 &	3.25 f
FPU_Add_Subtract_Function/U1129/Y (INVX8TS)	0.17 @	3.42 r
FPU_Add_Subtract_Function/U1885/Y (XOR2X1TS)	0.43 @	3.85 r
FPU_Add_Subtract_Function/U1901/Y (NAND2X1TS)	0.43 &	4.28 f
FPU_Add_Subtract_Function/U880/Y (OAI21XLTS)	0.45 &	4.73 r
FPU_Add_Subtract_Function/U1903/Y (AOI21X1TS)	0.25 &	4.98 f
FPU_Add_Subtract_Function/U1904/Y (OAI21X2TS)	0.15 &	5.13 r
FPU_Add_Subtract_Function/U1436/Y (AOI21X2TS)	0.16 &	5.29 f
FPU_Add_Subtract_Function/U1913/Y (OAI21X4TS)	0.21 &	5.50 r
FPU_Add_Subtract_Function/U1917/Y (AOI21X4TS)	0.15 &	5.65 f
FPU_Add_Subtract_Function/U1921/Y (OAI21X4TS)	0.17 &	5.82 r
FPU_Add_Subtract_Function/U1925/Y (AOI21X4TS)	0.13 &	5.95 f
FPU_Add_Subtract_Function/U1929/Y (OAI21X1TS)	0.41 &	6.36 r
FPU_Add_Subtract_Function/U1933/Y (AOI21X4TS)	0.27 &	6.63 f
FPU_Add_Subtract_Function/U1937/Y (OAI21X1TS)	0.46 &	7.09 r
FPU_Add_Subtract_Function/U1941/Y (AOI21X4TS)	0.30 &	7.39 f
FPU_Add_Subtract_Function/U2006/Y (XOR2X1TS)	0.58 &	7.97 r
FPU_Add_Subtract_Function/U905/Y (CLKMX2X2TS)	0.60 &	8.57 r
FPU_Add_Subtract_Function/Add_Subt_Sgf_module/ Add_Subt_Result/Q_reg[24]/D (DFFRX2TS)	0.00 &	8.57 r
data arrival time		8.57
clock clk (rise edge)	10.00	10.00
clock network delay (ideal)	1.00	11.00
clock uncertainty	-0.50	10.50
FPU_Add_Subtract_Function/Add_Subt_Sgf_module/ Add_Subt_Result/Q_reg[24]/CK (DFFRX2TS)	0.00	10.50 r
library setup time	-0.28	10.22
data required time		10.22
data required time		10.22
data arrival time		-8.57

Listado 5.6: Reporte STA de la FPU. Anotaciones sobre el análisis entre registros y puertos.

```

*****
Report : timing
        -path_type full
        -delay_type max
        -input_pins
        -nets
        -max_paths 1
        -transition_time
        -capacitance
        -sort_by slack
Design : FloatingPointUnit
Version: K-2015.06-SP3-3
Date   : Mon Feb 13 16:21:37 2017
*****

Startpoint: FPU_Multiplication_Function/Operands_load_reg/
XMRegister/Q_reg[10]
(rising edge-triggered flip-flop clocked by CLK)
Endpoint: FPU_Multiplication_Function/Sgf_operation/
SgfM_mul_0.middle/pdt_int_reg[15]
(rising edge-triggered flip-flop clocked by CLK)

Path Group: CLK          Path Type: max

Point          Fanout  Cap      Trans  Incr    Path
-----
clock CLK (rise edge)      ---      ---      0.50   0.00   0.00
clock network delay (ideal) ---      ---      ----- 1.00   1.00
Operands_load_reg/XMRegister/
Q_reg[10]/CK (DFFRX4TS)    ---      ---      0.00   0.00   1.00 r
Operands_load_reg/XMRegister/
Q_reg[10]/QN (DFFRX4TS)    ---      ---      0.00   0.88 * 1.88 f
n576 (net)                 2        0.01      -----
U1092/A (INVX2TS)          ---      ---      0.00   0.00 * 1.88 f
U1092/Y (INVX2TS)          ---      ---      0.00   0.44 * 2.32 r
n435 (net)                 9        0.08      -----
U1766/A (CLKXOR2X2TS)      ---      ---      0.00   0.00 * 2.32 r
U1766/Y (CLKXOR2X2TS)      ---      ---      0.00   0.58 * 2.90 f
n1110 (net)                2        0.01      -----
U1767/B (NOR2X2TS)         ---      ---      0.00   0.00 *2.90 f
U1767/Y (NOR2X2TS)         ---      ---      0.00   0.18 * 3.08 r
n1108 (net)                1        0.01      -----

```


U1211/A (XOR2X1TS)	—	—	0.00	0.00	*	3.08	r
U1211/Y (XOR2X1TS)	—	—	0.00	0.49	*	3.56	r
n1116 (net)	1	0.02	—	—	—	—	—
U1191/A (NAND2X6TS)	—	—	0.00	0.00	*	3.56	r
U1191/Y (NAND2X6TS)	—	—	0.00	0.35	*	3.92	f
n1906 (net)	10	0.04	—	—	—	—	—
U2505/B0 (OAI22X1TS)	—	—	0.00	0.00	*	3.92	f
U2505/Y (OAI22X1TS)	—	—	0.00	0.59	*	4.51	r
n1910 (net)	1	0.02	—	—	—	—	—
U1066/B (CMPR32X2TS)	—	—	0.00	0.00	*	4.51	r
U1066/S (CMPR32X2TS)	—	—	0.00	1.06	*	5.58	f
DP_OP_110J1_122_5487/n332(net)	1	0.01	—	—	—	—	—
DP_OP_110J1_122_5487/ U307/B (CMPR42X1TS)	—	—	0.00	0.00	*	5.58	f
DP_OP_110J1_122_5487/ U307/CO (CMPR42X1TS)	—	—	0.00	1.15	*	6.73	f
DP_OP_110J1_122_5487/n329(net)	1	0.01	—	—	—	—	—
DP_OP_110J1_122_5487/U303/ C (CMPR42X2TS)	—	—	0.00	0.00	*	6.73	f
DP_OP_110J1_122_5487/U303/ S (CMPR42X2TS)	—	—	0.00	1.01	*	7.74	f
DP_OP_110J1_122_5487/n319(net)	2	0.02	—	—	—	—	—
U1593/A (NOR2X4TS)	—	—	0.00	0.00	*	7.74	f
U1593/Y (NOR2X4TS)	—	—	0.00	0.18	*	7.92	r
n1671 (net)	3	0.02	—	—	—	—	—
U1594/A (NOR2X2TS)	—	—	0.00	0.00	*	7.92	r
U1594/Y (NOR2X2TS)	—	—	0.00	0.14	*	8.06	f
n976 (net)	1	0.01	—	—	—	—	—
U1596/A1 (AOI21X4TS)	—	—	0.00	0.00	*	8.06	f
U1596/Y (AOI21X4TS)	—	—	0.00	0.33	*	8.39	r
n1104 (net)	2	0.04	—	—	—	—	—
U408/A (INVX2TS)	—	—	0.00	0.00	*	8.39	r
U408/Y (INVX2TS)	—	—	0.00	0.22	*	8.61	f
n1669 (net)	4	0.01	—	—	—	—	—
U1607/A0 (AOI21X1TS)	—	—	0.00	0.00	*	8.61	f
U1607/Y (AOI21X1TS)	—	—	0.00	0.22	*	8.83	r
n986 (net)	1	0.00	—	—	—	—	—
U1051/A (XOR2X1TS)	—	—	0.00	0.00	*	8.83	r
U1051/Y (XOR2X1TS)	—	—	0.00	0.46	*	9.28	r
Sgf_operation/SgfM_mul_0.middle/ N15(net)	1	0.02	—	—	—	—	—
Sgf_operation/SgfM_mul_0.middle/ pdt_int_reg[15]/D (DFFQX1TS)	—	—	0.00	0.00	*	9.29	r
data arrival time	—	—	—	—	—	9.29	—

clock CLK (rise edge)	---	---	0.50	10.00	10.00
clock network delay (ideal)	---	---	---	1.00	11.00
clock reconvergence pessimism	---	---	---	0.00	11.00
clock uncertainty	---	---	---	-0.50	10.50
Sgf_operation/SgfM_mul_0.middle/ pdt_int_reg[15]/CK (DFFQX1TS)	---	---	---	---	10.50 r
library setup time	---	---	---	-0.27 *	10.23
data required time	---	---	---	---	10.23
<hr/>					
data required time	---	---	---	---	10.23
data arrival time	---	---	---	---	-9.29
<hr/>					
slack (MET)	---	---	---	---	0.94
Note: All the points starts from FPU_Multiplication_Function/					

5.0.2 Reportes del análisis de temporización estática (STA)

El reporte 5.7 se encuentra vacío y se debe a que la herramienta no encontró retardos significativos en la propagación de los datos, desde: los puertos de entrada a los registros, de los registros a los puertos de salida o entre registros, por lo tanto estos reportes son despreciables, para este diseño en particular.

Listado 5.7: Reporte STA de la FPU. Anotaciones sobre el análisis entre registros y puertos.

```
*****
Report : timing
-path_type full
-delay_type max
-slack_lesser_than 0.00
-max_paths 40
-sort_by slack
Design : FloatingPointUnit
Version: K-2015.06-SP3-3
Date   : Mon Feb 13 16:21:27 2017
*****

No paths with slack less than 0.00.
```

Cabe destacar que para el reporte post implementación física (5.5) se analizan 2 rutas críticas tanto para el bloque de suma/resta como para el bloque de multiplicación; sin embargo, el reporte del análisis STA únicamente corresponde al bloque de multiplicación, esto puede deberse a que el módulo de suma implementa una arquitectura optimizada de pipeline [25], mientras que el módulo de multiplicación no, es por ello que la herramienta de STA se concentra más en este módulo. En general en todos los casos el *SLACK* es alcanzado, lo cual indica una correcta implementación.

Funcionalidad

Dado que en esta sección se analiza el comportamiento de módulos matemáticos, que en esencia brindan el resultado de operar dos datos con una función particular (suma, resta o multiplicación, con signo). Conviene usar una estrategia de verificación que tome la unidad bajo prueba y la estimule con vectores de datos leídos desde archivos fuente, y generar un vector de respuesta, con, valga la redundancia, los resultados de operar los elementos de los vectores de estímulo, y guardar este vector en otro archivo. Generar un vector con las respuestas a un estímulo determinado, permite comparar con mayor facilidad si el diseño ejecuta la función para la que fue concebido después del proceso de síntesis.

Dicho lo anterior este apartado se considera más productivo mostrar un análisis gráfico de los resultados de las simulaciones, pero no mediante diagramas de tiempo, si no realizando un contraste entre las respuestas a las distintas simulaciones, aprovechando el hecho de que la estrategia de simulación usada, en el testbench (aportado junto al RTL de la FPU), consiste en estimular el diseño con un vector de datos leídos desde un archivo fuente, y guardando el vector de respuesta en otro archivo de texto. Mediante la herramienta *Octave* es posible graficar la relación entre los datos.

El testbench para la FPU, ejercita una serie de pruebas para verificar que las funciones de los módulos son correctos, tiene dos vectores de estímulo, y mediante un bucle se recorren los vectores estimulando el diseño y almacenando el resultado en un vector, en cada iteración. En total se ejecutan unas mil pruebas.

Se muestra el resultado de esas mil pruebas elaborando un gráfico de la siguiente manera: Se presentan dos figuras con tres gráficas (figuras 5.2 y 5.2), con el fin de no realizar demasiada contaminación visual, se les da un desnivel (*offset*) a los datos correspondientes a las simulaciones post síntesis, para verificar que tan similares son los resultados. El eje horizontal representa el índice de la muestra del vector de resultados, y el eje vertical muestra la magnitud, es decir el valor del resultado.

En las figuras 5.2 y 5.3 se puede observar el comportamiento del diseño ante una serie de estímulos, en esencia, datos de entrada para verificar si el algoritmo matemático es correcto. Tanto los vectores de estímulo como el vector de respuesta del banco de pruebas (testbench) se encuentran en formato hexadecimal IEEE574. Por lo que se realizó un pequeño script en *Octave* que convierte del formato IEEE574 a representación decimal estándar.

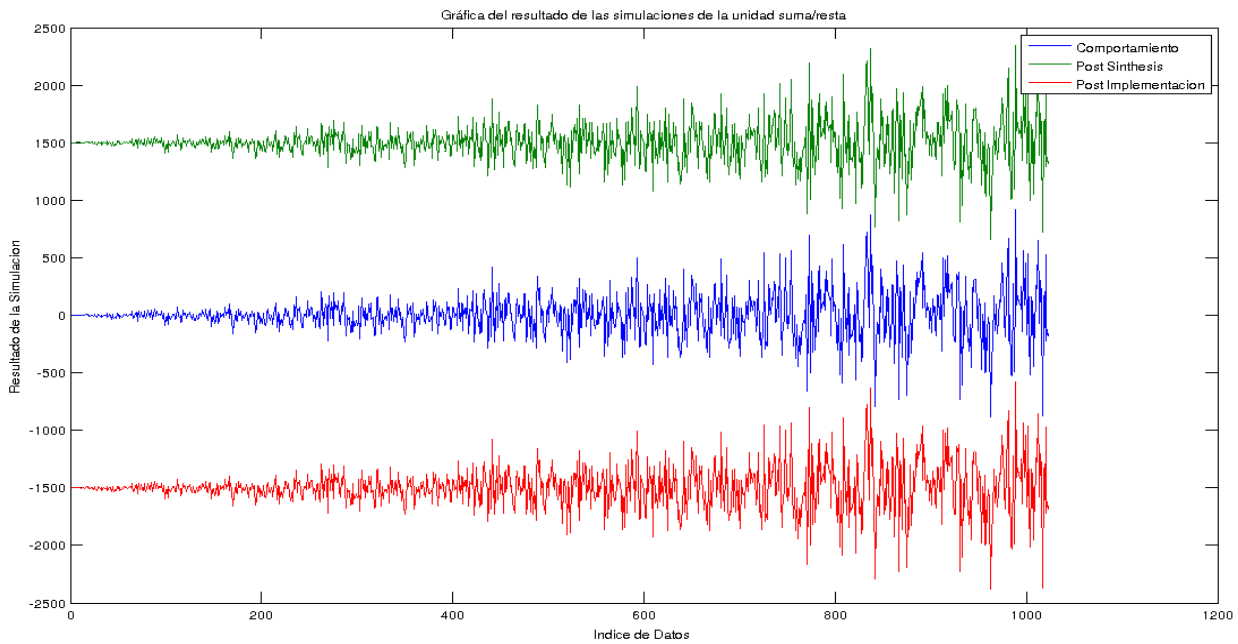


Figura 5.2: Gráfico de los resultados de las distintas simulaciones asociadas con el módulo de suma y resta de la FPU. Observar que hay un desfase entre los datos, hecho a drede para apreciar mejor la dinámica de las distintas gráficas.

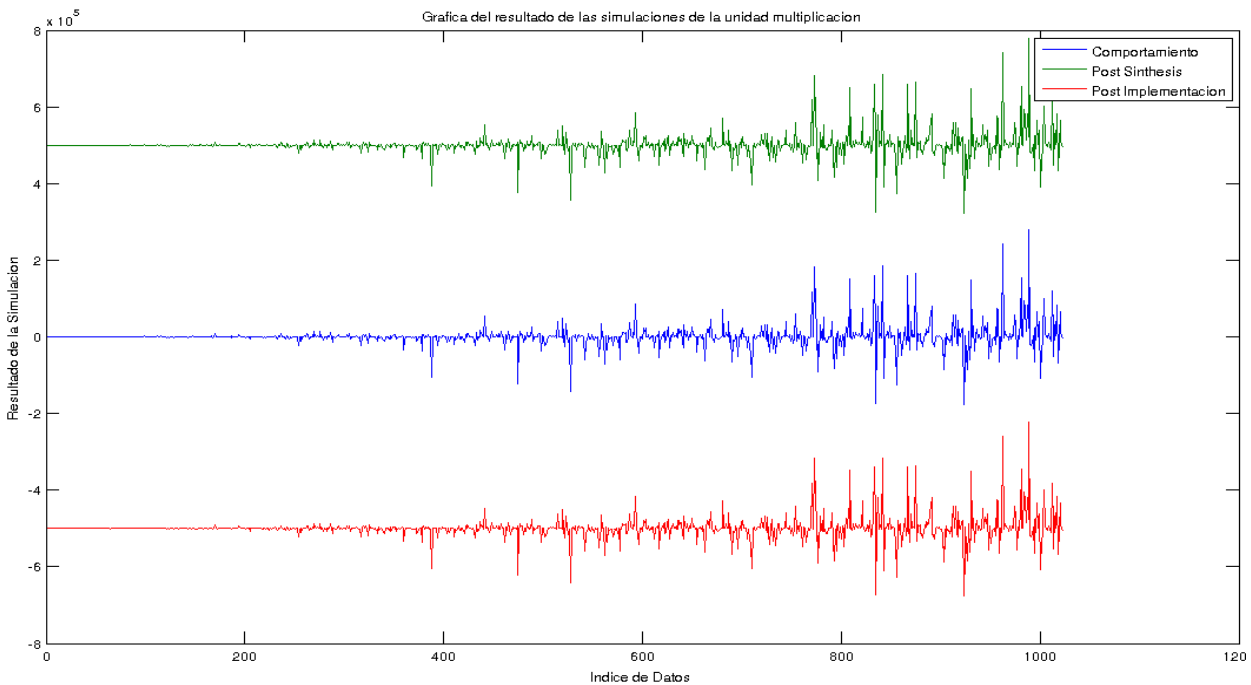


Figura 5.3: Gráfico de los resultados de las distintas simulaciones asociadas con el módulo de multiplicación con signo de la FPU. Observar que hay un desfase entre los datos, hecho a drede para apreciar mejor la dinámica de las distintas gráficas.

Como complemento a estas gráficas también se genera otro par de gráficas que muestran que tan desviados se encuentran los datos de las simulaciones post síntesis respecto a los esperables según la simulación por comportamiento, que se ha tomado como la referenciadora del diseño, en ausencia a otro modelo de alto nivel para tomar como referencia.

Las gráficas permiten dilucidar que tanto los módulos de suma y resta funcionan adecuadamente, pues presentan el mismo patrón que la simulación por comportamiento, la cual se toma como la referencia dorada para evaluar los resultados de la implementación.

Cabe resaltar la particularidad que se muestra en la figura 5.4, en esta figura se muestra una relación porcentual de desviación entre el dato de la referencia dorada y el dato obtenido en la simulación post síntesis, la gráfica correspondiente al resultado post síntesis lógica diverge enormemente para algunos estímulos puntuales, ubicados cerca de la mitad del vector de estímulo; sin embargo, la variabilidad de los datos de la simulación post síntesis física no es significativa, lo cual lleva a considerar que hay algún problema en el modelado de la síntesis lógica, probablemente por no definir adecuadamente las restricciones de diseño para el reloj durante el flujo de front end. Dado que las simulaciones post síntesis física y los reportes de temporizado indican que el diseño funciona adecuadamente, puede considerarse que el diseño es correcto.

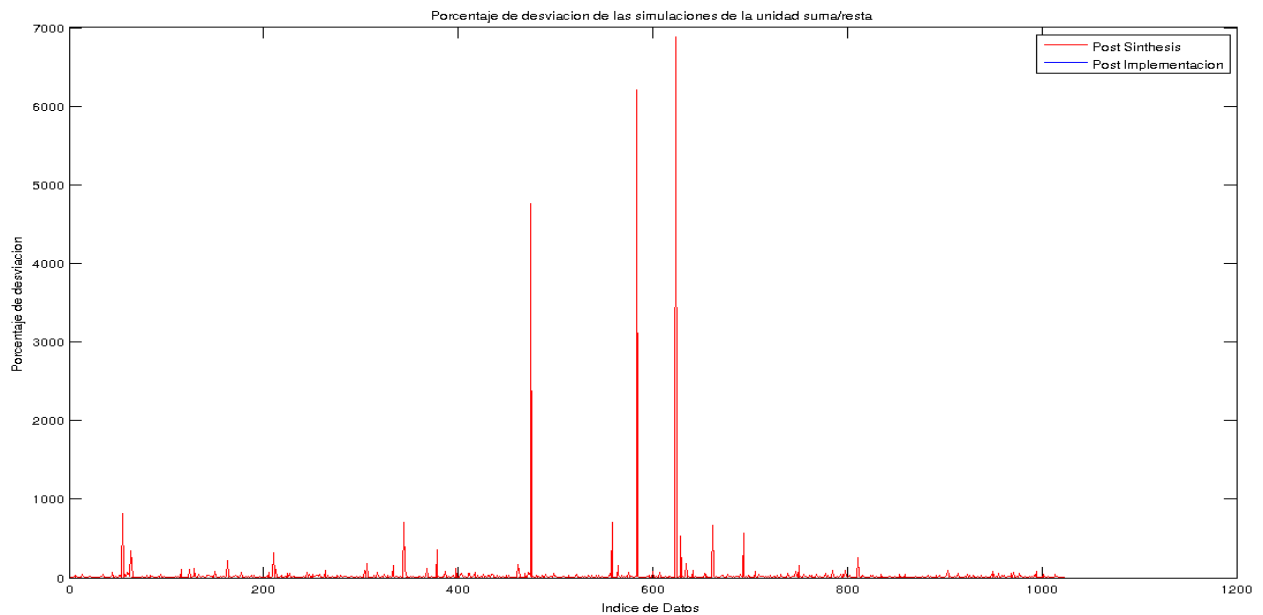


Figura 5.4: Gráfica del porcentaje de desviación, de las simulaciones post síntesis de la unidad de suma y resta de la FPU, respecto a simulación por comportamiento. Se que la simulación post síntesis lógica, presenta algunas incongruencias para ciertos datos, mientras la síntesis física es congruente.

Respecto a la multiplicación se obtuvieron datos consistentes en todas las simulaciones, la figura 5.3 muestra que todas las gráficas se superponen, y en como se aprecia en la figura 5.5 la desviación es nula.

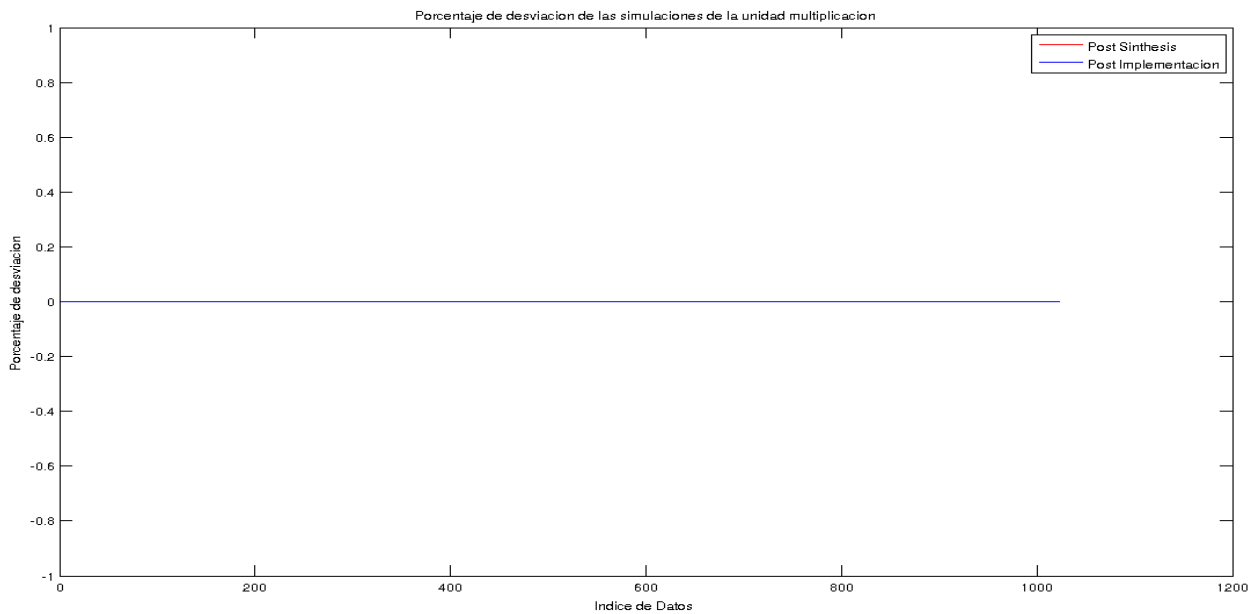


Figura 5.5: Gráfica que muestra el porcentaje de desviación, de las simulaciones post síntesis de la unidad de multiplicación de la FPU. Observe que ambas líneas son perfectamente horizontales, lo que indica que la simulaciones post síntesis son consistentes con la de comportamiento

5.1 Integración de los *IP Cores* de memorias SRAM

Respecto a los *IP Cores*, cabe destacar que se generaron 2 bloques de memoria SRAM de 4000 palabras y 2 palabras con un tamaño de palabra de 32 bits, con el fin de crear un precedente para la posterior integración correcta de los bloques de memoria para el microprocesador ASP. Los bloques fueron definidos con un tamaño arbitrario, considerando que un bloque de 4k es suficiente para la memoria de datos del ASP, y 2k son suficientes para la memoria de programa del ASP. Estas dimensiones se consideran una magnitud significativa para poder demostrar la inclusión de *IP Cores* dentro de un diseño.

La metodología seguida corresponde a la expuesta en la sección 3.4, como se mencionó al inicio de este capítulo la técnica usada para demostrar la correcta generación del *IP Core* es la de crear un wrapper, el cual en esencia es un código HDL que instancia la celda generada. Luego esta nueva celda (wrapper) se somete al flujo hasta generar un layout, donde aparece una celda estándar correspondiente al *IP Core* generado.

Es posible presentar todos los reportes asociados con la síntesis lógica y física de esta unidad; sin embargo, sería saturar de información redundante este trabajo, por lo que partiendo de la premisa de que la celda generada, debe ser correcta, pues es concebida con las herramientas propias del proveedor de la tecnología, se omite un análisis exhaustivo de esta integración.

El resultado de la integración del bloque *IP Core* correspondiente a la memoria SRAM para datos (4kx32) y el layout del wrapper usado sería el apreciado en la figura 5.6

Para demostrar la funcionalidad del *IP Core* generado se realiza una pequeña simulación post implementación física. El resultado se aprecia en las figuras 5.7 y 5.8. Puede apreciarse como el dato se escribe (en la dirección ffe) en el primer de reloj, si se presta atención al marcador M1 (línea vertical de color gris) se aprecia entre paréntesis el lapso que existe entre el flanco de la señal de reloj y el cambio en la señal de salida del dato. Este tiempo es distinto para las dos simulaciones siendo mayor en la figura 5.8 pues considera los retardos máximos del diseño, mientras que la figura 5.7 muestra el efecto de los retardos mínimos.

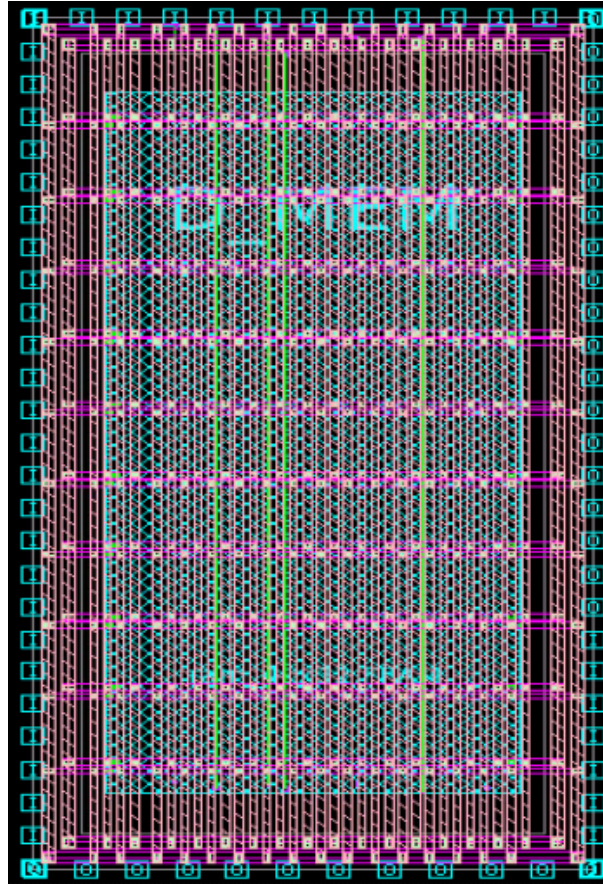


Figura 5.6: Foto captura del layout resultante de la implementación del *IP Core* de una memoria SRAM de 4kx32.

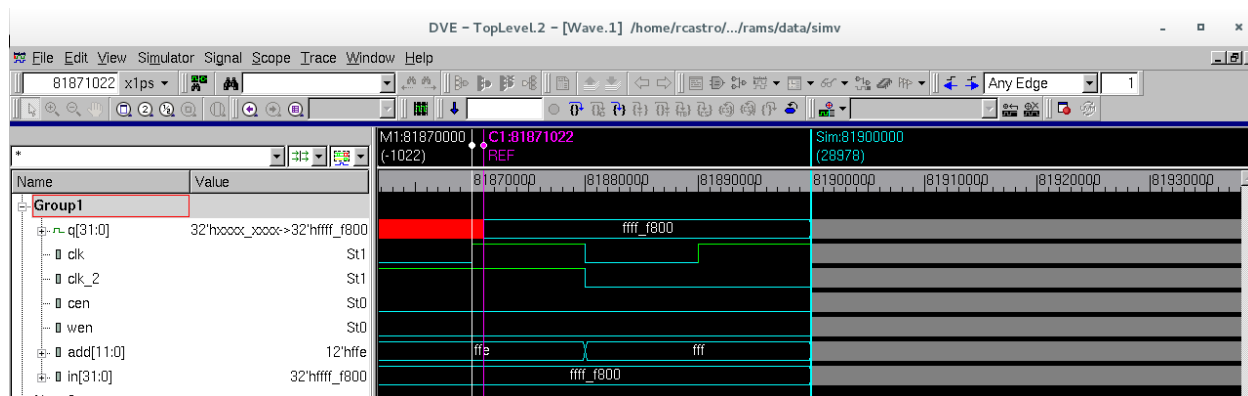


Figura 5.7: Simulación post implementación física del *IP Core* para la memoria de datos (4kx32) con el modelo de retardos mínimos

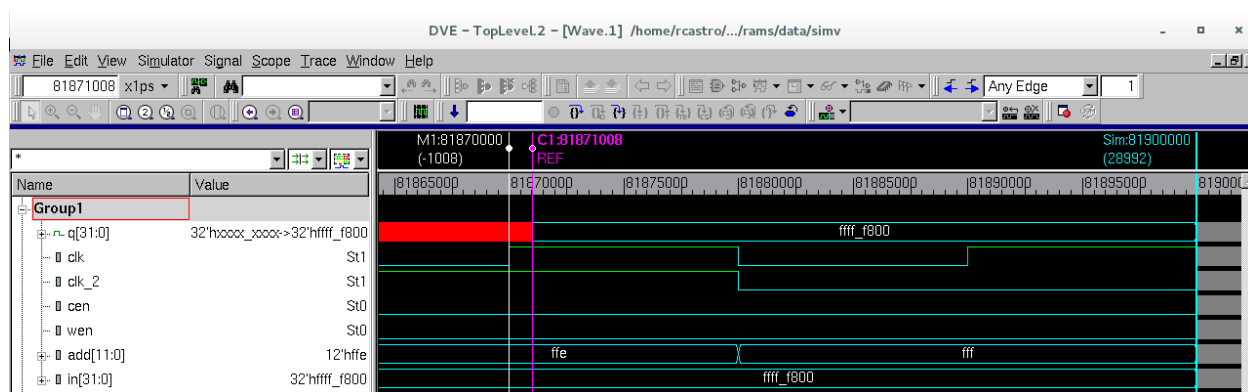


Figura 5.8: Simulación post implementación física del *IP Core* para la memoria de datos (4kx32) con el modelo de retardos típicos

Capítulo 6

Conclusiones y Recomendaciones

6.1 Conclusiones

Partiendo de una estructura básica para la integración de diseños codificados en RTL se ha logrado diseñar una jerarquía de archivos y directorios que permite implementar exitosamente un flujo de diseño de circuitos integrados digitales.

Mediante el uso de esta jerarquía fue posible implementar una unidad aritmética de punto flotante y demostrar que es correcta y funcional. Así mismo se demostró que es posible incorporar *IP Cores* dentro del diseño y que los mismos son correctos y funcionales.

Debido a particularidades que se escapan de la visión de este trabajo no fue posible demostrar la funcionalidad del microprocesador RISC-V. Debido a deficiencias en la estructura de su RTL, sin embargo, quedó demostrado que el flujo es eficaz y eficiente.

Realizar una evaluación cuantitativa sobre la estructura de *scripting* diseñada no es posible ya que responde a la visión de una única persona, y su paradigma particular de trabajo. Sin embargo, desde una perspectiva cualitativa y subjetiva, se considera la estructura de *scripting* eficiente, pues manifiesta atributos como: regularidad, localidad y continuidad.

6.2 Recomendaciones

Se recomienda realizar una iteración en el diseño del microprocesador ASP, enfocado a incorporar una unidad para la programación de las memorias y permitir el *booteo* del microprocesador.

Finalmente recomienda incorporar en una iteración futura optimizaciones asociadas con el área y la potencia de los módulos, incorporando técnicas como *clock gating*, las cuales no fueron utilizadas en este proyecto.

Bibliografía

- [1] SINAC. Áreas Silvestres Protegidas. [Online], Septiembre 2016.
- [2] INBio. Biodiversidad en Costa Rica. [Online], Septiembre 2016.
- [3] ICT. INFORME ESTADÍSTICO TRIMESTRAL, PRIMER SEMESTRE 2016, (IT-2016”). [Online], Septiembre 2016.
- [4] National Instruments. ¿Qué es una Red de Sensores Inalámbricos?. [Online], Septiembre 2016.
- [5] DCILab. Sonidos Ilegales. [Online], Septiembre 2016.
- [6] Tecnológico de Costa Rica. Escuela de ingeniería electrónica. itcr.: Investigación y extensión. [online], Noviembre 2015.
- [7] A. Chacón. and P. Alvarado. Sistema electrónico integrado en chip (SoC) para el reconocimiento de patrones de disparos y motosierras en una red inalámbrica de sensores para la protección ambiental. Vicerectoría de Investigación, Instituto Tecnológico de Costa Rica, Junio 2014. Proyecto No. 5402-1360-3101.
- [8] Carlos Salazar García. Implementación de un microprocesador de aplicación específica para la ejecución del algoritmo de modelos ocultos de markov en el reconocimiento de patrones acústicos. Tesis de maestría, Escuela de Ingeniería Electrónica. ITCR., Diciembre 2015.
- [9] Jordan Montero. Implementación de un sistema de reconocimiento de patrones acústicos de disparos y motosierras en un sistema embebido. Tesis de licenciatura, Escuela de Ingeniería Electrónica. ITCR., Diciembre 2015.
- [10] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press Oxford, 1995.
- [11] E. Salas. and P. Alvarado. Implementación de un banco de filtros digitales multitasa para la estimación energética espectral en una aplicación de protección ambiental. In *Convención de Centroamérica y Panamá*. XXX IEEE, Noviembre 2010.
- [12] E. Salas. and P. Alvarado. Implementation of an automatic gain control for audio signals in an application for environmental protection. In *Proceedings of the Conference on Technologies for Sustainable Development TSD2011*, Cartago, Costa Rica, 2011.

- [13] J. Cárdenas. Training strategies for hmm in the acoustic pattern recognition,. Tesis de maestría, Escuela de Ingeniería en Computación. ITCR, Mayo 2012.
- [14] M. Sequeira. and P. Alvarado. Modulo de reducción de dimensiones espectrales en un sistema empotrado nodal de una red inalámbrica de sensores. In *Proceedings of the Embedded Technology Conference*, San José, Costa Rica, 2011.
- [15] The MOSIS Service. What is mosis, 2015.
- [16] M. Sáenz. Reconocimiento de patrones acústicos para la protección del ambiente utilizando wavelets y modelos ocultos de markov. Tesis de licenciatura, Escuela de Ingeniería Electrónica. ITCR., Noviembre 2006.
- [17] E. Salas. Reconocimiento en tiempo real de patrones acústicos de motosierras y disparos por medio de una implementación en fpga de modelos ocultos de markov. Tesis de licenciatura, Escuela de Ingeniería Electrónica. ITCR., Mayo 2010.
- [18] M. Sequeira. Modulo de reducción de dimensiones espectrales en un sistema de reconocimiento de patrones acústicos de motosierras y disparos por medio de una implementación en fpga. Tesis de licenciatura, Escuela de Ingeniería Electrónica. ITCR., Junio 2011.
- [19] Beagleboard. Beagleboard-xM Rev C System Reference Manual. [Online], 2010.
- [20] L. Alfaro. Implementacion en hardware del sistema de reconocimiento de patrones acusticos (sirpa). Tesis de licenciatura, Escuela de Ingeniería Electrónica. ITCR., Junio 2013.
- [21] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.
- [22] M. Carvajal. Banco de pruebas del sistema de reconocimiento de patrones acústicos de motosierras y disparos. Tesis de licenciatura, Escuela de Ingeniería Electrónica. ITCR., Noviembre 2013.
- [23] C. Salazar-García, L. Alfaro-Hidalgo, M. Carvajal-Delgado, J. Montero-Aragón, R. Castro-Gonzalez, J. A. Rodríguez, A. Chacon-Rodríguez, and P. Alvarado-Moya. Digital integrated circuit implementation of an identification stage for the detection of illegal hunting and logging. In *Circuits Systems (LASCAS), 2015 IEEE 6th Latin American Symposium on*, pages 1–4, Feb 2015.
- [24] Diego Andrés Rodríguez Valverde. Diseño e implementación de una unidad aritmético-lógica de coma flotante para un procesador de aplicación específica. Tesis de licenciatura, Escuela de Ingeniería Electrónica. ITCR., Diciembre 2015.
- [25] Francis Alexander López Montero. Diseño e implementación de hardware para optimizar la unidad aritmética de coma flotante de un procesador de aplicación específica. Tesis de licenciatura, Escuela de Ingeniería Electrónica. ITCR., Junio 2016.

-
- [26] Michael John and Sebastian Smith. *Application-specific integrated circuits*, 1997.
- [27] Keith Barr. *ASIC Design in the Silicon Sandbox: A Complete Guide to Building Mixed-Signal Integrated Circuits*. McGraw-Hill Professional, 1 edition, 2006.
- [28] A Albert Raj and T Latha. *VLSI design*. PHI Learning Pvt. Ltd., 2008.
- [29] Neil Weste and David Harris. *Cmos VLSI Design. A circuits and systems perspective*. Addison-Wesley, 4 edition, 2005.
- [30] Larisa N Solovéva and Marina Marder. *Russian matryoshka*. Interbook, 1993.
- [31] Jairo Valverde, Dave Porras, Reinaldo Castro, Jorge Sequeira, Gabriel Loría. Tutorial para Flujo de Diseño de Circuitos Integrados Digitales con Tecnología IBM 8RF (CMOS 0.13 micrómetros), Octubre 2016.
- [32] Hima Bindu Kommuru and 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*.
- [33] José Alberto Villalobos. Entropia. *Revista de la Universidad de Costa Rica*, 30.
- [34] Salazar-Garcia C., Castro-Gonzalez R., and A. Chacon-Rodriguez. Risc-v based sound classifier intended for acoustic surveillance in protected natural environments. *Circuits & Systems (LASCAS)*, 2017 IEEE 8th Latin American Symposium, Febrero 2017.
- [35] A. Cervantes, J. Quiros, D. Rodriguez-Valverde, C. Salazar-Garcia, and A. Chacon-Rodriguez. Implementation of an open core iee 754- based fpv with non-linear arithmetic support. 2016 IEEE XXXVI, Noviembre 2016.

Apéndice A

Hoja de Información del Proyecto

Información del estudiante:

Nombre: Reinaldo D. Castro González.

Cédula: 3 0454 0969 **Carné ITCR:** 201106270.

Dirección de su residencia en época lectiva:

#531 Urb. Cocorí, San Francisco, Cartago.

Teléfono: 8400 1665 **Email:** shidarimo@gmail.com

Información del proyecto:

Nombre del Proyecto:

Integración de un microprocesador de aplicación específica en un flujo de diseño de circuitos integrados digitales..

Área del Proyecto: Investigación¹

Información de la empresa:

Nombre: Instituto Tecnológico de Costa Rica.

Zona: Cartago, Central, Oriental.

Dirección:

Un kilómetro al Sur de la Basílica de Nuestra Señora de los Ángeles (Avenida 14, Calle 15).

Teléfono: (506) 2550-5333 **Sitio web:** www.tec.ac.cr.

Actividad Principal: Docencia.

Información del asesor en la empresa: Nombre: Alfonso Chacón Rodríguez.

Puesto que ocupa: Profesor e Investigador.

Departamento: Escuela de Ingeniería Electrónica.

Profesión: Profesor **Grado académico:** Doctor.

Teléfono: 2550 9232 Ext.: 9232.

Email: alchacon@iter.ac.cr.

¹Electrónica Digital, Computación, Microelectrónica (VLSI).

