Tecnológico de Costa Rica
Sede de Alajuela

Escuela de Ingeniería Electrónica

Programa de Licenciatura en Ingeniería Electrónica

Sistema para el manejo de parámetros PVT - Etapa 2: Compilador con interfaz gráfica de usuario y control de versiones

Informe del Proyecto Final de Graduación para optar por el título de

Ingeniero en Electrónica con el grado académico de Licenciatura

Isaac Francisco Moreno Fuentes

Alajuela, 10 de noviembre de 2023

Declaro que el presente documento de proyecto de graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales 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 realizado y por el contenido del presente documento.

Isaac Francisco Moreno Fuentes

Alajuela, 10 de noviembre de 2023

Céd: 1 1767 0048

Instituto Tecnológico de Costa Rica
Sede de Alajuela
Escuela de Ingeniería Electrónica
Proyecto Final de Graduación
Acta de Aprobación

Defensa de Proyecto
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura

El Tribunal Evaluador aprueba la defensa del proyecto denominado *Sistema para el manejo de parámetros PVT - Etapa 2: Compilador con interfaz gráfica de usuario y control de versiones*, realizado por el estudiante Isaac Francisco Moreno Fuentes 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

LUIS ROBERTO PEREIRA ARROYO (FIRMA)
Firmado digitalmente por LUIS ROBERTO PEREIRA ARROYO (FIRMA)
Fecha: 2023.11.10 17:07:36 -06'00'

Dr. Roberto Pereira Arroyo
Profesor Lector

KERVIN SANCHEZ HERRERA (FIRMA)
PERSONA FISICA, CPF-02-0655-0231.
Fecha declarada: 15/11/2023 08:13:16 PM
Razón: Proyecto de Graduación Isaac

M.Sc. Kervin Sánchez Herrera
Profesor Lector

LUIS CARLOS ROSALES ALPIZAR (FIRMA)
Digitally signed by LUIS CARLOS ROSALES ALPIZAR (FIRMA)
Date: 2023.11.14 17:45:33 -06'00'

MBA. Luis Carlos Rosales Alpízar
Profesor Asesor

Alajuela, 10 de noviembre de 2023

Resumen

En una industria electrónica altamente competitiva, en la que es clave la innovación, se busca que las herramientas utilizadas sean cada vez más óptimas, tanto en el manejo de recursos materiales como humanos. Bajo este contexto se encuentra el presente proyecto de graduación, para optar por el grado de Licenciatura en Ingeniería en Electrónica en el Tecnológico de Costa Rica.

Los temas que se exploran en este proyecto son el procedimiento y las metodologías para definir una configuración de compilador capaz de producir un archivo TCL que genere una lista de parámetros de casos de prueba con un número reducido de instrucciones (en comparación con la solución existente), manteniendo al mismo tiempo la equivalencia funcional para la verificación de escenarios de proceso, voltaje y temperatura, también conocidos como corners o por las siglas PVT. El compilador integra un sistema de control de versiones que permite la importación de datos de diferentes proyectos y el almacenamiento de múltiples versiones, así como una interfaz gráfica de usuario que facilita la visualización y edición de variables, teniendo en cuenta las necesidades del cliente y múltiples factores relevantes para la verificación de corners.

Palabras clave: Corner, Temporización, Capas de configuración, VLSI, GUI, VCS, ASIC, TCL

Abstract

This document presents the graduation project, to opt for the Licentiate's degree in Electronic Engineering at the Tecnológico de Costa Rica. The main topic resulting from this project is the procedure and methodologies to define a compiler configuration capable of producing a TCL file which is capable of producing a list of test case parameters with a reduced number of instructions (compared to existing solution) yet maintaining functional equivalence for the verification of process, voltage, and temperature scenarios, also known as corners or by the acronym PVT. The compiler integrates a version control system that allows importing data from different projects and storage of multiple versions, as well as a graphical user interface which makes viewing and editing variables easier, taking into account the needs of the client and multiple factors that are relevant for the design of PVT scenarios.

Keywords: Corner, Timing, Layering, VLSI, GUI, VCS, ASIC, TCL

a mi querida familia...

Agradecimientos

Expreso mi más profundo agradecimiento a los profesores Kervin Sánchez, Luis Carlos Rosales, Hayden Phillips y Roberto Pereira por su valiosa asesoría, dedicación y el tiempo que brindaron para hacer posible este proyecto, y a todo el cuerpo docente del Tecnológico de Costa Rica que me acompañaron en el proceso de formación académica.

A su vez, quiero expresar mi agradecimiento a Intel, en especial a Emmanuel Barrantes y a su equipo de Insfraestructura por brindarme la oportunidad y el soporte para llevar a cabo esta iniciativa.

Isaac Francisco Moreno Fuentes

Alajuela, 10 de noviembre de 2023

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En una industria de dispositivos electrónicos cada vez más competitiva en la producción de chips, en la que cada año se espera dispositivos con mayor capacidad de procesamiento de datos, mejor manejo de memoria, mayor almacenamiento, entre otros; la exigencia para las empresas que los diseñan y fabrican aumenta cada vez más. Para cumplir dichas expectativas, se debe contar con herramientas que faciliten el diseño, verificación, prueba y manufactura de los circuitos integrados.

Conocer en qué escenario se va a desenvolver el chip que se piensa diseñar, es parte de las etapas iniciales de diseño ¿Va a ser utilizado en una laptop, en un servidor, en un dispositivo móvil, en una computadora de escritorio o en otro tipo de dispositivo? ¿A qué voltaje se debe operar? ¿Qué tecnología de fabricación se piensa utilizar? ¿Cuáles celdas están disponibles? ¿A qué temperaturas va a operar el dispositivo?

Para responder dichas preguntas, se debe modelar de la manera más precisa posible los escenarios de proceso, voltaje y temperatura (PVT) a los cuales se plantea operar el chip. Precisamente en este campo se ubica el proyecto, en crear una herramienta que facilite el proceso de definir dichos escenarios, a los de ahora en adelante nos referiremos a ellos simplemente como corners.

Este proyecto consiste en la implementación de un software para el diseño, verificación y compilación de parámetros de proceso, voltaje y temperatura (PVT) para Intel Corporation. El sistema debe tener como salida un archivo en lenguaje TCL, y para la compilación de este, se plantea que provenga desde otro lenguaje de programación, en nuestro caso Python, debido a las herramientas que facilita este lenguaje en cuanto a la creación de interfaces web, manejo de bases de datos, métodos para el control de versiones, facilidad de depuración y corrección de errores [1].

El compilador debe tener una serie de características para que supla las necesidades del

usuario, en nuestro caso Intel Corporation, las cuales van orientadas a mejorar la experiencia de usuario y asegurar la integridad de los datos a la hora de establecer los corners y sus múltiples parámetros, para lo que se plantea implementar control de versiones para que múltiples usuarios puedan trabajar sobre distintas versiones de los proyectos, una interfaz gráfica que facilite seleccionar, agregar, eliminar, buscar y comentar tanto corners como sus parámetros, y también optimizar el archivo en lenguaje TCL generado, esto de acuerdo a los requerimientos del cliente mostrados en 1.1.

A nivel de investigación se definen conceptos esenciales para dar contexto de los corners dentro del flujo ASIC, así como definiciones de VLSI e investigaciones relacionadas con compiladores para TCL, diseño de interfaces gráficas y métodos para el control de versiones.

1.1. Planteamiento del Problema

Para definir el problema actual y los requerimientos de la solución, Intel Corporation elaboró un volante en el cual se resumen los puntos relevantes para el proyecto PVT Parameter Managent System y el porqué del mismo.

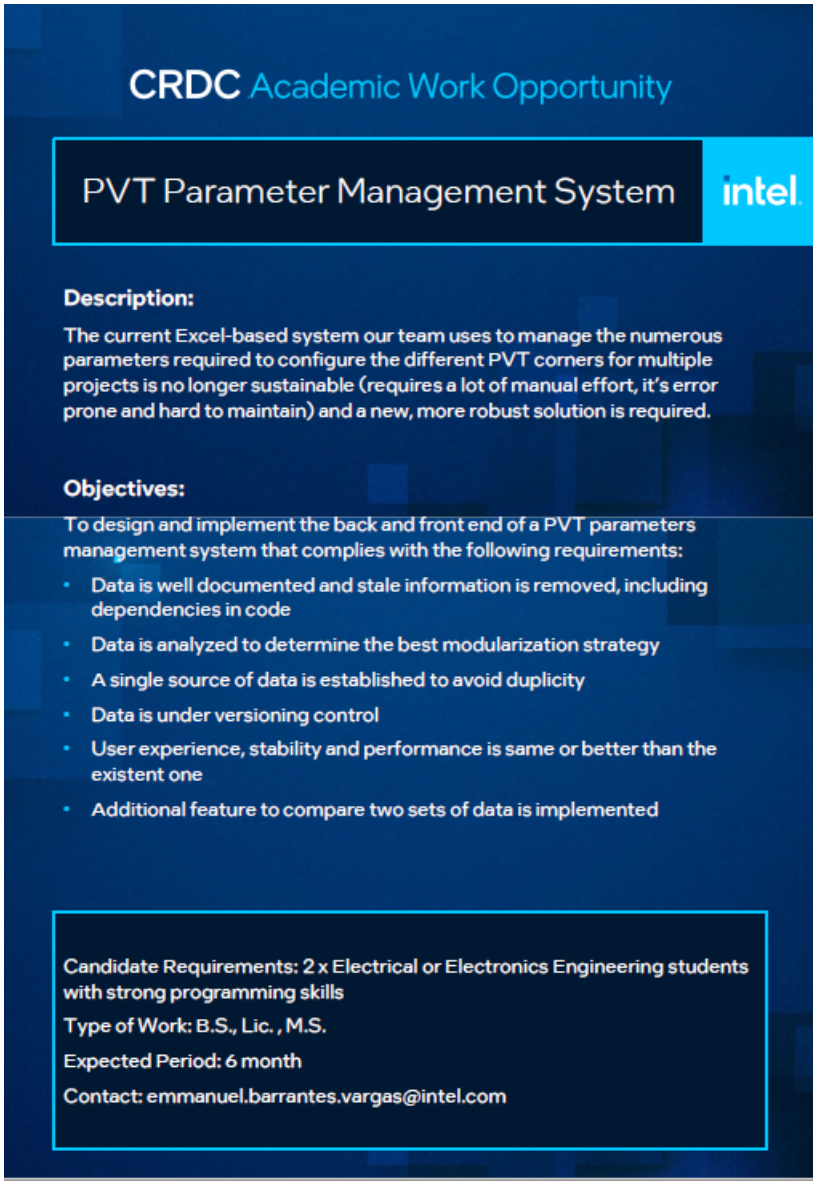


Figura 1.1: Volante del proyecto *Sistema para el manejo de parámetros PVT*

De acuerdo al cliente: "El sistema actual basado en Excel utilizado para gestionar la numerosos parámetros requeridos para configurar los distintos corners para múltiples proyectos ya no es sostenible (requiere mucho esfuerzo manual, es propenso a errores y difícil de mantener) por lo que se requiere una solución nueva y más robusta".

Dada la complejidad del proyecto *Sistema para el manejo de parámetros PVT*, como se especifica en la Figura 1.1 se requiere de dos personas para llevarlo a cabo. Por lo tanto el presente proyecto pretende resolver los puntos relacionados a la Etapa 2 de la Figura 1.2, que hace referencia al compilador, el cual integra: control de versiones, interfaz gráfica de usuario y optimización del TCL.

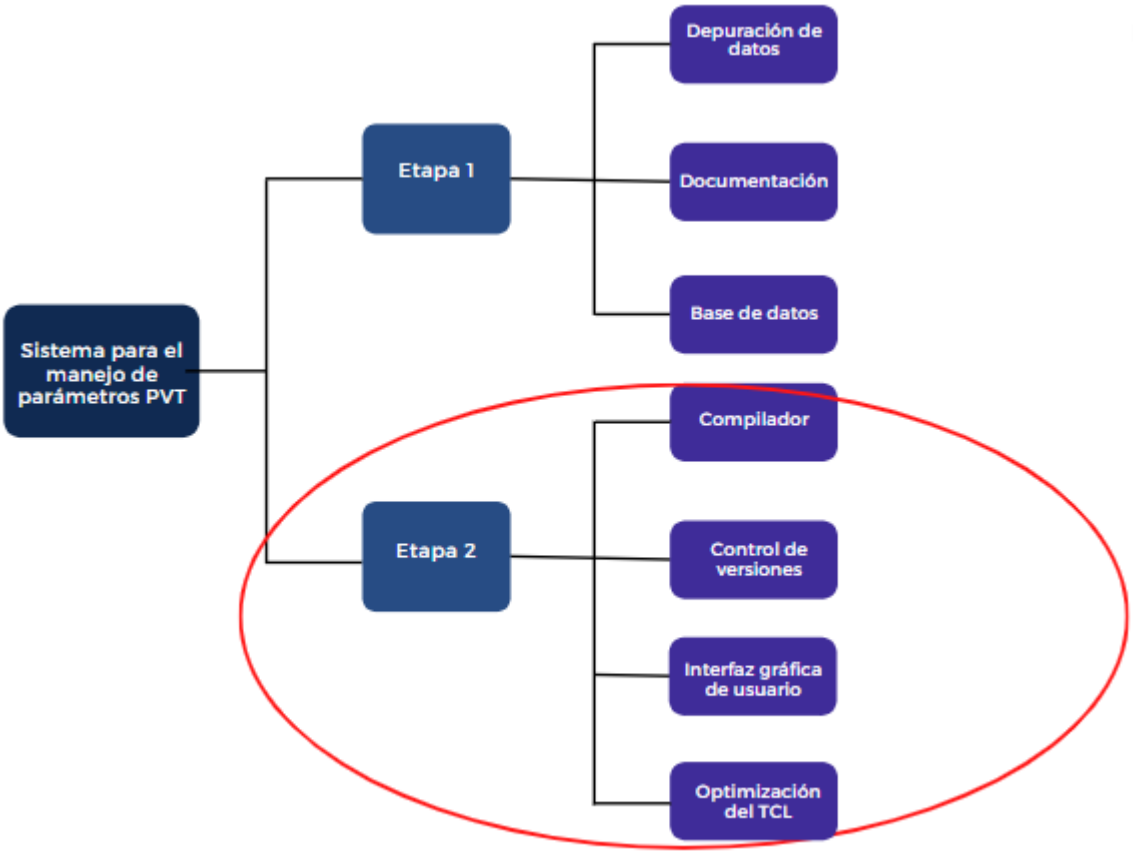


Figura 1.2: Etapas del proyecto PVT Parameter Management System

Como se muestra en la Figura 1.2 hay una Etapa 1, la cual consiste en la depuración de los datos, documentación y base de datos. La etapas se realizan en paralelo y el éxito de una etapa no depende de la otra.

Como parte de un análisis previo de los datos, para dimensionar el problema a resolver, se realizó un estudio para determinar en qué porcentaje son modificados los corners de un proyecto, lo que llevó a los resultados mostrados en la Figura 1.3.

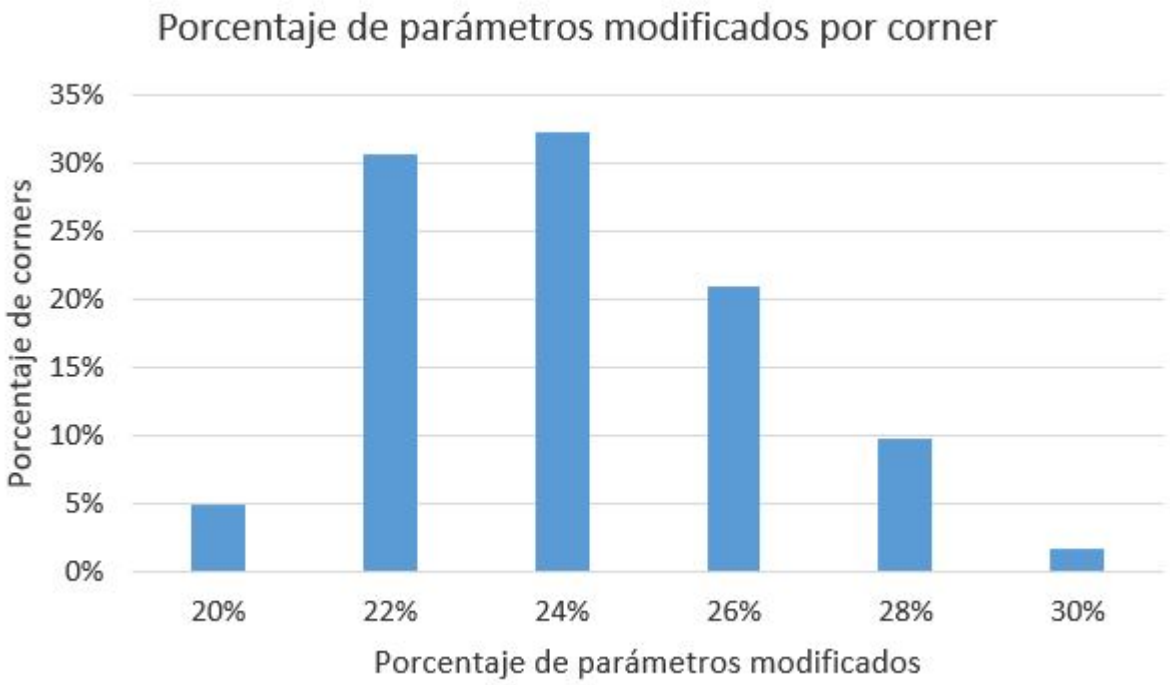


Figura 1.3: Porcentaje de corners que son modificados según la cantidad de parámetros variados

En la Figura 1.3 se observa una distribución normal con baja varianza, en la cual en el 5 % de los corners se modifican el 20 % de los parámetros, en el 31 % de los corners se modifican el 22 % de los parámetros, y una alta cantidad de corners siendo el 32 % a los cuales se les modifican el 24 % de los parámetros, posteriormente a un 21 % de los corners se les modifica el 26 % de los parámetros, un 10 % de los corners es modificado en un 28 % de los parámetros, y un 2 % de los corners poseen la mayor cantidad de cambios en sus parámetros, siendo esto en un 30 %.

La información extraída de 1.3 implica que el sistema se encuentra actualmente optimizado para que en promedio los corners tengan un 25 % de parámetros modificados, lo cual impacta directamente en las líneas de instrucciones en el compilador. Lo que se busca con el proyecto es disminuir dichas líneas de código, con base en experimentación y análisis estadísticos, optimizando así la salida en lenguaje TCL, además de aportar al usuario la posibilidad de trabajar en múltiples versiones y utilizar una interfaz que facilite el proceso de diseño.

Capítulo 2

Marco Teórico

En esta sección se muestran los fundamentos teóricos sobre el diseño de circuitos integrados VLSI, escenarios de proceso, voltaje y temperatura también conocidos como PVT o corners, y conceptos que abarcan la solución propuesta respecto a compiladores, control de versiones e interfaces gráficas de usuario GUI, así como soluciones implementadas anteriores a la investigación y el estado del arte. Esto con el fin de establecer un marco de referencia tanto de conceptos como de antecedentes para la investigación que se pretende desarrollar.

El proyecto planteado parte de información tanto confidencial como valiosa para el diseño de circuitos integrados actualmente, ya que cuenta con los escenarios específicos y tecnología en desarrollo, por ende los datos se presentan en grupos de manera estadística.

El algoritmo del compilador, puede ser visto desde dos puntos, el primero siendo las necesidades del cliente el cual requiere una herramienta robusta, eficiente y que asegure la integridad de los datos, y el segundo punto de vista es que debe ser intuitivo, rápido y amigable para el usuario.

Tomando en cuenta lo mencionado anteriormente, se definen cuatro subsecciones de investigación, la primera hace referencia a los datos manejados, los cuales son necesarios para el diseño en VLSI y establecer escenarios PVT, las tres subsecciones posteriores definen conceptos e investigaciones previas para el diseño, implementación y prueba de compiladores, control de versiones e interfaces gráficas.

2.1. Entradas para el algoritmo de compilación

Para la configuración de cada proyecto se debe establecer una base base de datos específica la cual detalla todos los escenarios en los cuales se desea modelar y optimizar el diseño. Dicha base de datos va en aumento, ya que a medida de que pasa el tiempo se necesitan nuevos procesos, métodos de fabricación, mejoras para el modelado, y la cantidad de corners; así como se describe en [2], el número de corners que necesitan ser verificados está aumentando a medida que se alcanzan tecnologías de 7 nm y posteriores, debido a factores como la temperatura, el voltaje y los cambios en el metal, además de otros parámetros que se toman en cuenta para el compilador como lo son la biblioteca de celdas, interconexión (capacitancias, resistencias e inductancias parasíticas), tipo de temporización de setup o hold, que como se define en [3], setup y hold definen una ventana dentro de la cual las señales no deben cambiar. Por otro lado, en el contexto del análisis de tiempo estático, setup define un valor antes del cual todas las transiciones deberían haber llegado, y hold define un valor antes del cual ninguna de las transiciones debería haber llegado.

Como se mencionó en el planteamiento de problema, el compilador actual importa la información directamente de un archivo xls, este genera una salida correcta, pero no permite al usuario tener la información de entrada bajo control de versiones, y la optimización que este cuenta, es el de tener los mismos valores predeterminados para todos los corners por proyecto, y sobre escribir los necesarios.

Ese modelo descrito ha funcionado, pero los usuarios solicitan un modelado en el cual se aprovechen las similitudes que tienen las tecnologías que comparten ya fábrica, proceso o proyecto, lo cual abarca desde el diseño de la base de datos, la cual se realiza de manera paralela en la Etapa 1, Figura 1.2, y el proceso de visualización y compilación de los datos.

Para definir el compilador más allá de una caja negra que toma valores a partir de una base de datos y la sintetiza en un formato legible por distintas herramientas de modelado, se puede empezar por describir qué tipo de datos opera, siendo estos los corners y las distintas categorías de parámetros que los definen.

2.1.1. Análisis de tiempo basado en corners

Históricamente, se supuso que el análisis de tiempo estático en una curva lenta y una curva rápida era suficiente para garantizar que se cumplieran los requerimientos de timing. Se estimaba el retardo del circuito en el peor de los casos desde el corner lento para la biblioteca de celdas estándar: este era el corner del proceso lento, de operación lenta con una tensión de alimentación baja y alta temperatura.

Las violaciones de hold por rutas de tiempo rápidas y la potencia dinámica en el peor de los casos se estimaban a partir del corner de proceso rápido, con un corner de funcionamiento rápido con una mayor tensión de alimentación baja temperatura, por ejemplo 0°C. Tanto el corner del proceso lento como el rápido existen para transistores tipo p como para el tipo n debido a voltajes de umbral más altos o bajos y longitudes de canal más largas o cortas a causa de la variabilidad del proceso, que pueden reducir o aumentar la corriente de fuga del umbral. El peor caso de fuga de potencia se puede estimar en el proceso rápido y el corner de alta temperatura .

En la práctica, como se menciona en [2] el equipo de diseño puede estar más preocupado por ciertos escenarios, y estos pueden ser diferentes de un diseño a otro. Por ejemplo, los chips de ultra bajo consumo pueden tener diferentes compensaciones que los chips de alto rendimiento, y aquí es donde las cosas se complican. Ciertos tipos de diseños, como los chips de teléfonos móviles, priorizan el alto rendimiento limitado por la capacidad de la batería, a la hora de ejecutar aplicaciones o realizar funciones intensivas. Pero cuando está inactivo, se busca que esté operativo y funcional, pero con el menor consumo de energía. Por lo tanto, deben optimizarse en un conjunto más amplio de perfiles.

En el diseño de circuitos digitales, normalmente se supone que el retardo de la celda aumenta con la disminución del voltaje y aumento de la temperatura, esta suposición es la base del enfoque por corners con bibliotecas de celdas en análisis de tiempo estático (STA). Sin embargo, esta suposición se rompe a voltajes de suministro bajos porque el retardo de la celda puede disminuir con el aumento de la temperatura. Este fenómeno es causado por una competencia entre la movilidad y el voltaje de umbral para dominar el retardo de la celda. Nos referimos a este fenómeno como la dependencia de temperatura invertida (ITD) [4].

Las dependencias de temperatura normal e invertida se pueden ver en los gráficos 2.1 y 2.2, respectivamente, los cuales fueron obtenidos en [5]. Donde en cada gráfico se muestra cómo cambia el retardo de una celda NAND de una biblioteca de 90 nm en función del voltaje y la temperatura. La misma biblioteca de 90 nm se utiliza para ambos resultados. Para la dependencia del voltaje, estos gráficos concuerdan: el retardo aumenta a medida que disminuye el voltaje. Para la dependencia de la temperatura, estos gráficos no están de

acuerdo debido a ITD: El gráfico superior indica que el retardo aumenta con el aumento de la temperatura independientemente del voltaje, mientras que el gráfico inferior indica que el retardo aumenta con el aumento o la disminución de la temperatura dependiendo de si el voltaje es mayor o menor que el voltaje de cruce de casi 1,12 V.

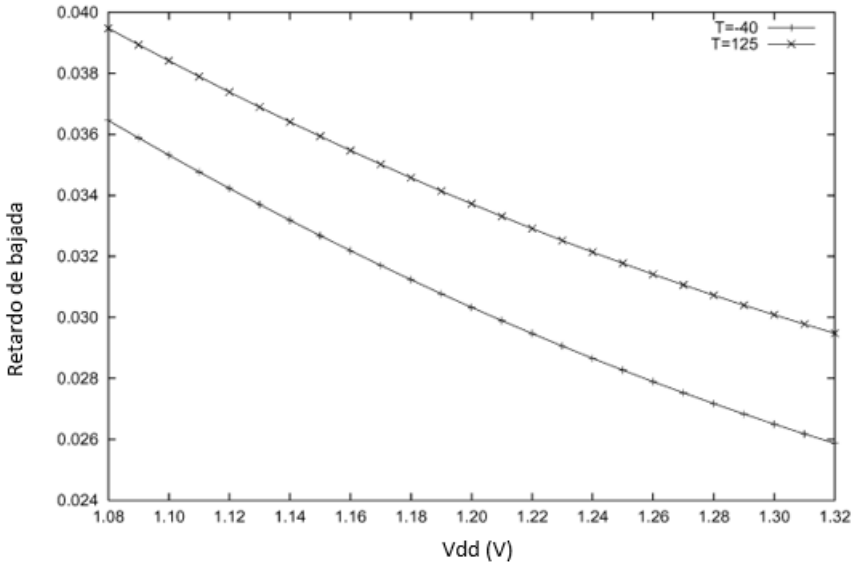


Figura 2.1: NAND Retardo de bajada (Vdd, Temperatura)[5]

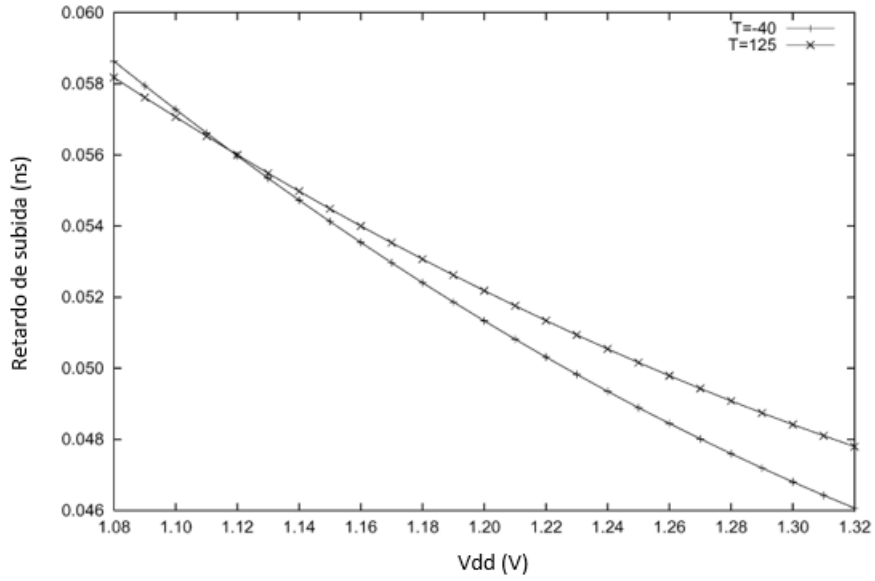


Figura 2.2: NAND Retardo de subida (Vdd, Temperatura) [5]

El retardo de caída en la imagen superior y subida en la imagen inferior de una celda NAND en función del voltaje y la temperatura para un S_{inp}=0.015 ns, Cout=0.003 pF. El gráfico superior muestra una dependencia normal de la temperatura, y el inferior una invertida.

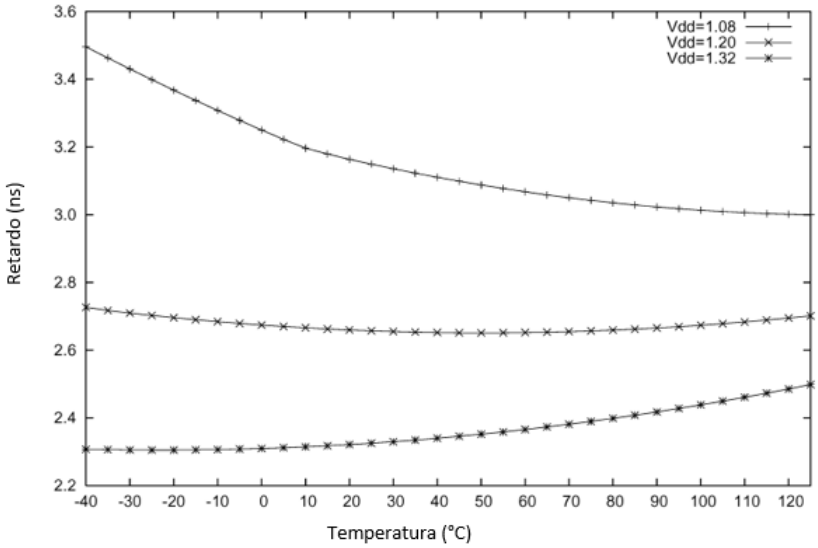


Figura 2.3: Retardo de una celda con 16 rutas [5]

El efecto de ITD en las rutas es más grave y se puede ver en la Figura 2.3. Si el retardo de una ruta sigue la dependencia normal de la temperatura, se espera que aumente monótonamente con el aumento de la temperatura. Por ejemplo, la curva de retardo para Vdd = 1,32 sigue este comportamiento. Sin embargo, debido a ITD, el retardo de una ruta puede aumentar o disminuir con el aumento de la temperatura. Por ejemplo, la curva de retardo para Vdd = 1,08 disminuye al aumentar la temperatura, mientras que la curva de retardo para Vdd = 1,20 primero disminuye y luego aumenta al aumentar la temperatura. La última curva de retardo muestra especialmente que no es trivial determinar la temperatura que minimiza un retardo de trayectoria. Esto, por supuesto, genera dificultades en el análisis del tiempo de espera debido a su dependencia de dichas rutas.

En la práctica, una combinación de variaciones de procesos lentos y rápidos y diferencias en las temperaturas puntuales en un chip puede dar lugar a fallas imprevistas por un solo corner del proceso. A medida que las tecnologías de proceso son cada vez más pequeñas, se deben modelar más variables para cubrir los casos de uso, como se menciona en [2] Una convención de nomenclatura para los corners de proceso es utilizar designadores de dos letras, donde la primera letra se refiere al corner del transistor de canal N (NMOS) y la segunda letra se refiere al corner del transistor de canal P (PMOS). En esta convención de nomenclatura, existen tres corners: típico, rápido y lento. Los corners rápido y lento presentan movilidades de portadores más altas y más bajas que las normales, respectivamente. Por ejemplo, un corner designado como FS denota NFETs rápidos y PFETs lentos. Donde algunos corners que comúnmente se modelan son FF (rápido rápido), SF (lento rápido), SS (lento lento), FS (rápido lento), TT (típico típico)
.

Actualmente tanto la optimización como sign-off de tiempo para una combinación de muchos corners y modos ha hecho común en la industria de circuitos integrados. Además,

la mayoría de los chips fabricados pueden ser sustancialmente más rápidos y tienen una potencia promedio más baja en condiciones normales del circuito, cabe resaltar que los costos de diseño aumentan por este tipo de análisis.

Aunque algunos elementos del circuito están bajo un control más estricto en los procesos actuales, la variación de otros elementos ha aumentado. Por ejemplo, una pequeña reducción en el voltaje de umbral del transistor o la longitud del canal puede provocar un gran aumento de las fugas. Una pequeña disminución en el rendimiento puede ser aceptable para cumplir con los requisitos de temporización.

Dada la inviabilidad de modelar el comportamiento de un circuito en todos los puntos del espacio-tiempo continuo, el análisis de tiempo se realiza sobre un conjunto discreto de puntos de ese espacio. Para un circuito activado por flanco, el primer nivel de discretización está en la dimensión del tiempo reduciendo el problema a un análisis de tiempo de un ciclo bajo el supuesto de que la sincronización del circuito se realiza con una señal de reloj restringida que tiene un período fijo con pequeñas fluctuaciones (jitter).

El segundo nivel de discretización está en el proceso (P) y las variaciones ambientales, es decir, el voltaje (V) y temperatura (T). Para cada parámetro, se define un conjunto discreto de valores representativos, y se selecciona un subconjunto de puntos en el eje correspondiente de la cuadrícula discreta. Un ejemplo típico se muestra en la 2.4 [5].

La dimensión de tensión en la Figura 2.4 tiene un punto central que representa la tensión nominal (p. ej., 1,0 V) y dos puntos extra que representan los valores máximo y mínimo alcanzables por potenciales fluctuaciones alrededor del voltaje nominal, típicamente ±10

Para la temperatura, los fabricantes definen diferentes grados según el dominio de aplicación en que se va a utilizar el circuito, por ejemplo, industrial, automotriz y militar. cada grado tiene un rango diferente de temperaturas de funcionamiento, por ejemplo, [- 40 °C, 125 °C] y [0 °C, 85 °C]. Los valores extremos de diferentes grados se seleccionan para la cuadrícula discreta, junto con algunos valores que representa una temperatura ambiente típica.

Las variaciones del proceso son cambios únicos que resultan de perturbaciones en el proceso de fabricación, que provocan cambios en los valores nominales de parámetros como la longitud efectiva del canal (Leff), el espesor del óxido (tox), la concentración de dopante (Na), el ancho del transistor (w), el espesor del dieléctrico de la capa intermedia (ILD) (tILD) y la altura y el ancho de la interconexión, entre otros [5].

Un corner es un punto en la cuadrícula de variabilidad discreta de Proceso/Voltaje/Temperatura (PVT), por ejemplo, [FF, - 40 °C, 1.1V]. Los fabricantes seleccionan un conjunto de corners en la cuadrícula PVT y proporcionan un caracterización de los dispositivos y las interconexiones en cada corner.

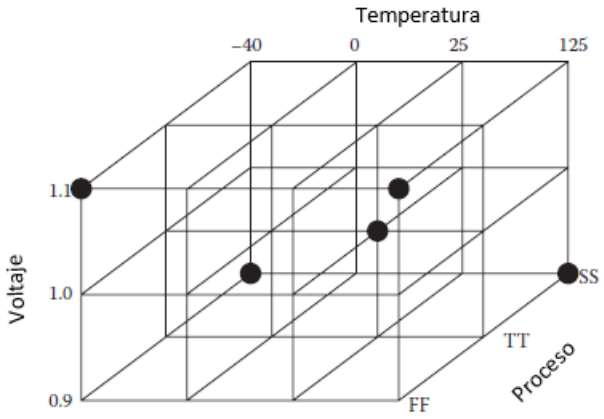


Figura 2.4: Representación gráfica de los corners [5]

En la Figura 2.4 se muestran las tres dimensiones de proceso, voltaje y temperatura en el cual se pueden modelar una amplia gana de corners, pero ¿Es el mismo cubo en todos los proyectos?, y no solo eso ¿Son estas las únicas dimensiones?.

De acuerdo a la especificacion recibida en la empresa, actualmente no basta con modelar las variables de PVT, sino también su interconexión, lo cual se puede visualizar como una estructura conformada por cubos de la Figura2.4 aumentando así las dimensiones de los corners. Además de las dimensiones que agrega esta estructura interconectada, por lo que es aquí donde el número de corners con los que se debe lidiar a nivel de diseño comienza a aumentar, porque la forma de la caja y las dimensiones de esa caja cambian en diferentes entornos de funcionamiento [2].

2.1.2. Caracterización de corners

Para conseguir la configuración adecuada de un corner, se debe definir múltiples variables, rondando los 400 parámetros en proyectos recientes de Intel, con la posibilidad de que haya dependencias de valores externos a la definición misma del corner (valores que no se encuentran en la base de datos).

Como se menciona en [2] los diseñadores a la hora de definir un corner, buscan un escenario que no sea excesivamente pesimista y que pueda afectar las metas de rendimiento establecidas para el circuito integrado. Se diseñará alrededor de algo intermedio o ligeramente agresivo, realizarán su análisis de tiempo, su análisis de potencia, harán todo. De esta manera, la mayor parte de las actividades de diseño se realizan alrededor del corner.

Se pueden agrupar los parámetros por categorías de acuerdo a su función, dando como resultado, parámetros de biblioteca, contexto, limitación, parasíticos, ambiente y temporización

Los parámetros de contexto brindan información que caracteriza al corner. siendo estos los valores para proceso, voltaje y temperatura, así como el tipo de extracción de parasíticos utilizada para el corner, lo que diferencia estos parámetros del resto es que son únicos para cada corner, la combinación de estas cuatro variables es única para cada escenario y sirve como etiqueta del mismo, creando así un "punto" nuevo en la Figura 2.4.

Los parámetros relacionados a bibliotecas en el contexto de circuitos integrados, son una colección de funciones de circuito, implementadas en una tecnología de circuito integrado particular, que un diseñador de circuitos integrados o aplicación para automatización de diseño electrónico (EDA) puede seleccionar para implementar un diseño. Las cuales están estrechamente ligadas a las bibliotecas de software, donde una colección de unidades de código de objeto que se pueden vincular, ya sea estáticamente o en tiempo de ejecución, con otras bibliotecas y/o código objeto para producir un programa de software [6].

El contenido de dichas bibliotecas, en el contexto del proyecto, son principalmente celdas, que se definen como una primitiva en una biblioteca de circuitos integrados. Para efectos de esta especificación, "primitiva" significa que las propiedades de temporización de la celda se describen directamente en el módulo de cálculo de retardo y potencia, sin referencia a la aplicación de la estructura interna de la celda. A su vez existen tipos de celdas, las cuales identifican por su nombre una celda en particular en la biblioteca. Las cuales suelen agruparse en *clusters*, que constan de instancias de celdas que están restringidos entre sí debido a ubicación u otra(s) característica(s) compartida(s). No se considera valido tener una sola instancia de celda en mas de un grupo a la vez.

Donde la interconexión entre celdas y clusters se realiza mediante nets, las cuales son una abstracción que expresa la idea de una conexión eléctrica entre varios puntos en un diseño. En una representación jerárquica del diseño, las redes pueden ocurrir en todos los niveles y pueden conectarse a pines de niveles jerárquicos más bajos (incluidas instancias de celda), puertos del nivel jerárquico actual y entre otros niveles.

Los parámetros de limitación. o más conocidos como derate, de acuerdo a [7] son utilizados para la disminución de las tensiones eléctricas, térmicas y mecánicas aplicadas a una pieza con el fin de disminuir la tasa de degradación y prolongar la vida útil esperada de la misma. La reducción de la capacidad de funcionamiento aumenta el margen de seguridad entre el nivel de tensión de operación y el nivel real de falla de la pieza, proporcionando una protección adicional contra anomalías del sistema no previstas por el diseñador.

Las celdas dependen a su vez de parámetros dados por la tecnología, temporización, extracción de parasíticos, entre otros. Donde los datos de tecnología son datos utilizados para calcular las propiedades de tiempo de una instancia de celda en función de su contexto en el diseño. Este término incluye información que no es específica del tipo de celda y datos específicos para cada tipo de celda en la biblioteca. El tipo de datos utilizados varía según la metodología de cálculo del temporización. Los datos generales y los datos de las celdas pueden estar contenidos en el mismo archivo o en archivos separados. Los datos de celda también pueden combinarse con los modelos de temporización de cada celda.

La importancia del modelado de pararámetros de extracción de parasíticos, como se menciona en [8], recae en que cada parasítico (inductancia, capacitancia y resistencia) en celdas e interconexiones desempeña un papel importante al obstaculizar el rendimiento eléctrico general de los dispositivos VLSI de alta frecuencia. La inductancia parásita de los terminales puede causar fluctuación en la tierra, lo cual a su vez puede provocar retraso en la propagación o errores lógicos. La inductancia mutua entre terminales genera acoplamiento entre líneas adyacentes, lo que resulta en cross-talk. La capacitancia de carga y la resistencia de los terminales reducirán significativamente la velocidad general del sistema. Además, la capacitancia terminal a terminal puede agregar acoplamiento no deseado. Es evidente que la calidad de las técnicas utilizadas para medir estos parasítico se vuelve muy importante al caracterizar varios paquetes y conexiones para dispositivos de alta velocidad.

Los parámetros de temporización establecen valores tanto para relojes como para celdas en la ruta de datos, con el fin de modelar setup y hold y así establecer un intervalo con respecto a la transición de una señal de referencia durante la cual otra señal no puede cambiar de valor. Las violaciones de setup y hold se explican en el libro [3] con la gráfica 2.5.

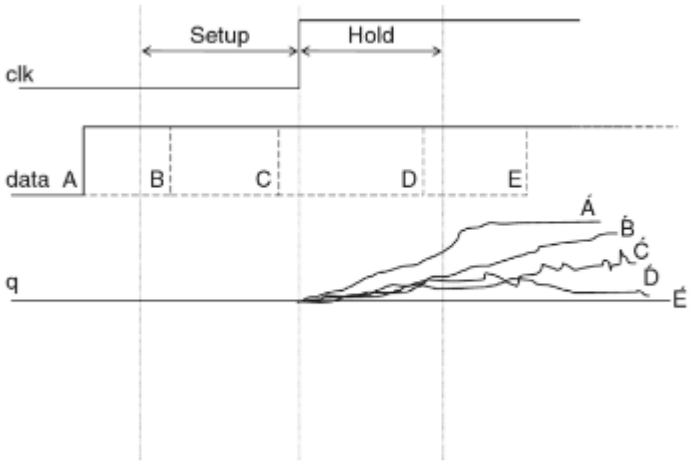


Figura 2.5: Respuesta a la salida para violaciones/cumplimiento de setup y hold [3]

En la gráfica 2.5 A, B, C, D y E muestran diferentes puntos en los cuales la entrada de datos podría cambiar. Á, B́, Ć, D́ y É muestran las formas de onda correspondientes en la salida del flip-flop. Transiciones en A o en E cumplen con los requisitos de setup y hold. Por lo tanto, la forma de onda de salida en el flip-flop es también muy limpia (es decir, confiable). B solo viola el requisito de setup, mientras que D solo viola el requisito de hold. Por ende, B y D finalmente alcanzan el valor previsto, pero les toma más tiempo llegar al valor previsto final.

Esta verificación de tiempo se aplica con frecuencia a flip-flops y latches para establecer un intervalo estable para la entrada de datos con respecto al flanco activo del reloj o la transición de activo a inactivo de la puerta.

Son necesarios dos valores límite para definir el intervalo estable. El tiempo de setup es el tiempo anterior a la transición de la señal de referencia cuando comienza el intervalo estable y será negativo si el intervalo estable comienza después de la transición de la señal de referencia. El tiempo de hold es el tiempo posterior a la transición de la señal de referencia cuando finaliza el intervalo estable y será negativo si el intervalo estable finaliza antes de la transición de la señal de referencia.

Tomando en cuanta dichos efectos, y otros que dependen de la tecnología, fabricación y herramienta utilizada definida en parte por los parámetros de ambiente, y extracción de parasíticos, se realiza un modelo de temporización, que representa el comportamiento de celdas y nets para aplicaciones como la simulación y análisis de tiempo.

Ya que existen decenas de parámetros por categoría, a la hora de diseñar los corners, se requiere definir valores que van desde escalamientos. nombre de bibliotecas, magnitudes, hasta expresiones regulares dependiendo del parámetro y su finalidad, ya que se modelan una extensa variedad de escenarios, desde un corner de proceso SS (nMOS lento y pMOS

lento) y parasíticos maxRC (resistencia y capacitancia máximas) a la vez se diseña un corner de proceso FF (nMOS rápido y pMOS rápido) con un parasíticos minRC (resistencia y capacitancia mínimas). Dado que el diseño puede ser mixto, se modelan los posibles dominios de voltaje, y sus corners correspondientes, tomando en cuenta variaciones entre bibliotecas, contextos, derates y demás categorías.

2.2. Definiciones para circuitos integrados

El estándar IEEE 1481 [9], define una serie de conceptos para el diseño de circuitos integrados, los cuales sientan las bases y son necesarios para dar contexto al proyecto en cuanto a su importancia en el flujo ASIC, y conceptos asociados para el diseño de corners.

2.2.1. Diseño de circuitos integrados VLSI

Esta sección describe qué es el diseño VLSI, específicamente el diseño basado en celdas. Se inicia con una breve definición y reseña histórica de los circuitos integrados y cómo han evolucionado a lo largo del tiempo. Posteriormente, se detallan los distintos pasos del flujo ASIC que se siguen en el desarrollo de circuitos integrados, desde la definición de la funcionalidad del circuito hasta su fabricación. Se enfatiza la complejidad del proceso de diseño de circuitos integrados modernos, y cómo esa complejidad está directamente relacionada con la cantidad de parámetros que se toman en cuenta para el modelado de escenarios PVT.

Un circuito integrado (CI) es un pequeño dispositivo electrónico basado en semiconductores que consta de transistores, resistencias y condensadores. Los circuitos integrados son los componentes básicos de la mayoría de los dispositivos y equipos electrónicos. Un circuito integrado también se conoce como chip o microchip [10].

En 1965, Gordon Moore observó que al graficar el número de transistores que se pueden fabricar en un chip de la manera más económica da una línea recta en una escala semilogarítmica. En ese momento, descubrió que el número de transistores se duplicaba cada 18 meses. Esta observación se ha llamado Ley de Moore y se ha convertido en una profecía autocumplida. La Ley de Moore se basa principalmente en la reducción del tamaño de los transistores y, en menor medida, en la construcción de chips más grandes. El nivel de integración de chips se ha clasificado en pequeña, mediana, gran y muy gran escala.

Los circuitos de integración a pequeña escala (SSI), como el inversor 7404, tienen menos de 10 puertas, con aproximadamente media docena de transistores por puerta. Circuitos de integración de mediana escala (MSI), como el contador 74161, tienen hasta 1000 puertas. Los circuitos de Integración a gran escala (LSI), como los microprocesadores simples de 8 bits, tienen hasta 10.000 puertas. Pronto se convirtió evidente que tendrían que crearse nuevos nombres cada cinco años si esta tendencia de nombres continuaba y, por lo tanto, el término integración a muy gran escala (VLSI) se usa para describir la mayoría circuitos integrados desde la década de 1980 en adelante. Para dimensionar la disminución del tamaño de los circuitos integrados se pueden comparar el primer microprocesador, el 4004 de Intel, lanzado en noviembre de 1971 al mercado, el cual fue diseñado para una

tecnología de 10um, con los producidos actualmente en 7nm por TSMC, esta diferencia en tamaño se traduce que en el mismo espacio físico se pueden colocar 1000 veces más transistores que hace 50 años. [5]

En respuesta a una lista de requerimientos de diseño cada día más exigente, se ha pasado de un flujo de diseño que constaba de una simple lista de tareas, a un proceso complejo e iterativo, llamado flujo ASIC que sintetizando la información en [5] [11] se muestra de manera simplificada a continuación:

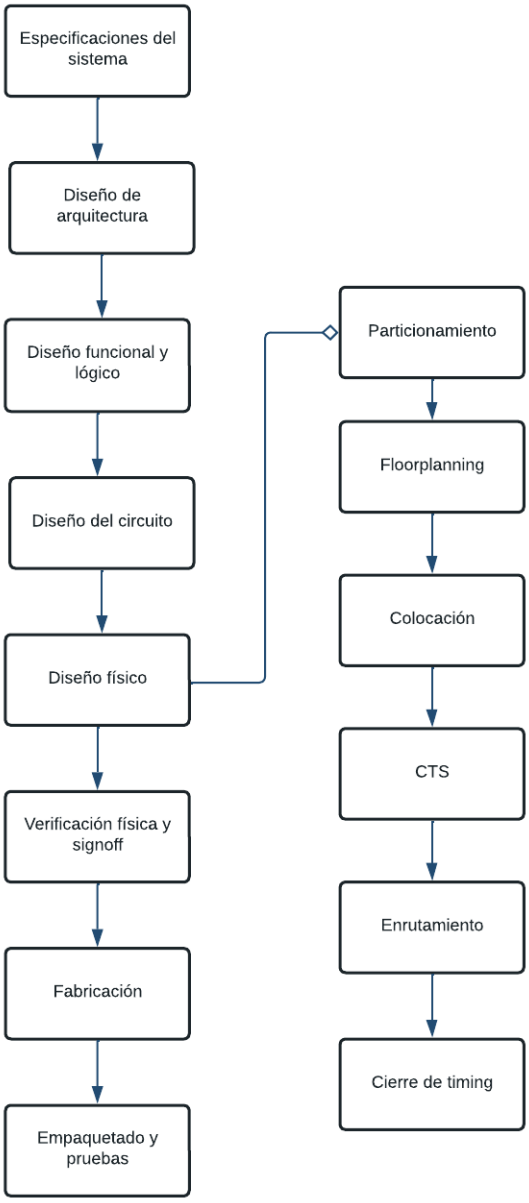


Figura 2.6: Flujo ASIC

En el flujo ASIC mostrado en 2.6 se parte de una fase conceptual en la cual se desarrollan los objetivos funcionales y la arquitectura del chip.

En el diseño conceptual también se deben definir en qué escenarios se va a usar el chip y su propósito; una vez realizadas estas tareas se inicia con la fase de diseño lógico del RTL, en esta etapa la arquitectura se implementa en un lenguaje de nivel de transferencia de registro, *RTL* por sus siglas en inglés, y luego se realizan simulaciones para verificar que realiza correctamente las funciones deseadas.
La etapa de diseño físico inicia por el particionamiento que como se describe en [12] consiste en descomponer un circuito en subcircuitos o módulos más pequeños, que pueden ser diseñados o analizados de manera individual, la cual se basa en una estrategia de dividir y conquistar para el diseño de chips que puede implementarse al distribuir cada bloque individualmente y luego volver a ensamblar los resultados como particiones geométricas.

Históricamente, esta estrategia se utilizaba para la partición manual, pero se volvió inviable para netlists grandes. En su lugar, la partición manual puede realizarse en el contexto de módulos a nivel del sistema al considerarlos como entidades únicas, en casos en los que la información jerárquica esté disponible. En contraste, la partición automática de netlists puede manejar netlists grandes y redefinir una jerarquía física de un sistema electrónico, que abarque desde particiones hasta secciones, super secciones o fullchip.

Una vez aprobadas las pruebas del RTL, se procede con el floorplanning, en donde los módulos RTL de nivel superior del chip se asignan a grandes regiones del chip, se asignan pines de entrada/salida (E/S). Como se menciona en [12] en esta etapa se determina las formas y disposición de los subcircuitos o módulos, así como las ubicaciones de los puertos externos y los bloques IP o macro, lo que optimiza tanto las ubicaciones como las relaciones de aspecto de los bloques individuales, las interconexiones y el delay que estas implican.

Posterior al floorplaning se realiza la síntesis, al RTL, donde las puertas en la lista de conexiones se asignan a ubicaciones que no se superponen en el chip. La colocación es el proceso de colocar el diseño sintetizado (conexión estructural de celdas estándar) en el floorplan especificado. Si bien se colocan celdas menores (como buffers y celdas de conexión) que tienen lugar por separado y entre varias etapas de diseño, la colocación o placement generalmente se refiere a la ubicación inicial de las celdas estándar.

Antes de interconectar las celdas, en la etapa de enrutamiento, se colocan los relojes del circuito, para lo cual se realiza primeramente un árbol de síntesis de reloj, *CTS* por sus siglas en inglés, en el cual primero agrupan elementos secuenciales, sobre todo en función de dónde se encuentran en el diseño en relación con el pin de reloj de nivel superior y la lógica de activación de reloj común. El número de elementos en cada grupo es selecciona-

do para que no presente una carga demasiado grande a una celda de conducción. Estos grupos de elementos secuenciales son las "hojas" del árbol del reloj unidas a las ramas. Por último, el algoritmo CTS intenta garantizar que el retardo desde el pin de reloj de nivel superior hasta las hojas sean todos iguales, esto lo logra agregando y dimensionando buffers de reloj entre el pin de nivel superior y las hojas.

Para definir las celdas necesarias en el flujo ASIC descrito, el cual se centra en la etapa de pre-silicio, es necesario verificar los escenarios de operación del chip, también conocido como corners en la etapa de signoff, los cuales van a determinar qué celdas se pueden utilizar para cumplir con los requerimientos del proyecto, ya que al realizar etapas como placement, síntesis o CTS se debe trabajar con las celdas adecuadas para evitar violaciones de setup, hold, potencia, interconexión, entre otras.

La etapa de fabricación, no se puede ver como un proceso ajeno al diseño, ya que el proveedor y el proceso de fabricación del circuito, también afectan la manera en la que se modela en pre-silicio, ya que delimita las bibliotecas de celdas que se pueden utilizar, la metodología para modelar escenarios y la distancia entre metales, determinando así qué proyectos son o no realizables de acuerdo a las limitaciones que presente la fábrica.

2.2.2. Generación de archivos en formato TCL

Como parte de las necesidades del usuario es que el software genere un scipt en formato TCL, esto como requisito de compatibilidad con los programas de la empresa, en el cual se encuentren todos los corners y parámetros definidos para determinado proyecto, e incluya condicionales especiales y derates específicos de acuerdo con tecnología que se esté diseñando.

A medida que los proyectos se vuelven más complejos, definir los corners es un proceso que toma días y múltiples revisiones. Por lo tanto cometer un error en la definición de un corner, significa una pérdida de tiempo y dinero, esto debido a que cada vez que se realiza un cambio, este debe ser aprobado por una serie de regresiones que toman horas, sin mencionar el esfuerzo adicional para depurar algo que se consideraba correcto.

Como se menciona en [13] los scripts de TCL tienen gran utilidad para compilar con la misma configuración y orden cada vez. También se puede acceder a funciones poderosas de las herramientas de EDA que solo están disponibles a través de TCL (como ejecutar el lugar y enrutado varias veces con diferentes opciones para mejorar la temporización). Cuando los scripts de Tcl están construidos correctamente, generalmente se requiere poco mantenimiento.

Ahora bien, tomando esto en consideración, la literatura consultada en [14] define Tcl como un lenguaje dinámico, también conocido como lenguaje de secuencias de comandos, utilizado para controlar y extender aplicaciones. Tcl proporciona facilidades generales de programación suficientes para la mayoría de las aplicaciones. Además, Tcl es integrable y extensible. Su intérprete es un biblioteca de funciones escritas en lenguaje C que se pueden incorporar fácilmente a las aplicaciones, y cada aplicación puede ampliar las funciones principales de Tcl con comandos adicionales ya sea exclusivo de la aplicación o proporcionado por bibliotecas complementarias (como extensiones en la comunidad TCL). Dichas extensiones cuando facilitan la creación de scripts entre lenguages de programación, son llamados preprocesadores [15], dentro de los que destacan Assemble tool, tpp, Tipi y g2pp.

Assemble tool agrupa múltiples archivos de código fuente Tcl en un solo archivo. También funciona como un preprocesador que entiende #define y #ifdef / #ifndef. Assemble se desarrolla como parte del proyecto Sqawk, el cual es un programa que usa SQL y puede combinar datos de varios archivos y está alimentado por SQLite.

Tpp es utilizado para ejecutar secuencias de comandos Tcl para eliminar espacios en blanco y comentarios extraños antes de la ejecución, así como proporcionar la capacidad de usar macros y declaraciones #define constantes.

Tipi es un procesador de macros diseñado para usarse en código Tcl, que tiene características similares al preprocesador C, g2pp es una herramienta que convierte Tcl en un lenguaje de preprocesamiento.

Dichos preprocesadores aun se encuentran lejos de los objetivos que se buscan en la empresa, dado que son herramientas de uso general, y no presentan una solución especifica a las necesidades de un usuario. Un compilador de scripts en TCL puede ser llevado un paso más allá, al ser de uso específico, como lo hizo Jia Huang y Hamid Shahnasser en su paper A Preprocessor Tcl Script Generator for NS-2 Communication Network Simulation [16].

En dicho paper, se parte de que la simulación de redes es de los métodos de evaluación predominantes en la comunicación de redes, además de que es ampliamente utilizado para el desarrollo de nuevas arquitecturas de comunicación y protocolos de red. La herramienta de simulación NS-2 ha sido ampliamente utilizada para estos propósitos, y tiene como entrada un archivo en formato Tcl.

El preprocesador de Huang y Shahnasser integra una interfaz gráfica con múltiples pestañas, en la cual los usuarios pueden escoger los parámetros para redes cableadas o inalámbricas, para luego compilar el tcl.

El preprocesador de [16] cuenta con más características de selección y visualización de datos que los de uso general como tpp o assemble tool, pero no posee control de versiones para trabajar con múltiples proyectos de varios usuarios, así como tampoco puede importar o editar bases de datos.

2.2.3. Control de versiones

Muchos desarrolladores de software han utilizado los sistemas de control de versiones (VCS) durante el desarrollo de proyectos, ya que les ayuda a administrar los códigos fuente y les permite conservar todas las versiones del proyecto en el que han trabajado. Es el camino hacia la gestión, organización y coordinación del desarrollo de los objetos. En ingeniería, los desarrolladores de software deben colaborar entre sí para desarrollar un mejor producto. Por lo tanto, VCS es muy útil porque también es compatible con un marco colaborativo que facilita las cosas a los desarrolladores de software[17].

Los sistemas de control de versiones (VCS) permiten la aceleración y la simplificación del proceso de desarrollo de software, y permitir nuevos flujos de trabajo. Realizan un seguimiento de los archivos y su historial y tienen un modelo para el acceso simultáneo.

En [18] dos enfoques diferentes para los VCS:

1. Sistemas de control de versiones centralizados (CVCS).
2. Sistemas de control de versiones distribuidos (DVCS).

Los sistemas de control de versiones centralizados se denominan de esta manera porque solo hay un repositorio central. El repositorio es básicamente un servidor en el que se almacenan todos los archivos rastreados, incluido su historial. Se puede acceder al repositorio a través de LAN o WAN desde cualquier parte del mundo. Cada etapa del historial de un archivo se identifica mediante una revisión o versión (normalmente, un número entero). Una revisión consiste en el archivo y algunos metadatos. Los metadatos que se almacenan pueden diferir según el programa. La revisión más reciente a menudo se llama cabeza o head.

El usuario puede retirar archivos individuales, así como el todo el repositorio y puede especificar la revisión que quiere recuperar del repositorio (por ejemplo, head o cualquier otra revisión). Al hacer un checkout el usuario recupera una copia de trabajo en su computadora local, una copia de trabajo no incluye historial, pero permite al usuario editar los archivos.

Después de editar los archivos (por ejemplo, implementar una característica determinada) el usuario envía los archivos modificados al repositorio. También agrega un registro de confirmación o un mensaje de confirmación, que es una descripción de lo que ha cambiado desde la última versión. Esta es una parte de los metadatos que se guardan en el historial. Al confirmar la modificación del archivo, se debe crear una copia en el repositorio, lo que crea una nueva revisión. Los cambios entre dos revisiones de un archivo se denominan diff o delta.

Una vez realizada la confirmación del usuario A, el usuario B no tiene la revisión actual de los archivos nunca más. Por lo tanto, el usuario B necesita ejecutar una actualización para recuperar la revisión actual. Si el usuario B trabajó en el archivo que cambió el usuario A, el VCS intenta fusionar los archivos automáticamente. Pueden surgir conflictos si los archivos contienen cambios incompatibles y los usuarios necesitan resolver estos conflictos manualmente. El desarrollo principal tiene lugar en la línea principal o el tronco. Si hay una necesidad de una línea de desarrollo independiente, el usuario puede iniciar una rama. Las ramas son a menudo se utilizadas para implementar una característica. Durante el proceso de implementación de la misma, la rama puede volverse inestable, pero no afecta a otros usuarios. Una vez que la implementación se completa y se prueba, la característica de la rama se fusiona de nuevo en el tronco. La mayoría de las operaciones anteriores requieren acceso a la red. Por lo tanto, la red es siempre el cuello de botella en el control de versiones.

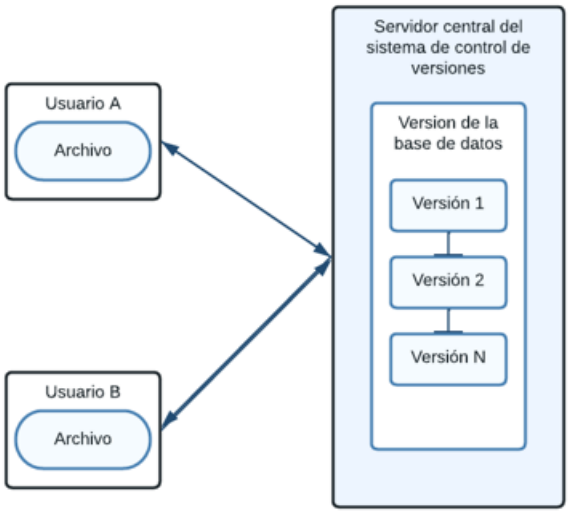


Figura 2.7: Control de versión centralizado

En cuanto al control de versiones distribuido, se diferencia principalmente de los centralizados en que no requieren un servidor central que contenga el repositorio, además de que cada usuario tiene acceso completo al repositorio desde su computadora, a esto se llama repositorio local. En un sistema distribuido no hay necesidad de un servidor que contenga toda la información, pero a su vez para el usuario le implica más espacio de memoria ya que almacena todo el repositorio en lugar de solo una copia.

Un repositorio local permite al usuario trabajar completamente desconectado. La mayoría de las operaciones son mucho más rápidas porque son locales. Por ejemplo, para examinar las diferencias entre dos archivos es una operación simple. Dependiendo del tamaño del archivo sólo toma una fracción de segundo. Sin embargo, si uno de los archivos está almacenado en un servidor, el archivo debe ser descargado primero.

Sin embargo, en algún momento el usuario necesita acceso a la red para compartir su repositorio con otros. El usuario A puede hacer su repositorio local disponible a través de la red o puede crear un repositorio remoto en un servidor, que es más común. El usuario B ahora puede clonar el repositorio remoto de usuario A. A partir de ese momento, el usuario B tiene un repositorio local y puede trabajar sin conexión, como se muestra en la Figura 2.8 .

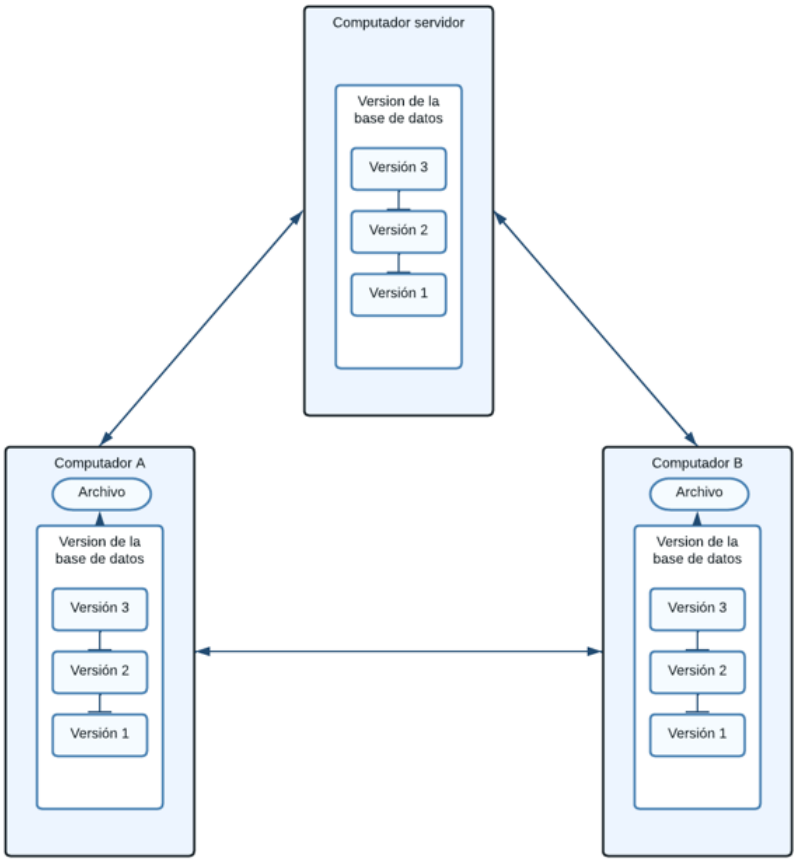


Figura 2.8: Control de versión distribuido

El control de versiones distribuido presentado en la Figura 2.8, se adapta a un sistema de repositorios Git, que como se explica en [19], es una base de datos que contiene toda la información necesaria para retener y gestionar las revisiones y el historial de un proyecto. En Git, al igual que en la mayoría de los sistemas de control de versiones, un repositorio conserva una copia completa de todo el proyecto a lo largo de su vida útil. Sin embargo, a diferencia de la mayoría de los otros VCS, el repositorio de Git no solo proporciona una copia de trabajo completa de todos los archivos en el repositorio, sino también una copia del repositorio en sí con el que trabajar.

Git mantiene un conjunto de valores de configuración dentro de cada repositorio. A diferencia de los datos de archivo y otros metadatos del repositorio, la configuración no se propaga de un repositorio a otro durante una operación de clonación o duplicación. En cambio, Git gestiona e inspecciona la configuración y la información de configuración de forma individual para cada sitio, usuario y repositorio.

Dentro de un repositorio, Git mantiene dos estructuras de datos principales: el almacén de objetos y el índice. Todos estos datos del repositorio se almacenan en la raíz de tu directorio de trabajo en un subdirectorio oculto llamado .git. El almacén de objetos está diseñado para ser copiado de manera eficiente durante una operación de clonación como parte del mecanismo que respalda un VCS completamente distribuido. El índice es información transitoria, privada de un repositorio y se puede crear o modificar según sea necesario.

En el centro de la implementación del repositorio de Git se encuentra el almacén de objetos, este contiene archivos de datos originales y todos los mensajes de registro, información del autor, fechas y otra información necesaria para reconstruir cualquier versión o rama del proyecto.

Git coloca únicamente cuatro tipos de objetos en el almacén de objetos: blobs, árboles, commits y etiquetas. Estos cuatro objetos atómicos forman la base de las estructuras de datos de nivel superior de Git, en [19] se detallan estas estucturas, las cuales se sintetizan a continuación.

Cada versión de un archivo se representa como un blob. Blob, una contracción de "binary large object"(objeto binario grande), es un término comúnmente utilizado en informática para referirse a alguna variable o archivo que puede contener cualquier tipo de datos y cuya estructura interna es ignorada por el programa, estos contienen los datos de un archivo pero no su metadata, ni siquiera su nombre.

Un objeto árbol representa un nivel de información de directorio. Registra los identificado-

res de blobs, los nombres de las rutas y parte de metadatos para todos los archivos en un directorio. También puede hacer referencia recursiva a otros objetos árbol (sub-árboles) y así construir una jerarquía completa de archivos y subdirectorios.

El objeto commit contiene metadatos para cada cambio introducido en el repositorio, incluyendo el autor, quien realizó los cambios, mejor llamado committer, la fecha del commit y el mensaje de registro. Cada commit apunta a un objeto árbol que captura, en una instantánea completa, el estado del repositorio en el momento en que se realizó el commit.

La etiqueta asigna un nombre arbitrario, aunque presuntamente legible por humanos, a un objeto específico, generalmente un commit. Aunque la referencia 8ca711h812 se refiere a un commit exacto y bien definido, un nombre de etiqueta más familiar como TFG-1.0-Alpha podría tener más sentido. Con el tiempo, toda la información en el almacén de objetos cambia y crece, rastreando y modelando las ediciones, adiciones y eliminaciones de tu proyecto. Para utilizar eficientemente el espacio en disco y el ancho de banda de la red, Git comprime y almacena los objetos en archivos de paquetes (pack files), que también se colocan en el almacén de objetos.

Es importante ver a Git como algo más que un Sistema de Control de Versiones (VCS, por sus siglas en inglés): Git es un sistema de seguimiento de contenido. Esta distinción, aunque sutil, guía gran parte del diseño de Git y es quizás la razón clave por la que puede realizar manipulaciones internas de datos con relativa facilidad.

El seguimiento de contenido en Git se manifiesta de dos maneras críticas que difieren fundamentalmente de la mayoría de los demás sistemas de control de revisiones.

En primer lugar, el almacén de objetos de Git se basa en el cálculo hash del contenido de sus objetos, no en los nombres de archivos o directorios de la estructura de archivos original del usuario. Por lo tanto, cuando Git coloca un archivo en el almacén de objetos, lo hace en función del hash de los datos y no del nombre del archivo. De hecho, Git no realiza un seguimiento de los nombres de archivos o directorios, que están asociados a los archivos de formas secundarias. Una vez más, Git realiza un seguimiento del contenido en lugar de los archivos.

Si dos archivos separados tienen exactamente el mismo contenido, ya sea en el mismo directorio o en diferentes directorios, Git almacena una sola copia de ese contenido como un blob dentro del almacén de objetos. Git calcula el código hash de cada archivo únicamente en función de su contenido, determina que los archivos tienen los mismos valores SHA1 y, por lo tanto, el mismo contenido, y coloca el objeto blob en el almacén de objetos indexado por ese valor SHA1, el cual es un algoritmo de hash seguro que sirve para encriptar la data. Ambos archivos en el proyecto, independientemente de su ubicación en

la estructura de directorios del usuario, utilizan ese mismo objeto para el contenido.

Si uno de esos archivos cambia, Git calcula un nuevo SHA1 para él, determina que ahora es un objeto blob diferente y agrega el nuevo blob al almacén de objetos. El blob original permanece en el almacén de objetos para que el archivo sin cambios lo utilice.

Segundo, la base de datos interna de Git almacena eficientemente cada versión de cada archivo, no sus diferencias, a medida que los archivos pasan de una revisión a la siguiente. Debido a que Git utiliza el hash del contenido completo de un archivo como nombre para ese archivo, debe operar en cada copia completa del archivo. No puede basar su trabajo ni las entradas de su almacén de objetos en solo una parte del contenido del archivo ni en las diferencias entre dos revisiones de ese archivo.

Al final Git calcula la información solicitada como un conjunto de cambios entre diferentes blobs con hashes variables, en lugar de almacenar directamente un nombre de archivo y un conjunto de diferencias.

2.2.4. Interfaz gráfica de usuario

Dentro de los requerimientos del cliente se define que se debe mejorar la experiencia de usuario, para ello se propone una interfaz gráfica de usuario la cual permita una interacción con el sistema de una manera lógica, intuitiva y eficiente. Andrew Dillon en [20] menciona que el diseño de interfaces de computadora que sean utilizables y fáciles de aprender por los humanos es un problema no trivial para los desarrolladores de software. Como las tecnologías de la información median muchas de las actividades que ahora realizamos de manera rutinaria, el proceso de interacción humano-computadora es de fundamental importancia.

Cuando diseña la interfaz de usuario de un sistema informático, se decide qué pantallas el sistema mostrará, qué aparecerá exactamente en cada pantalla y cómo se verá. También se debe decidir en qué puede hacer clic el usuario y qué sucede cuando lo hace, además de todos los demás detalles de la interfaz de usuario. Es responsabilidad del programador que el sistema cumpla con los requisitos de uso y sea intuitivo [21].

Christian Gram en su paper [22] define una ruta para el diseño de interfaces gráficas, que sumándole los conceptos que establecen en [20] para una buena interacción humano-computador, se puede converger a las siguientes consideraciones para la interfaz gráfica de usuario del presente proyecto, las cuales se definen a continuación.

Visibilidad del estado del sistema

El primer concepto ligado a la visibilidad del estado del sistema, es la observabilidad. Un sistema es observable siempre y cuando toda información potencialmente relevante para el usuario esté disponible. Esto no quiere decir que un sistema siempre poseerá toda la información pero si le permitirá una facilidad al usuario de inspeccionar toda información relevante en cualquier momento. En segundo lugar se debe tomar en cuenta la insistencia, la cual aplica en una interfaz gráfica si la estructura del diálogo asegura que el usuario perciba toda información necesaria.

La interfaz gráfica de usuario puede ser vista como una unión entre el sistema y el mundo real, en la cual se debe asegurar tanto la honestidad como la previsibilidad, los cuales aplican de la siguiente manera: un sistema es honesto si la estructura del diálogo asegura una unión entre la interpretación del usuario y la interpretación del diálogo que destinó el diseñador para esa información. Por otro lado un sistema es previsible si el usuario puede predecir reacciones futuras del sistema desde su estado actual o desde una interacción previa.

Control del usuario y libertad

El control del usuario y libertad implica tres conceptos clave en el proyecto, los cuales son la accesibilidad, personalización y prevención. En cuanto a la accesibilididad se refiere a si el usuario puede acceder a cualquier estado, sin importar el estado actual en el que se encuentra. La personalización consiste en que cambie el contenido y la forma del diálogo "sobre la marcha". Y por último pero no menos importante se debe contemplar si el sistema el preventivo, esto implica en qué grado se le da al usuario la libertad de elegir el siguiente paso en su interacción.

Consistencia y estándares

Los estándares a nivel de interfaz gráfica van ligados a los utilizados por el cliente para documentación, seguridad y calidad. Por otro lado se debe buscar que su implementación sea integral, lo que implica que la nueva interfaz no debe ser significativamente diferente en uso a previos sistemas ya existentes.

En cuanto a la prevención del fallo se debe contemplar la reutilización de entradas y salidas, y tolerancia tanto al ritmo como a la desviación. En cuanto a la reutilización de entrada y salida, un sistema que posee una reutilización de la entrada y la salida, permite que el uso de entradas y salidas anteriores, pueda usarse nuevamente en un futuro.

En cuanto a la tolerancia al ritmo, se refiere a que el sistema posee una reutilización de la entrada y la salida, además permite que el uso de entradas y salidas anteriores, y pueda usarse nuevamente en un futuro. La tolerancia al ritmo implica que el sistema logra recuperar información o crear copias de seguridad.

Flexibilidad y eficiencia de su uso

En cuanto a flexibilidad y eficiencia de uso refiere, se toma en cuenta la multiplicidad de dispositivos, donde múltiples entradas y salidas de dispositivos pueden usarse para el diálogo. La eficiencia de tiempo de ejecución en el proyecto implica desde el acceso al proyecto hasta la selección de parámetros a través de distintas ventanas, todo este asegurando la mantenibilidad del sistema, en la cual se facilite la detección de errores. Una vez establecido un marco teórico, el cual contiene tanto conceptos como investigaciones relacionadas, se plantea la hipótesis y objetivos del proyecto en el siguiente capítulo.

Capítulo 3

Hipótesis y Objetivos de la Investigación

3.1. Hipótesis

Una buena escogencia de parámetros PVT es esencial para el diseño de los circuitos integrados, por lo que una herramienta robusta, intuitiva y confiable es de gran ayuda para su establecimiento y verificación. La hipótesis planteada pretende suplir dichas necesidades con una nueva herramienta optimizada:

"Un compilador de parámetros PVT que integra control de versiones e interfaz gráfica para generar archivos de configuración en formato TCL, es capaz de optimizar en al menos un 5 % la cantidad de instrucciones que se requieren para completar la definición de parámetros en comparación con la cantidad de instrucciones actualmente requerida."

3.2. Objetivo General

Implementar un compilador que integre control de versiones e interfaz gráfica de usuario, que optimice la cantidad de instrucciones necesarias para definir parámetros PVT respecto a la solución actual.

3.3. Objetivos Específicos

1. Describir las necesidades de los usuarios y la base de datos requerida para el compilador.
2. Compilar un archivo en formato TCL que sea una versión equivalente y optimizada de la generada actualmente.
3. Adaptar un control de versiones que permita la creación y acceso de múltiples archivos de distintos usuarios.
4. Implementar una interfaz gráfica de usuario para visualizar y editar parámetros, corners y proyectos.
5. Establecer y llevar a cabo un plan de pruebas para el compilador.

3.4. Alcances y Limitaciones

Dado que el proyecto se desarrolla en un contexto cambiante, se establecen las condiciones en las cuales la investigación se lleva a cabo, con el fin de delimitar los resultados que se obtengan de la misma. Las consideraciones que no se contemplen dentro de los objetivos o entregables o que se encuentren fuera de los alcances y limitaciones indicados a continuación, quedan por fuera del proyecto.

Los datos mostrados están sujetos a escalamiento o cambio de nombre debido a temas de confidencialidad. En cuanto a las pruebas de la interfaz gráfica, estas son hechos de manera manual y verificadas por el cliente. Por estándares de seguridad de Intel, no se pueden generar archivos ejecutables.

Las funciones que se pueden realizar desde la interfaz gráfica son:

Mostrar Tablas de proyectos, parámetros y corners.
Agregar Corners y parámetros a un proyecto.
Eliminar Tanto corners o parámetros de un proyecto.
Editar Corners o parámetros de un proyecto.
Buscar Corners y parámetros a través de barras de búsqueda.
Compilar Un archivo en formato TCL que sea equivalente a la solución actual.
Crear Nuevos proyectos, corners, parámetros y áreas de trabajo.

Capítulo 4

Descripción Detallada de la Solución

Este capítulo aborda tanto la descripción detallada de la solución como la metodología implementada. Siendo el objetivo principal de la solución el de diseñar un compilador que integre control de versiones e interfaz gráfica de usuario, que optimice la cantidad de instrucciones necesarias para definir parámetros PVT respecto a la solución actual.

En relación con la metodología, se detalla el plan de trabajo para llevar a cabo un proyecto el cual requiere constante retroalimentación de los usuarios, planeamiento continuo y pruebas constantes, para que el mismo sea aprobado tanto a nivel académico y cumpla su propósito como herramienta de ingeniería en la industria.

Tanto el control de versiones como el compilador trabajan con base en un sistema de capas el cual fue adaptado para la solución, así como las distintas funciones de rastreo de información, el algoritmo utilizado por el compilador para optimizar líneas de código y la utilidad de la visualización y edición a partir de la interfaz gráfica.

4.0.1. Metodología

La metodología de trabajo llevada a cabo, dado al constante desarrollo, retroalimentación y planeamiento que requiere el proyecto, fue la metodología Agile, que de acuerdo con [23] este modelo iterativo e incremental permite realizar entregas rápidas a los clientes y mejorar su satisfacción. Agile se centra principalmente en dar la máxima prioridad a los clientes y sus necesidades, realizar entregas frecuentes en ciclos incrementales cortos, incorporar a los clientes como parte del equipo y colaborar con ellos, reaccionar rápidamente a los cambios, tener planes flexibles de corta duración, simplicidad en la solución de un problema, reuniones presenciales y contar con equipos “auto organizados y multifunciones”. Además, actividades como la planificación, modelado, la construcción y diseño de pruebas, en lugar de ser secuenciales, son actividades continuas. Otra metodología que es-

tuvo entre las opciones fue la cascada o desarrollo incremental, pero como explican Gonen y Sawart en [24] en el modelo de proceso en cascada o incremental, es casi obligatorio que el cliente proporcione sus requisitos en la fase inicial antes de que comience el desarrollo, lo que significa que no es receptivo a los cambios en los requisitos, siendo esto último el motivo de su descarte.

El ciclo de trabajo de Agile en[25] se describe en cinco etapas, pre-planificación, planificación, entrega de planificación, planificación iterativa y gestión del backlog.

En la pre-planificación se formula el backlog del producto, que consiste en todos los requisitos y una lista de todo el trabajo deseado en el proyecto. Para cada ítem del backlog se asigna una prioridad, nivel de esfuerzo que requiere llevarlo a cabo y un propietario.

Luego continúa la etapa de planificación, en la que se decide el tiempo entre iteraciones, en nuestro caso cada dos semanas, la velocidad de desarrollo, que para este proyecto promedió una tarea de alta exigencia y otra de mediana por iteración, además de las historias o tareas designadas por desarrollador.

La entrega de la planificación contiene todas las características y las diversas tareas que se implementarán en el proyecto, se fragmenta para formar iteraciones por parte del equipo del proyecto y los clientes. Estas iteraciones se asignan a cada liberación, y cada liberación proporciona al cliente el modelo de trabajo del proyecto con las características enumeradas en esa iteración específica. Con base en la planificación de la liberación de las iteraciones, se pueden estimar las fechas para las iteraciones posteriores y una fecha aproximada del modelo final del proyecto.

Este proceso se realizó cada dos semanas, mediante un planeamiento iterativo, donde se realizaba una reunión para tener retroalimentación sobre la iteración actual del proyecto, también se discuten los cambios que deben realizarse en el backlog del producto, incluida la reasignación de prioridades de las características en el backlog del producto. Una vez que se ha completado este paso, el equipo procedía a asignar las diversas tareas (historias de usuario) discutidas en la reunión a los miembros del equipo.

La última etapa, de gestión del backlog como se menciona en [26], resume los cambios en el alcance, la prioridad y las estimaciones del proyecto; se puede modificar después de cada iteración según sea necesario para mejorar las características o eliminarlas. La prioridad de los elementos en el backlog del producto puede cambiar después de cada iteración de acuerdo con las necesidades y comentarios del cliente.

Mediante la metodología planteada, surgieron ramificaciones de trabajo, que iban desde el

estudio a fondo de la solución actual, hasta el entrenamiento relacionado a herramientas de desarrollo propias de la infraestructura de Intel; propuestas que dieran una solución integral entre las etapas que se desarrollaban en paralelo, las cuales dan pie a las siguientes secciones.

4.0.2. Caracterización del sistema actual

Partiendo del planteamiento del problema el cual expresa una necesidad de nuestros clientes de una nueva herramienta para el diseño de corners, como parte de la solución queríamos tomar lo mejor de la herramienta actual, y buscar herramientas de ingeniería que nos permitieran suplir las necesidades relacionadas a la gestión y compilación de datos, para ello se realizaron una serie de reuniones con los usuarios para definir posibles soluciones y prioridades, lo cual se resume a continuación.

Las prioridades de los clientes para nuestro proyecto son que haya un compilador único para todos los proyectos, el cual funcione siempre como mínimo igual al actual, y que la opción de optimización sea opcional, el compilador debe funcionar tanto para proyectos antiguos como para los nuevos, no hay un valor fijo para el tiempo máximo de compilación, pero que no sea más de cuatro o cinco minutos para todo un proyecto; en cuanto al control de versiones necesitan poder compartir sus áreas de trabajo, poder tomar el área de trabajo de un compañero, así como tener la posibilidad de actualizar sus datos de manera rápida a la última versión en caso de que otra persona haya hecho cambios en el repositorio principal, así como poder regresar a versiones anteriores del proyecto de manera sencilla. En cuanto a la interfaz, esta debe facilitar la visualización de parámetros a través de capas, lo que se explica en la siguiente sección, además de permitir la edición tanto del valor de los parámetros como de posibles condicionales que requieran agregar en una determinada jerarquía de configuración.

Con los requisitos iniciales de los clientes expresados en el planteamiento del problema y la lista de prioridades anteriormente mencionada, surge la duda ¿Qué se puede aprovechar de la solución actual? Ya que en la empresa se ha trabajado con el mismo modelo por años, se busca que la transición a una nueva herramienta fuera lo más orgánico posible, pero con una nueva arquitectura. Para lograr este cometido inició por analizar las entradas y salidas del compilador actual, así como su código, para hacer observaciones.

Mediante observaciones al método de compilación actual, se concluye que hay una optimización base, la cual parte de un corner predeterminado para todo el proyecto, el cual tiene un valor definido para todos los parámetros y se compila de primero, y para los siguientes corners el compilador hace una copia exacta del corner predeterminado y luego se sobrescriben los valores que sean necesarios. A la hora de compilar los parámetros, estos pueden tener condicionales, los cuales varían de proyecto en proyecto, y lo que se hace actualmente es que cada tipo de condicional tiene una etiqueta específica que el diseñador debe memorizar o tenerla anotada, y si el tipo de condicional que requieren no está en el compilador, tienen que ir a programarlo.

Ya que el compilador actual obtiene los datos a partir de un archivo de Microsoft Excel

(xls), este no puede contar con control de versiones de git, ya que este formato no es soportado por la herramienta; por otro lado, el compilador actual cuenta únicamente con la información del xls de proyecto, lo que implica que puede haber información que se comparta entre proyectos del mismo proceso que no se aprovecha, y debe definirse de cero en cada proyecto.

Las listas de requisitos, prioridades y observaciones dieron un panorama de los pasos a tomar, siendo el primero la habilitación del control de versiones para una nueva base de datos.

4.0.3. Implementación del control de versiones

En el marco teórico de definieron las bases de datos centralizadas y distribuidas, y cómo estas últimas se adaptaban a un control de versiones de Git, ahora bien, mediante los conceptos definidos de árboles y commits se tuvo que diseñar un modelo de trabajo que se adaptara a las necesidades de los clientes, para luego crear una serie de funciones que cumplieran con dicho propósito.
Basados en el análisis de la solución actual, tenemos que se cada cliente configura un proyecto por archivo, y lo que necesita es que pueda devolverse a versiones anteriores, actualizar su paquete de datos a la versión más reciente y poder compartir su diseño, además de que como se menciona en el planteamiento del problema, los datos de todos los proyectos deben estar centralizados.

Para cumplir la lista de requerimientos mencionada, se definió una metodología tal que cada usuario tuviera un 'área de trabajo' individual, pero cuando lo considerara oportuno pudiera compartir su configuración de corners con los otros desarrolladores, para esto se definieron dos repositorios, uno para funciones y otro para datos. El repositorio de funciones contiene el backend del proyecto, desde las funciones que llama la interfaz gráfica, hasta el control de versiones, y el repositorio de datos contiene la información ordenada por jerarquía, por ejemplo por proceso y proyecto.

Las áreas de trabajo anteriormente mencionadas parten del repositorio de datos, el de funciones solo recibiría cambios para actualizaciones o mantenimiento de la herramienta, dicha área es un branch del repositorio de datos, sobre el cual los usuarios definen un contexto, siendo el contexto cualquier proyecto o subproyecto que deseen configurar. Sobre este branch el usuario puede definir y experimentar parámetros como desee, y en el momento que lo considere le puede hacer un commit o push al branch, aquí es importante recalcar que no se está trabajando directamente sobre el repositorio principal, permitiendo así crear distintas versiones del contexto que está trabajando sin afectar a los otros usuarios, pero ya cuando tenga una configuración óptima que considere apropiada compartir con los otros usuarios, puede hacer push de sus corners al repositorio principal.

Cada usuario puede crear cuantas área de trabajo como lo considere necesario, lo esperado es que solo necesite una por contexto, pero de igual manera hicimos pruebas hasta con doscientas áreas de trabajo por usuario y no hubo problemas, ahora bien como en git para trabajar sobre un branch, se debe hacer un clon (configuramos la herramienta para que haga el clon de manera predeterminada), dicho clon es un directorio que puede ser compartido entre usuarios que tengan los permisos necesarios, y mediante el repositorio de funciones diseñado se pueden crear, editar, compartir, actualizar y restaurar contextos, permitiendo así una configuración flexible y robusta de distintas jerarquías de corners.
En conjunto con los repositorios, se implementó una base de datos de tipo SQL para el manejo de metadatos de las áreas de trabajo, la cual cumple el propósito de almacenar una identificación alfanumérica y una etiqueta que le asigna nuestro cliente, la cual se recomienda que contenga información relevante de la configuración de corners que está desarrollando, la etiqueta puede ser utilizada para identificar fácilmente el proyecto, la versión, el dueño del área de trabajo u otros atributos que se consideren necesarios.
A continuación se explica cómo interactúan los repositorios, en conjunto con la base de datos, para la implementación del control de versiones, partiendo de la generación de las áreas de trabajo, su actualización, y del cómo se comparten, restauran y detectan conflictos.

Generación de áreas de trabajo

Las áreas de trabajo al consistir en un contexto de desarrollo, ya sea global, de proyecto u otra jerarquía de desarrollo, se implementó una función basada en Git que toma como entrada el repositorio principal, en el cual se encuentran todas las posibles configuraciones disponibles, y a partir de ahí el desarrollador puede elegir en cuál trabajar, con base a esa decisión se genera el contexto, el cual es un branch del principal, pero enfocado en el proyecto que se desee trabajar. Además de ser un área en la que se pueden configurar los corners de acuerdo a los requerimientos de voltaje, tecnología, temperatura o interconexión que amerite el contexto, una vez se realizan los cambios, se habilitó el push del brach que hace la función de guardado, como se especifica en la siguiente sección.

Guardado del área de trabajo

Como se mencionó en la sección de Generación de áreas de trabajo, contamos con un repositorio principal, del cual los desarrolladores generan cuantos branches sean necesarios para el desarrollo de proyectos, pero ¿Cómo se mantiene el control de versiones si cada quien tiene su área o áreas de trabajo? Una vez que se realizan cambios al proyecto en cuestión, el desarrollador puede hacer un push de su área (branch) al repositorio principal, una vez que los cambios están en el repositorio principal, otros desarrolladores

pueden actualizar sus datos mediante un Git pull, estas actualizaciones pueden ocasionar conflictos propios de git cuando es modificado un parámetro en la versión en la cual se trabaja y al mismo tiempo fue actualizado en el push, para estos casos se implementaron mensajes que alertan al usuario.

Áreas de trabajo compartidas y restaurar sesiones

Una petición de los usuarios es que se puedan compartir áreas de trabajo sin necesariamente hacer un push, para ello los usuarios de la herramienta tienen permisos específicos, que les permiten un desarrollo en paralelo de un área de trabajo, de esta manera pueden compartir un contexto y hacer experimentos en conjunto para los distintos escenarios en los que puede llegar a operar el circuito. Para gestionar las distintas configuraciones que pueden crearse a lo largo del diseño de corners de un circuito, se implementó una función para el reporte de versiones, la cual genera una lista para que los usuarios puedan escojer cuál visualizar o editar.

4.0.4. Compilación

El algoritmo de compilación consta de dos etapas principales, la primera es la de consolidación, el cual es un intérprete de las capas de configuración que se ven involucradas en un contexto, tomando en cuenta la precedencia de cada una de ellas y si se encuentran activas, la segunda etapa consta de la optimización de los parámetros y corners en formato TCL, y mediante observación, experimentación y análisis encontrar una configuración, como se menciona en el objetivo general del proyecto, que optimice en al menos un cinco por ciento el TCL compilado, el cual contiene toda la información de los corners para un proyecto.
Implementar una solución basada en el diseño por capas asemeja el particionamiento descrito en el marco teórico, ya que lo que buscamos es descomponer el problema del diseño de corners en otros más pequeños, que para nuestro propósito son cada una de las capas de configuración, y una vez que todas estén definidas aplicar el algoritmo de compilación para consolidar lo definido a través de ellas.

Configuración por capas

Como se mencionó en la introducción y en el Estado del Arte, actualmente los clientes cuando inician un proyecto, la metodología que siguen es tomar como base un corner de referencia, comúnmente llamado "default" y sobreescriben uno a uno los parámetros que sean necesarios en cada corner, lo que genera una serie de dudas a la hora de buscar una solución, siendo la primera ¿El corner predeterminado tiene el valor óptimo para cada parámetro, el que más se repite? En el momento en el que el corner default no tenga el valor que más común a través de los corners, tendrá implicaciones tanto a la hora de almacenar datos como a la hora de compilar, porque una característica que se busca es que cuando hay un único corner default este cubra la mayoría de los casos, un ejemplo de esto sería que si se desea habilitar en el noventa por ciento de los corners un parámetro que permita el análisis de parasíticos y sus coordenadas, lo mejor sería que el valor default sea activo, sino habría que modificar manualmente en la mayoría de escenarios.

La segunda pregunta sería ¿Cómo se puede optimizar la definición de los corners? A solicitud del cliente, como valor agregado del trabajo final de graduación, se trabajó en una solución que permitiera a los usuarios reciclar configuraciones realizadas en diseños anteriores, por ejemplo si se inicia un nuevo proyecto, de un mismo proceso que ya se hubiera trabajado en el pasado, sería útil tener una herramienta que le permitiera importar al nuevo proyecto un escenario conocido y solamente modificar los parámetros que sean necesarios, este concepto se puede extender aun más, ya que se podrían importar corners por planta de fabricación, o dado el caso, de configuraciones globales.

Este cambio de metodología de configuración conllevó un reto para el proyecto, ya que el nuevo compilador no solo debía optimizar la cantidad de líneas del TCL de salida, sino que también debía integrar un sistema de configuración por capas, ya que ahora no todos los parámetros se definen a nivel de proyecto, sino que una porción de los parámetros se puede definir a nivel de planta de fabricación, otros a nivel de proceso, y otra parte se mantienen específicos para el proyecto y sus posibles subproyectos o IP. Para llevar este concepto de capas a la práctica había que definir maneras para llevar registro de dónde se realizan los cambios de los parámetros, visualización, control de versiones por área de trabajo, y su valor final, lo que involucra las funciones para la configuración de parámetros y visualización en tiempo real de los escenarios, cada escenario pasó a ser un archivo tipo JSON de configuración.

Por la manera en la que se definen los proyectos, independientemente de cómo de definan los corners, estos forman parte de una jerarquía de configuración, por lo que se busca una solución acorde a este hecho, ya que la herramienta anterior presentaba limitaciones que no le permitían acceder por completo a las ventajas que presentan el diseño por sobre escritura, por ejemplo, cada vez que se define un nuevo proyecto, este forma parte de un proceso, que por lo general define qué bibliotecas de celdas se pueden utilizar dada la

tecnología y la densidad de transistores; a su vez dicho proceso es propio de una planta de fabricación, la que puede ser interna o externa a la compañía, esto conlleva una distinta parametrización de los escenarios y por consiguiente más variables que la solución contempla, para tener la capacidad de utilizar un solo compilador independientemente de la tecnología o fábrica. Así como el proyecto pertenece a un proceso y el proceso a una planta de fabricación, todo lo que se comparta a nivel de fábricas se encuentra en una capa global, así como se muestra en la Figura 4.1 cada anillo interno tiene precedencia sobre uno externo.

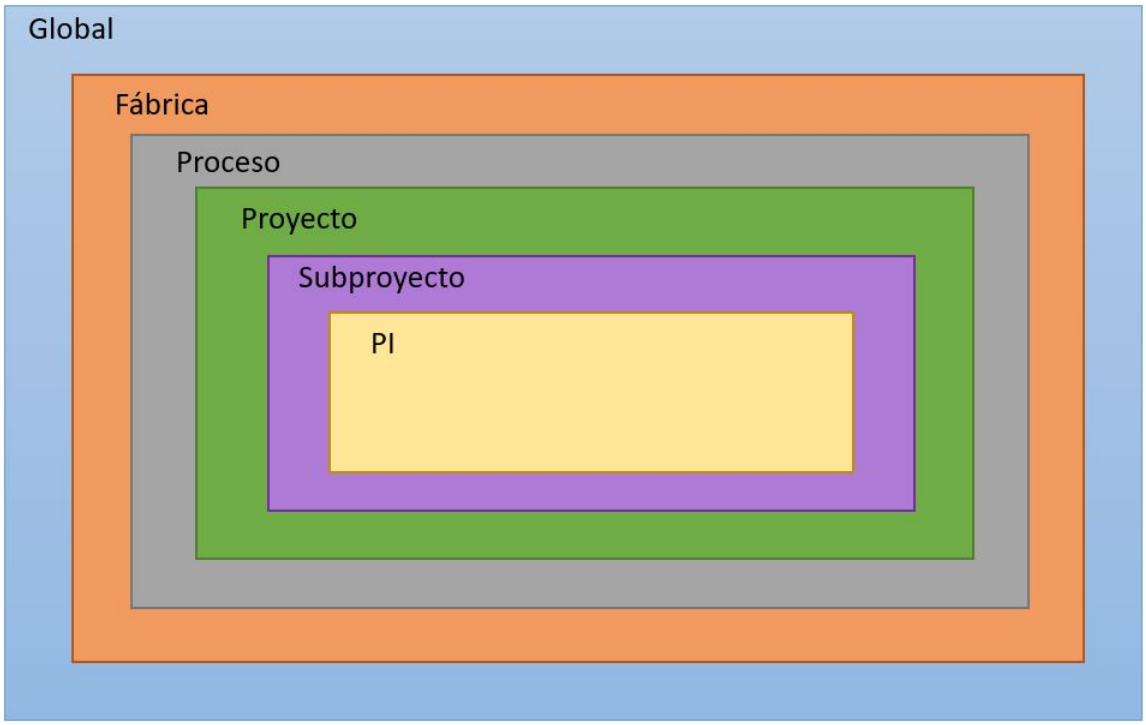


Figura 4.1: Capas de Configuración

Mediante la Figura 4.1, se introduce el concepto de capas superiores e inferiores, llámese capa superior a cualquiera que se encuentre en un anillo exterior de la que se tome como referencia e inferior a cualquiera de un anillo interno. En 4.1 muestran dos capas más de las cuatro ya descritas, las cuales son subproyecto y IP, las que pertenecen a un proyecto, las cuales sirven como punto de partida definir que no todas las capas superiores deben tener una configuración para que funcione alguna inferior, por ejemplo, por arquitectura se definió una precedencia de subproyecto sobre IP, pero puede existir un IP que pertenezca directamente a proyecto, y que no esté relacionado a algún subproyecto.

La precedencia de capas es un punto clave para el funcionamiento del sistema, ya que cualquier valor que se define en una capa superior, se puede sobre escribir en una inferior y tanto el compilador como la interfaz gráfica tienen la capacidad de llevar registro de estos cambios. Trabajar por capas de configuración en JSON no es un concepto nuevo para el cliente, ya que hay otras herramientas de desarrollo que cuentan con este tipo de

```
{
    "enable_corner": 1,
    "reference_corner": "None",
    "parameters": {
        "temperature": {
            "status": 1,
            "latest_hier": "project",
            "value": 65,
            "flows": [
                "opt",
                "pv"
            ],
            "overwrites": {
                "condition": ["condition_1", "condition_2"],
                "result": ["result_1", "result_2"]
            },
            "comments": "NO COMMENTS"
        }
    }
}
```

Figura 4.2: Estructura de un corner

arquitectura, pero lo que si era necesario era definir una estructura de configuración que se amoldara a las necesidades actuales y previera en la medida de lo posible, los requerimientos de futuros proyectos, dando como resultado la estructura de corner mostrada en la Figura 4.2.

Todos los corners independientemente del proyecto siguen la estructura presentada en 4.2, esta fue diseñada para agregar atributos que permitan tanto la legibilidad de las capas, así como su configuración y descripción, algunos de estos atributos son propios del corner y otros de los parámetros, lo que se sintetiza en la Tabla 4.1.

Atributo	Descripción
enable_corner	Habilitación del corner, esta llave cuando está activa indica que el corner debe ser compilado.
reference_corner	Corner de referencia, puede usarse el default, otro o ninguno. Se habilitó como una prevista para futuras mejoras de la interfaz y el compilador.
parameters	Incluye solamente los parámetros que son modificados en la capa actual.
≪attribute≫	La llave de attribute es variable, ya que es el identificador del parámetro, este incluye su contexto y atributo por configurar.
status	Habilitación del parámetro, esta llave cuando está activa indica que el parámetro debe ser compilado.
latest_hier	Última jerarquía modificada, esta variable es utilizada para mejorar la legibilidad de las capas e identificar fácilmente la última capa en la que un atributo tuvo cambios, la de mayor precedencia
value	Valor del atributo
flows	Etapas del flujo ASIC que afecta la configuración del atributo
overwrites	Condiciones especiales de uso para el parámetro, ya que el valor de este puede depender de otras variables
condition	Lista de condiciones asociadas al parámetro
result	Lista de posibles resultados asociados al parámetro si se cumple la condición respectiva, los índices de la lista de condition corresponden al mismo índice de result
comments	Comentarios del usuario para llevar registro de los cambios, o bien recomendaciones de diseño

Tabla 4.1: Atributos del JSON de corner

Basado en las definiciones de la Tabla 4.1 el JSON de la Figura 4.2, sería el caso de un corner activo a nivel de proyecto ya que *enable_corner* está activo y *latest_hier* es project, además cuenta con dos parámetros que son modificados a esa jerarquía: *voltage* y *temperature*, ambos parámetros están activos para la capa ya que su *status* es 1; el primer

parámetro *voltage* cuenta con un valor de 0.7 y afecta dos flujos, el de opt, el cual está relacionado a las etapas del flujo ASIC que van desde síntesis hasta route, como el de pv que se utiliza para signoff. Ninguno de los parámetros cuenta con comentarios, y solo el de *temperature* cuenta con condiciones de uso para el parámetro, lo cual se interpreta como, si se cumple *condition_1* se configura *result_1*, pero si se cumple *condition_2* se debe configura *result_2*, el valor por defecto del parámetro siempre es el *value* a menos que se cumpla un condicional.

El compilador el cual se expone en la siguiente sección, hace el análisis de atributos para todos los corners a través de las capas, y genera un resultado considerando la precedencia y las sobre escrituras, que pueden ser a cualquier atributo de la Tabla 4.1, por ejemplo a nivel de proceso un corner puede tener el *enable_corner* desactivado, pero a nivel de proyecto estar activo y con un *value* distinto, y en subproyecto agregarse un comentario, o cambiar un condicional, cualquier combinación que requiera el ingeniero a cargo, mientras se encuentre dentro de los atributos disponibles.

Algoritmo de compilación y optimización

El diseño del compilador fue llevado a cabo en dos modalidades, la primera que iguala en líneas de TCL al anterior expuesto en el estado del arte, que toma los datos a partir de excel y genera el archivo de configuración en TCL mediante un compilador en lenguaje Perl; pero con la nueva metodología basada en JSON; el segundo modo es el de optimización, este busca reducir la cantidad de líneas de configuración del TCL, así como se describe en el objetivo general del proyecto.

Habilitar una opción para compilar de manera similar a la anterior cumple dos funciones, la primera es facilitar la transición a la nueva herramienta de diseño, ya que al implementar cambios mediante la configuración por capas, interfaz gráfica y control de versiones, la alternativa de compilar las capas y obtener el mismo archivo de configuración al que está familiarizado, fue una decisión de diseño que se integró para acompañar al usuario a transicionar su trabajo a la solución que propone este proyecto. El segundo motivo para habilitar en el nuevo compilador la opción de generar un archivo de configuración que siguiera el mismo formato que el anterior, fue para facilitar pruebas y verificación manual, ya que el algoritmo de verificación automático fue desarrollado posteriormente cuando ya se tenía una definición de capas, configuración y compilación estable.

La principal función del compilador es tomar los corners almacenados en una base de datos y traducirlos a un formato TCL para que pasen a ser parámetros de configuración. Ahora bien, la solución se planteó a partir de las entradas y salidas del compilador antiguo, ya que la lógica que se podía reutilizar de este era escasa, dado a la diferencia en los archivos de entrada y entre lenguajes de programación, uno en Perl y otro en Python.

La salida, el archivo en formato TCL, tiene una estructura básica la cual debe cumplir para ser interpretado, la cual consta de cinco etapas en general, que se resumen en la Tabla 4.2, de las cuales dos fueron modificadas para lograr la optimización que busca el proyecto.

Etapa	Descripción
Limpieza de valores previos	Se asignan las listas de corners y parámetros en nulo, así como los valores que haya definido el intérprete del TCL
Inicialización de corners	Se inicializan todos los corners del proyecto a modo de lista
Inicialización de atributos	Se inicializan todos los parámetros del proyecto a modo de lista
Configuración del corner de referencia	Se inicializa un corner para ser tomado como referencia y todos sus parámetros, quien clone de este corner va a tener los valores de este corner por defecto
Configuración de los corners clon	Se inicializan los corners, clonan del que tengan como referencia y luego se defienen únicamente los valores que varían a su valor por defecto

Tabla 4.2: Estructura base del TCL

Las tres primeras etapas del TCL: Limpieza de valores previos, inicialización de corners e inicialización de atributos, mostradas en 4.2 se mantienen igual con y sin optimización, ya que son requeridas por el intérprete a la hora de inicializar variables; por otro lado, tanto la configuración del corner de referencia como los corners clon fueron el punto de partida para la optimización.

En el estado del arte, se cuenta con una configuración con base en un solo un corner de referencia, al cual el resto de corners usan para clonar; al clonar un corner lo que se hace es copiar el valor de todos los parámetros, para luego sobre escribir los que difieren. Al trabajar solamente con un corner de referencia, lo que puede suceder es que clonen corners con baja correlación, ya que pueden variar en derates, parasíticos, bibliotecas, o hasta en los mismos valores de voltaje, proceso y temperatura, lo que ocasiona que se tengan que sobre escribir la gran mayoría de parámetros.

La solución gira en torno a un algoritmo de compilación que provea una alta correlación entre el corner de referencia y el clon, cabe recalcar que para la metodología anterior se podían usar varios corners de referencia, pero al ser una revisión manual, en la mayoría de los casos se terminaba descartando.

Para poner a prueba la hipótesis que se tenía hasta ese momento de si aumentando la correlación entre el corner de referencia y los que clonan de este, disminuye la cantidad de sobre escrituras, se diseñó el compilador de tal manera que tuviera la capacidad de trabajar con múltiples corners de referencia, y que agrupara corners dado un parámetro.

La agrupación de corners se basa en un atributo de configuración, que se le pasa por parámetro al compilador, lo que hace el algoritmo primeramente es definir los diferentes valores que toma el parámetro en el contexto, luego cada uno de estos valores se utiliza para agrupar los corners que tengan el mismo valor, y por último se escoge arbitrariamente un corner por grupo que va a servir como referencia para los que compartan el mismo valor de ese parámetro . Por ejemplo en un proyecto que contiene 174 corners y 363 parámetros por corner, dentro de los que hay 14 voltajes distintos y 16 procesos de extracción, lo que quiere decir que el algoritmo de optimización lo que haría en vez de definir un solo corner de referencia, haría 14 si se compila con respecto a voltaje, si se hace con respecto proceso de extracción habrían 16, y así sucesivante con cualquier parámetro que se tome como referencia.

El algoritmo de que sigue el compilador, se muestra en el diagrama de flujo de la Figura 4.3 con etapas que se ejecutan independientemente si hay optimización, para luego tomar decisiones basado en las entradas del sistema, que son el proyecto y el parámetro de referencia. Para la experimentación se ejecutó este algoritmo con todos los parámetros disponibles por proyecto, para así encontrar uno o varios parámetros comunes a través de proyectos que dieran la mejor optimización.

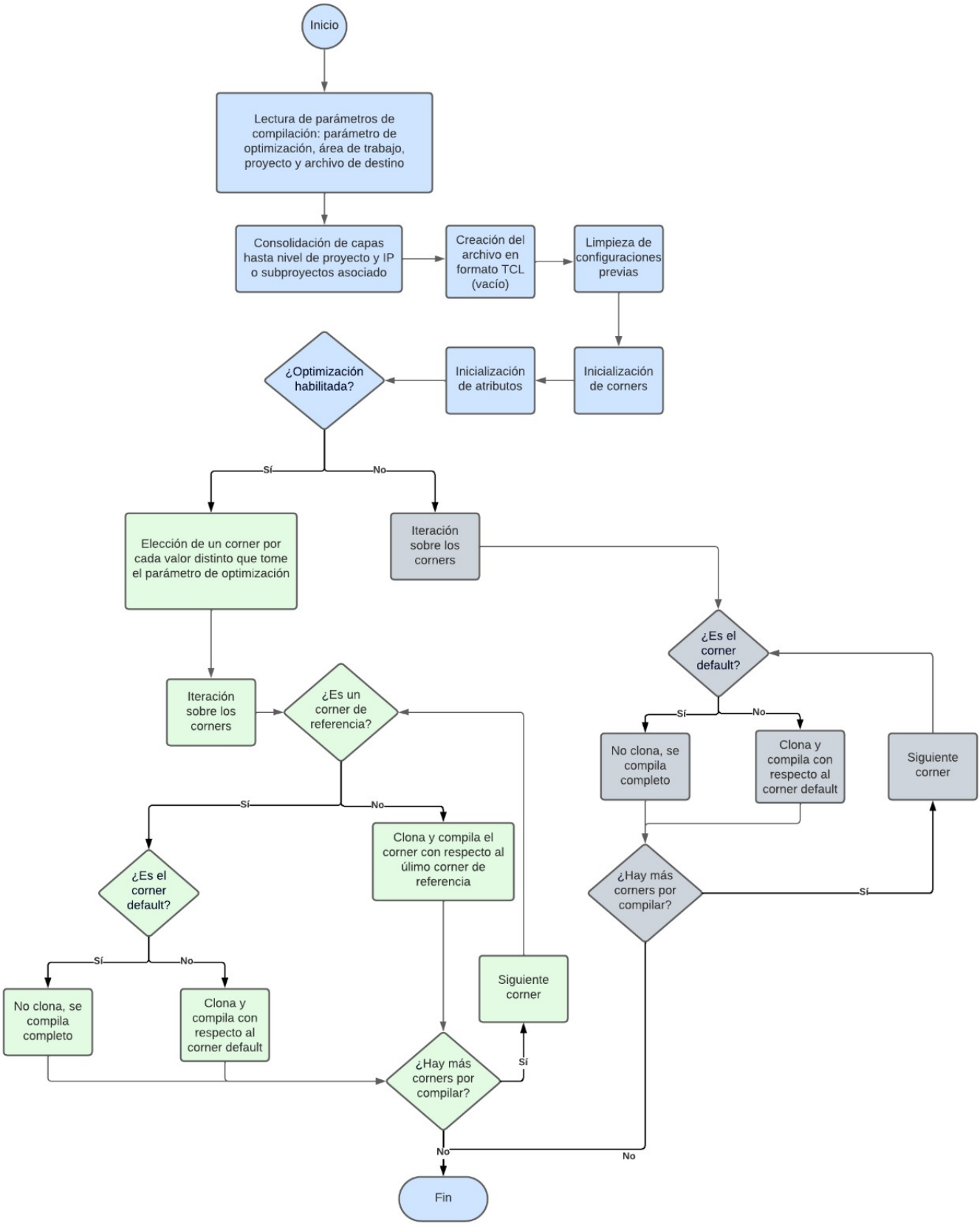


Figura 4.3: Algoritmo de compilación

En la Figura 4.3 se tiene de color verde las etapas que son exclusivas del proceso cuando no se habilita la optimización, por otro lado las etapas de color celeste se ejecutan cuando se habilita la agrupación de corners por parámetro de configuración, por último las de color gris son aquellas etapas que se comparten por ambas metodologías.

El algoritmo primeramente consolida las capas hasta nivel de proyecto, y las de supproyecto o IP en caso de que existan, el resultado de cada consolidación se almacena como un objeto en una lista, seguidamente inicializa el archivo en formato TCL, para luego ejecutar las etapas de limpieza de valores previos, inicialización de corners e inicialización de atributos independientemente si se optimiza como lo indica su color gris.

Posteriormente se revisa revisa si está habilitada la compilación mediante un parámetro de configuración, si este es el caso se ejecuta ejecuta la agrupación por corners y establece los de referencia, una vez con esa lista se interan los corners, el *corner default* es el primero que se compila, y este no clona de ningún otro, el resto de corners de referencia se clonan y compilan con respecto al *corner default*. El restante de corners clonan de su corner de referencia, con el cual comparten su valor para el parámetro de referencia.

En el caso de que no haya un parámetro de configuración para referencia, se ejecutan las etapas de color verde que consisten en compilar el *corner default* completo y luego el resto de corners clonan de este.

Para identificar los parámetros que optimizan el algoritmo de la Figura 4.3 , se llevó a cabo un experimento con distintos proyectos, que consistió en probar el compilador con cada uno de los parámetros del proyecto como referencia, lo que generó cientos de archivos de configuración TCL, tal y como se muestra en la Figura 4.4.

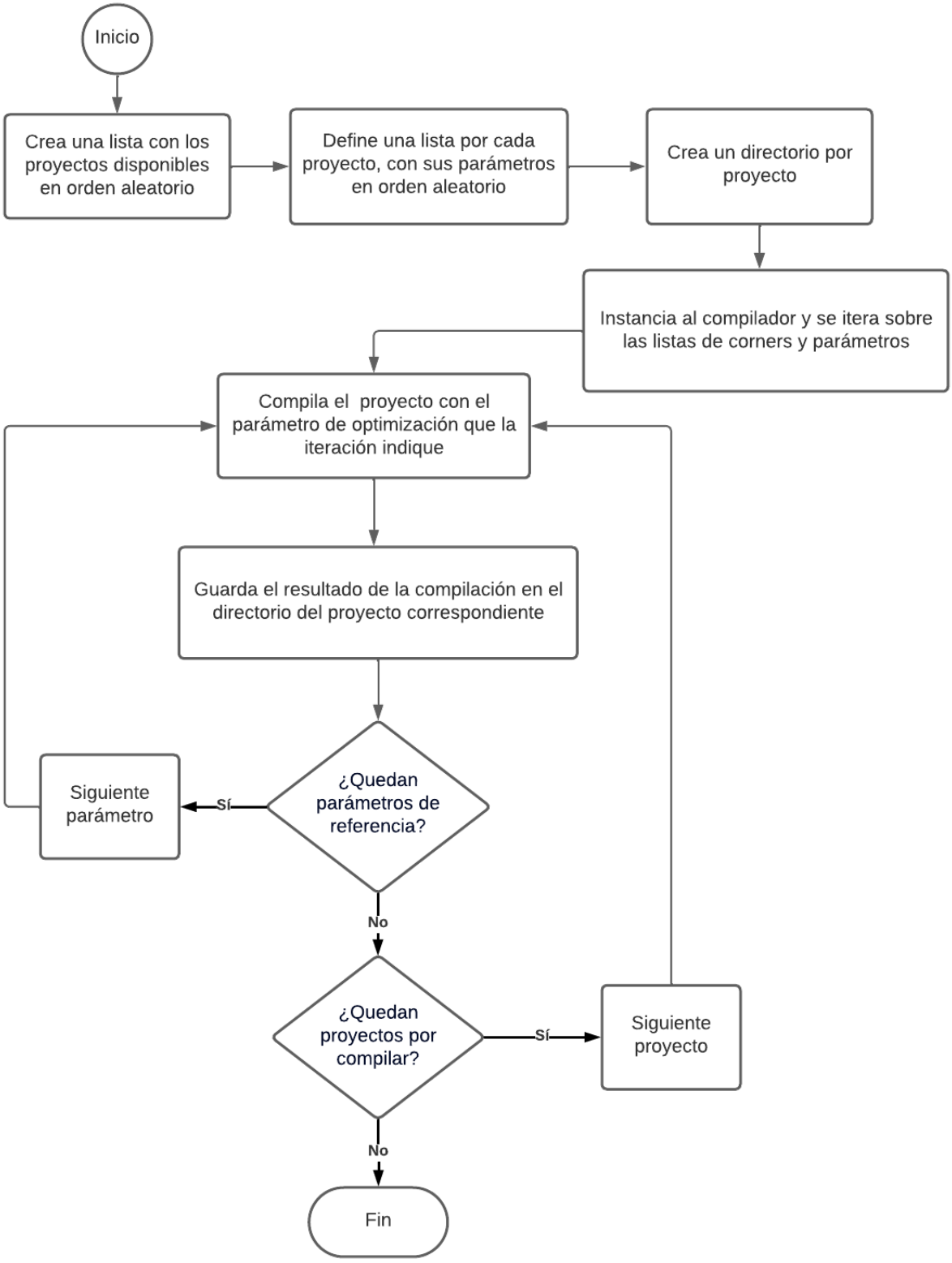


Figura 4.4: Algoritmo para experimentación

4.0.5. Verificación

El algoritmo para experimentación 4.4 se llevó a cabo en tres ocasiones, y como este lo indica, se compila un archivo de configuración por cada parámetro de cada proyecto, lo que generó miles de archivos en formato TCL, por lo que su revisión tuvo que automatizarse. La verificación parte del principio de que todos los archivos de configuración de cada proyecto, sin importar cuántas líneas tenga, al ser interpretados por el mismo ambiente de variables, el valor final de cada parámetro en cada corner, debe ser igual a si se compilara ese proyecto con la metodología anterior. El algoritmo que toma cada uno de estos archivos de configuración y los inicializa en un ambiente de variables se muestra en la Figura 4.5.

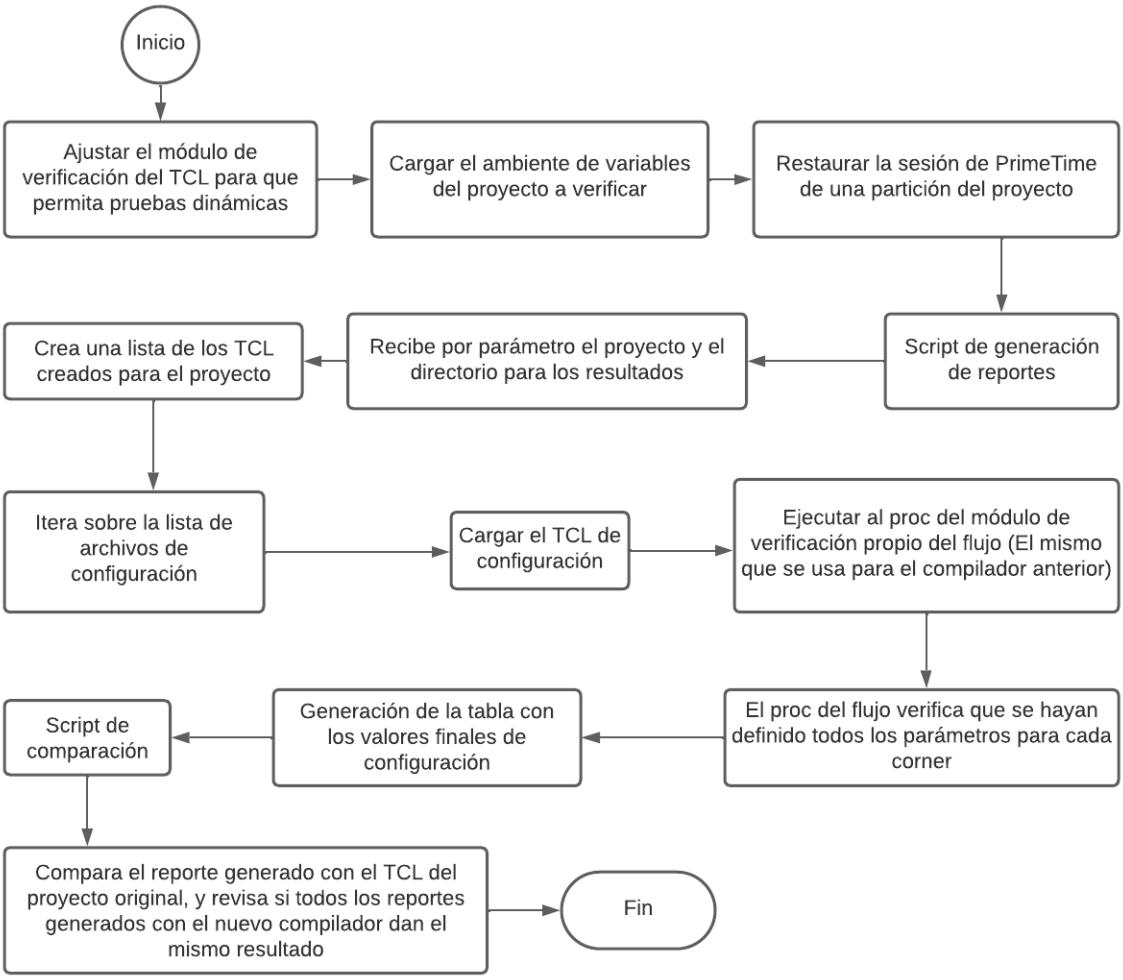


Figura 4.5: Algoritmo de verificación

4.0.6. Interfaz gráfica de usuario

La interfaz gráfica centraliza las funciones para la configuración tanto de de corners como de sus parámetros, permite crear áreas de trabajo compartidas para el desarrollo conjunto de proyectos bajo control de versiones, una vez creada el área de trabajo el usuario puede escoger si trabajar en un proyecto existente o bien crear uno nuevo, mediante un catálogo de los proyectos disponibles, para así definir un contexto. Una vez definido el contexto el usuario puede escoger si la edición la realiza dentro o fuera de la interfaz, ya que se habilitó la opción de importar y exportar archivos de configuración en formato csv; las herramientas de diseño disponibles en la interfaz se describen a continuación.

Área de trabajo

La configuración de corners inicia por escoger el área de trabajo, en la Figura 4.6 de *Work Area* los usuarios pueden realizar múltiples acciones con los botones de *options*, de izquierda a derecha son para abrir el area de trabajo (flecha diagonal), compilar el contexto asociado al área de trabajo seleccionada (engranaje), compilar y cargar al repositorio principal el área de trabajo (triple engranaje), y por último las opciones de exportar (flecha hacia abajo) e importar (flecha hacia arriba) respectivamente.
En las otras columnas tenemos la de *id* que contiene un identificador alfanumérico para el área de trabajo asociada, el cual se utiliza para hacer consultas a la base de datos, el *user_name* es quien haya creado el área de trabajo.
El control de versiones del área de trabajo se lleva mediante el *fb_name* el cual es el feature branch en GIT, la columna de *clone_path* indica el directorio donde se encuentra el repositorio de datos asociado al área de trabajo, y el *custom_tag* es una etiqueta que asigna el usuario para identificar el área de trabajo.

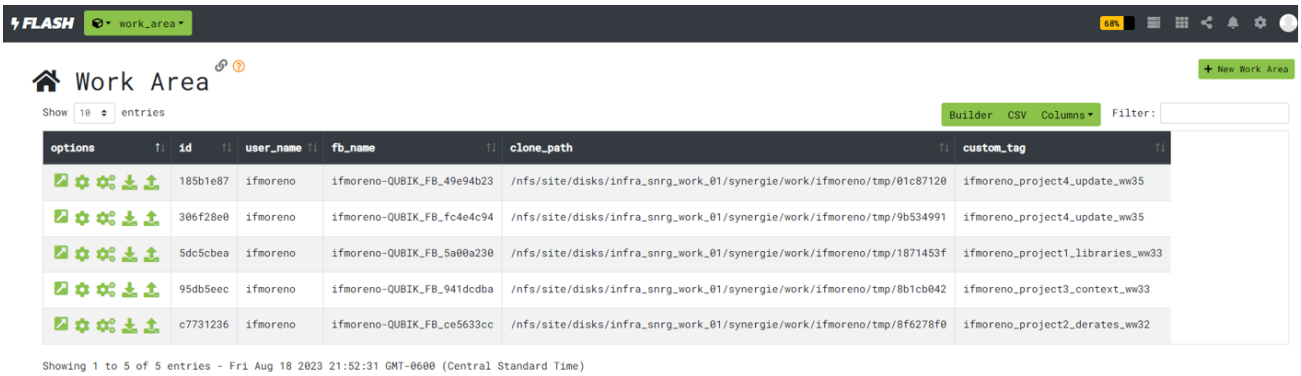


Figura 4.6: Interfaz gráfica: Áreas de trabajo

La ventana de *Work Area* con el botón de *+New Work Area* abre la siguiente pestaña 4.7, donde el usuario puede crear una nueva área de trabajo y asignarle la etiqueta correspondiente, en este caso la de "ifmoreno_project1_tfg_ww33".

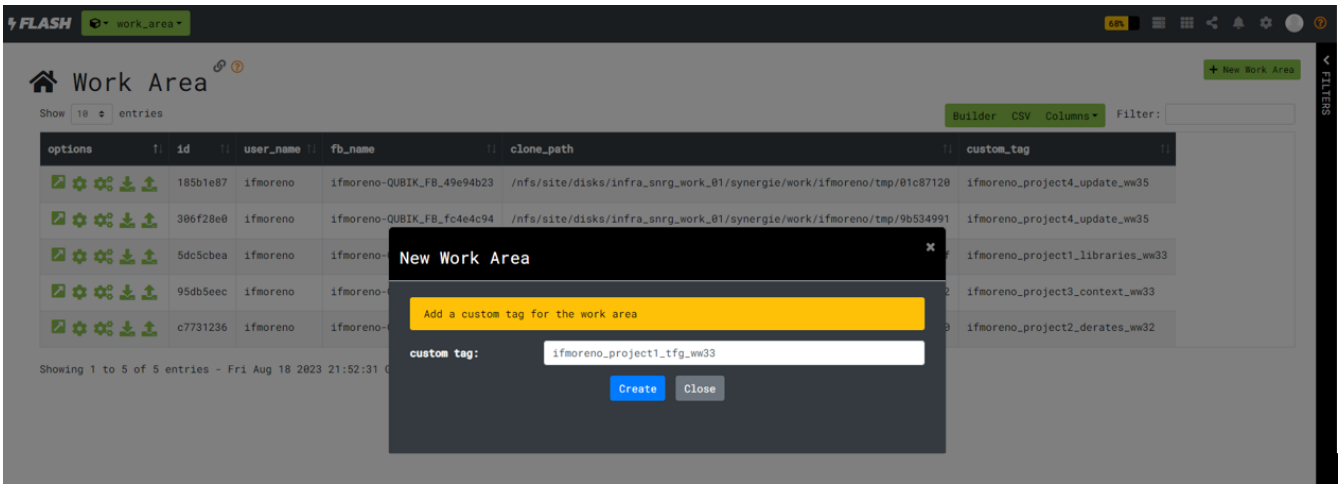


Figura 4.7: Interfaz gráfica: Nueva área de trabajo

Proyectos

La ventana de proyectos de la Figura 4.15, es accedida cuando se pulsa el botón de abrir área de trabajo y no existe el contexto asociado, en esta ventana el usuario puede escoger un contexto ya existente, que sería cualquier fila de la tabla y abrirlo, compilarlo, subirlo al repositorio principal igual que en *Work Area* con los botones de options o bien crear uno nuevo con el botón de *+New Project*.

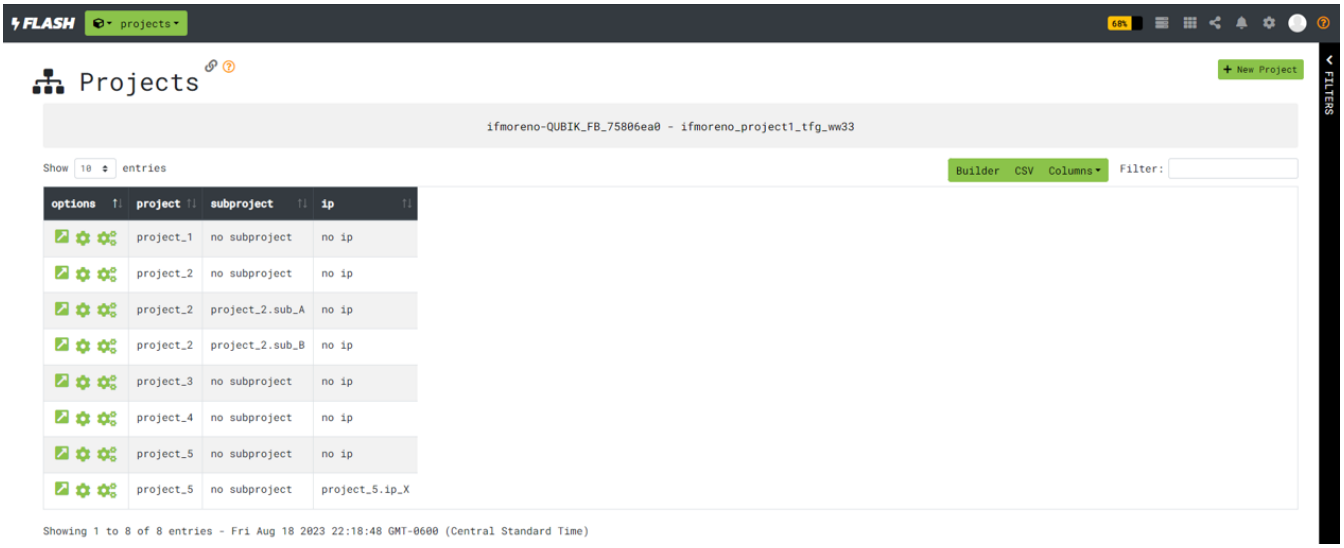


Figura 4.8: Interfaz gráfica: Proyectos

Al pulsar el +*New Project* en 4.15, se abre la pestaña de la Figura 4.9, donde el usuario crea un proyecto con base en otro mediante una expresión regular, las expresiones habilitadas para referencia son *proyecto*, *proyecto.subproyecto* y *proyecto.ip*, para así copiar las capas correspondientes.

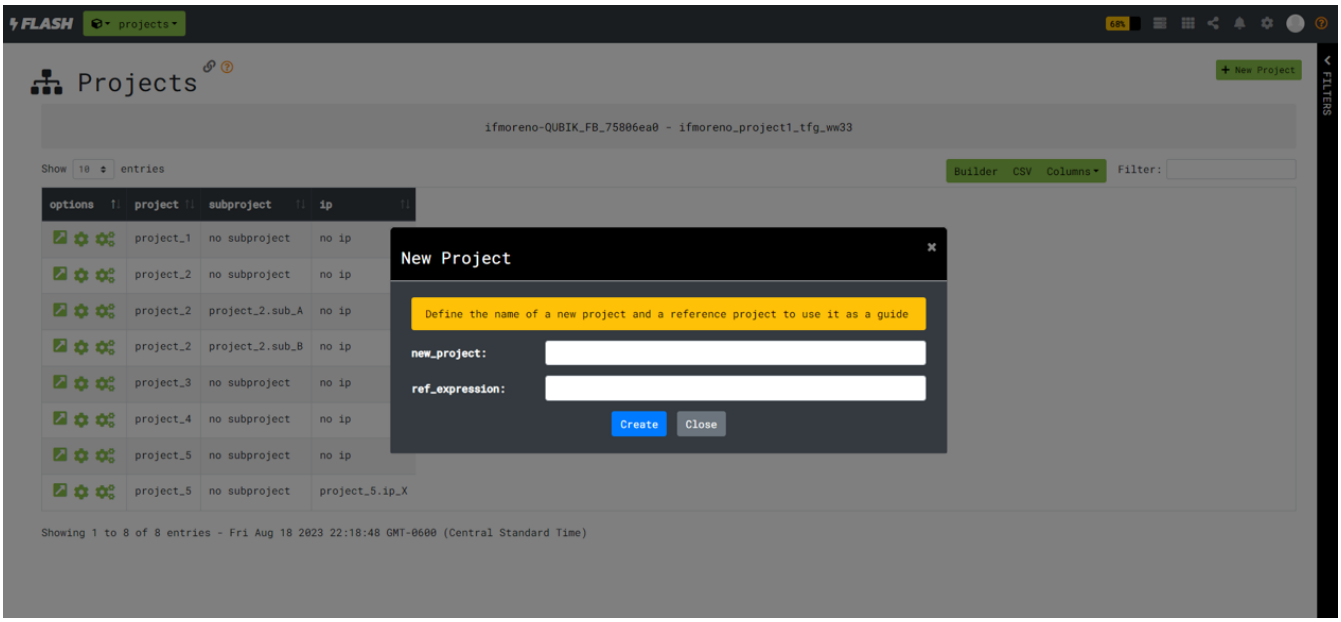


Figura 4.9: Interfaz gráfica: Nuevo proyecto

El contexto seleccionado se guarda en la base de datos, y se asocia al área de trabajo, una vez hecho esto, el usuario puede realizar modificaciones al contenido del contexto ya sea mediante la interfaz o exportando los archivos de configuración JSON a csv, modificarlos y luego importarlos de nuevo para actualizar los valores tanto de corners como de sus parámetros, el formato que sigue el csv es de: capa, corner, tipo (valor o conidiconal), estatus (activo o inactivo), valor, flujo y condicional (este es opcional), en el csv se puede tanto editar, filtrar o comparar tanto corners como parámetros.

Menú de corners

La ventana de *All corners*, Figura 4.10 puede ser accedida mediante dos formas, la primera es a través del botón de abrir el área de trabajo que se encuentra dentro de la ventana *Work Area* una vez seleccionado el contexto asociado, o bien desde la ventana de *Projects*, mostrando como resultado todos los corners disponibles para edición.

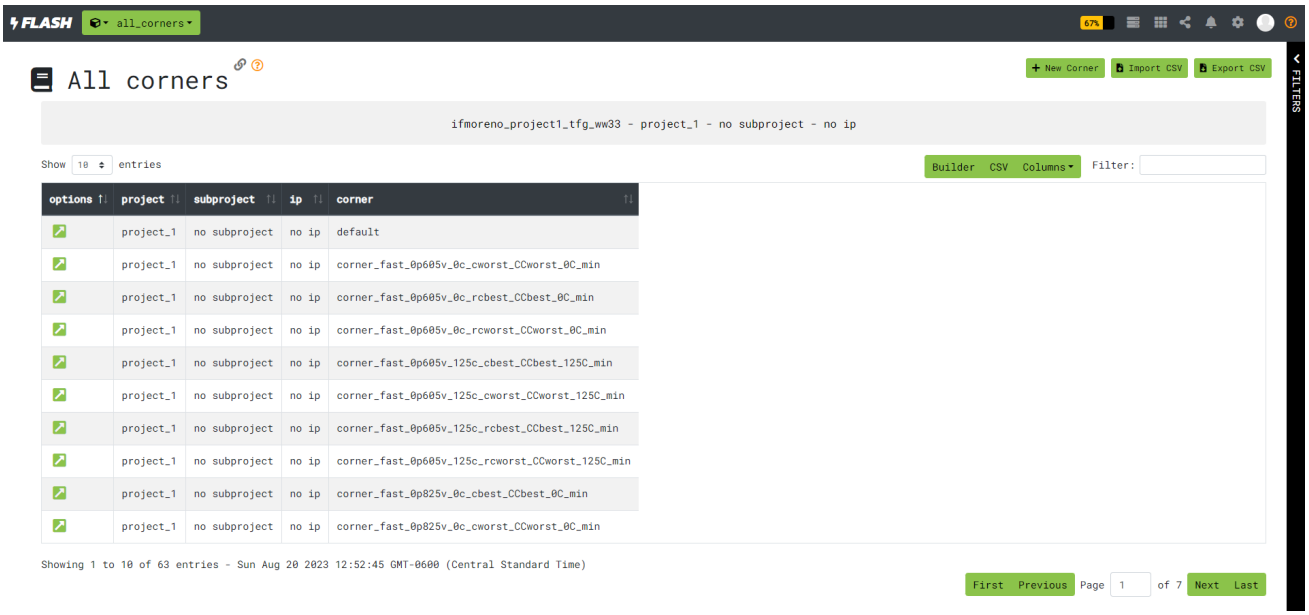


Figura 4.10: Interfaz gráfica: Todos los corners

All corners muestra el contexto actual en el báner gris, los botones de la tabla permiten abrir cualquier corner para su configuración, lo que se haría al presionar el botón de la fila correspondiente en la tabla, o crear uno nuevo con el botón de *+New Corner* que despliega la pestaña de la Figura 4.11. En la parte superior de *All corners* se habilitó importar y exportar a formato csv el contexto, con los botones de *Import CSV* y *Export CSV*.

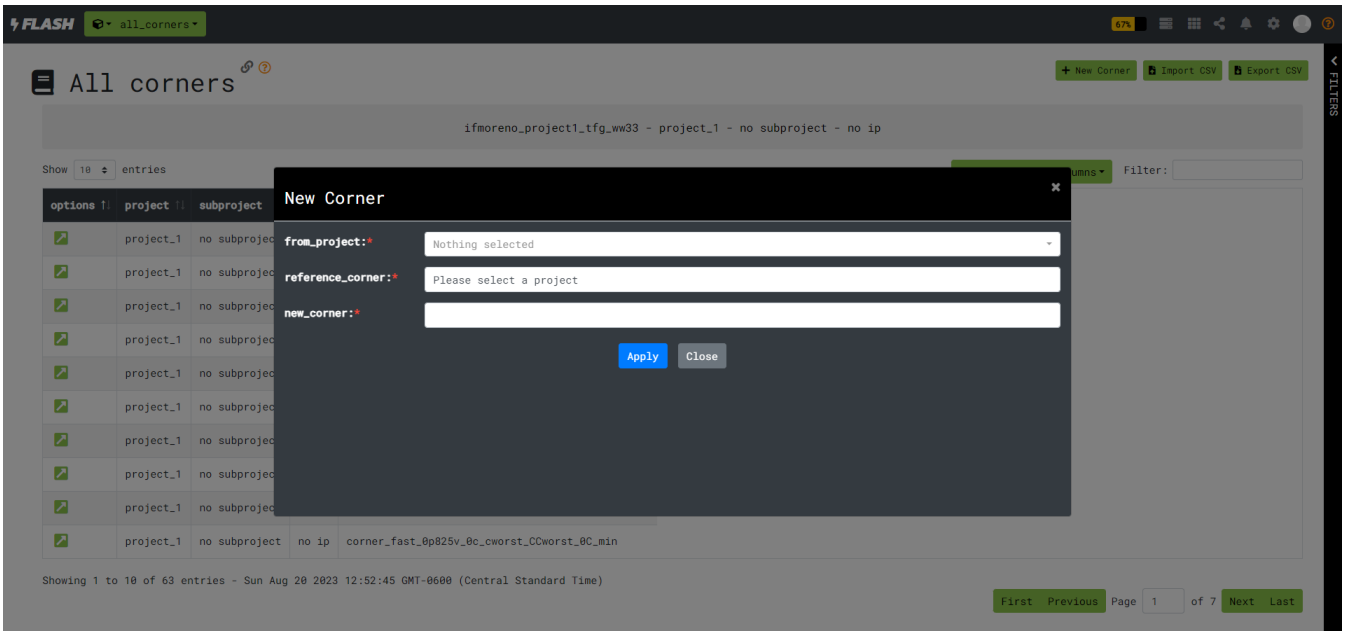


Figura 4.11: Interfaz gráfica: Nuevo corner

La pestaña para crear un nuevo corner de la Figura 4.11 a petición del cliente se creó a modo de catálogo, donde se puede escoger cualquier proyecto del repositorio y una vez hecho esto, el usuario puede elegir un corner de ese proyecto para copiarlo en el contexto actual con la etiqueta que desee.

Configuración de corner específico

Una vez se selecciona un corner del menú 4.10, la herramienta carga las capas de configuración del escenario, sus parámetros y condicionales, para así mostrar la ventana de *Unique Corner* de la Figura 4.12, esta ofrece en la columna de *options* botones para editar tanto valores como condiciones de uso del parámetro (ícono del lápiz), así como agregar nuevas condiciones desde cero o basadas en otras (ícono bloques), o bien deshabilitar ya sea un condicional o todo el parámetro (ícono basura), y búsquedas con la barra de *Filter*.

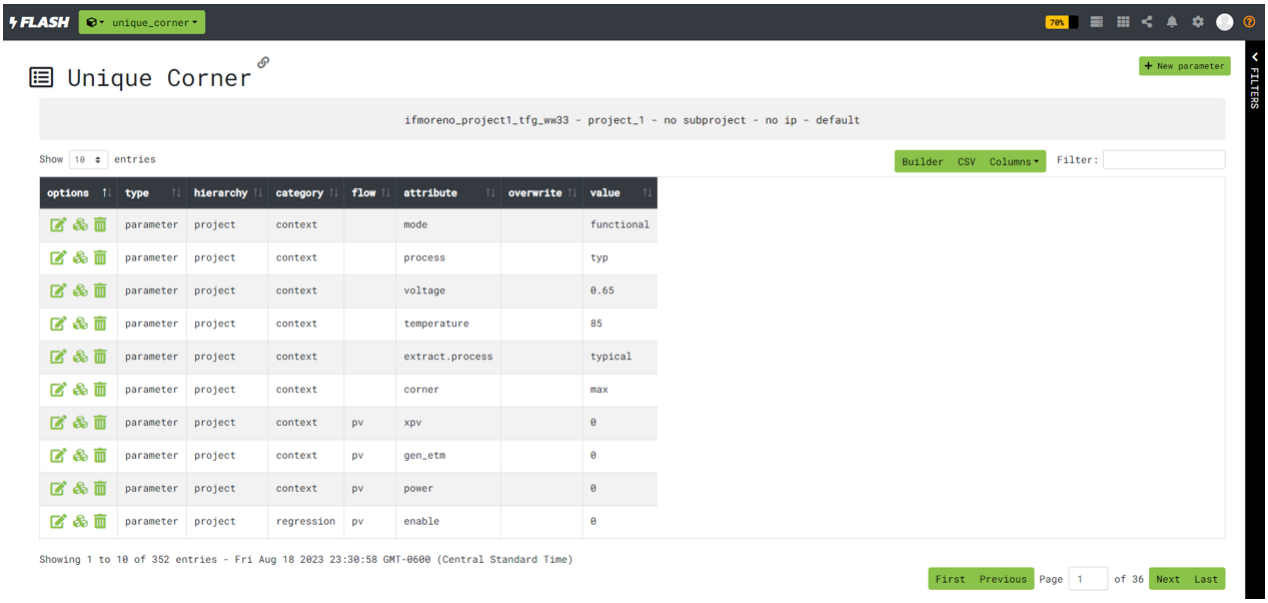


Figura 4.12: Interfaz gráfica: Parámetros de un corner específico

En el caso que se requiera editar alguno de los parámetros del corner, al presionar el primer botón de options, se muestra la pestaña de la Figura 4.13 donde se auto completa el parámetro seleccionado en la casilla de *parameter* y el valor actual en *value*, para el que el usuario lo edite de acuerdo a las necesidades del proyecto.

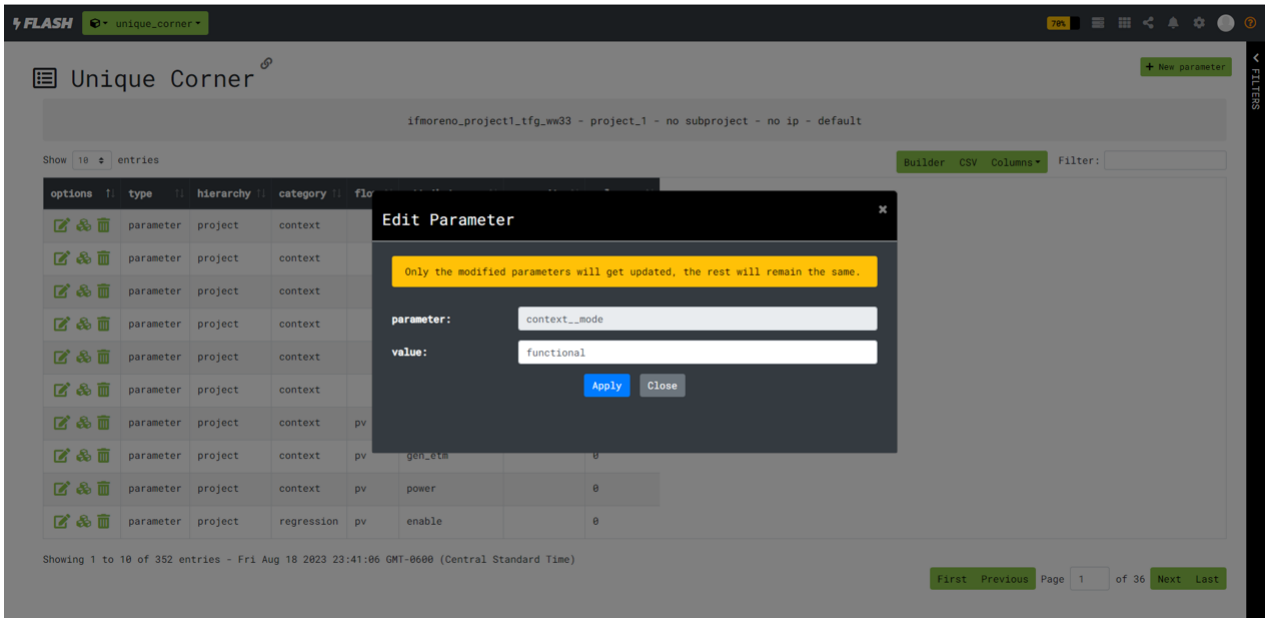


Figura 4.13: Interfaz gráfica: Editar parámetro

La Figura anterior 4.13 nos muestra la pestaña en caso de que el usuario desee cambiar solamente el valor de un parámetro, en caso de que se requiera agregar condiciones de uso lo puede hacer en la pestaña de la Figura 4.14, que se despliega al pulsar el botón de bloques en la columna de *options*.

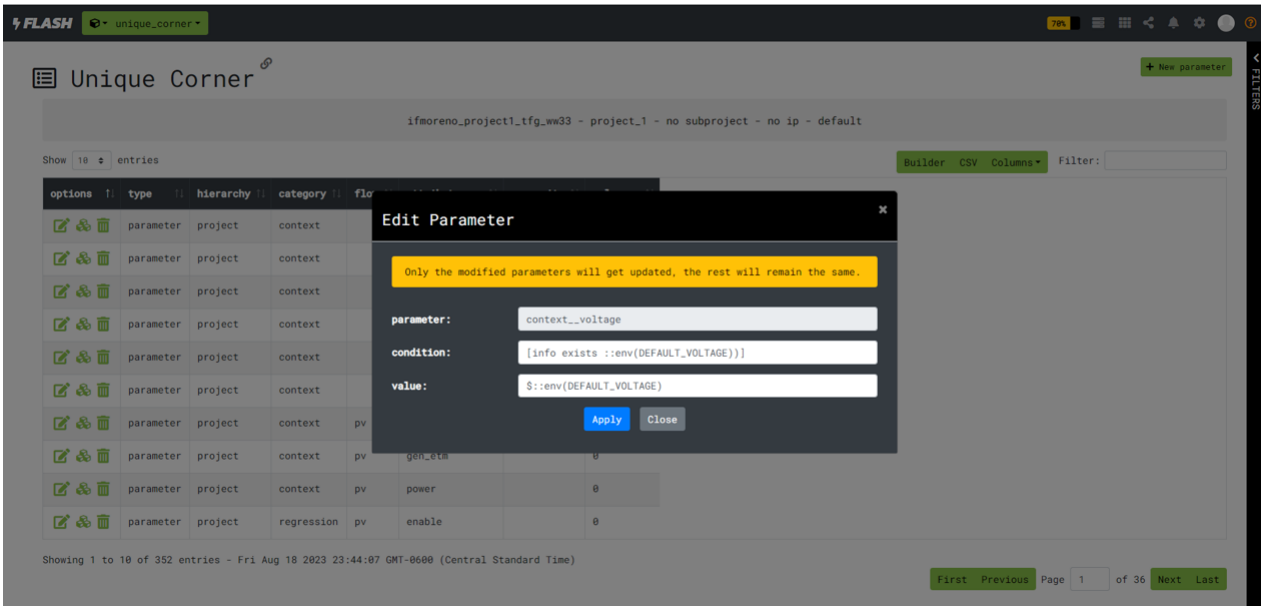


Figura 4.14: Interfaz gráfica: Editar/Crear condicional

Mediante la pestaña 4.14 el usuario puede crear condiciones para un parámetro directamente en sintaxis de TCL; estas se crean en pares condición-resultado, la cantidad de condiciones es transparente para el usuario, ya que es el compilador el encargado de darle el formato final. En caso de que el contexto requiera la habilitación de un nuevo parámetro este se puede hacer con el botón de la esquina superior en *Unique Corner* que aparece como *New Parameter*, que al pulsarlo despliega la siguiente pestaña, donde aparece un menú desplegable de categorías y flujos, para luego establecer el nombre del nuevo parámetro y su valor por defecto.

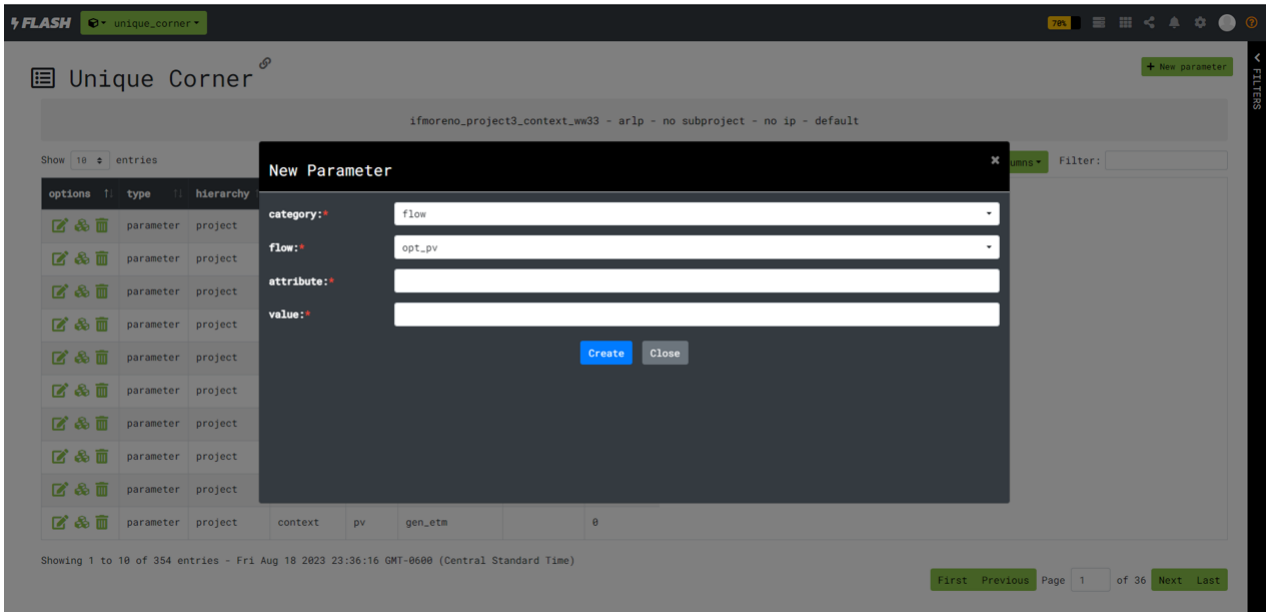


Figura 4.15: Interfaz gráfica: Nuevo parámetro

Capítulo 5

Análisis de resultados

En este capítulo se analizan los resultados obtenidos a partir del algoritmo de experimentación 4.4 y de verificación 4.5, así como la distribución de los datos, porcentajes de optimización, y el impacto de los distintos factores que afectan la solución.

5.0.1. Optimización

El algoritmo de experimentación descrito en la Figura 4.4 se realizó para tres proyectos, de los cuales el primero cuenta con 309 parámetros, el segundo 365, y el tercero 312, en cuatro iteraciones, para un total de 3708 experimentos. El objetivo principal de este experimento era encontrar uno o más parámetros, comunes entre proyectos que al ser utilizados como referencia para la optimización, redujeran al menos un cinco porciento las líneas de TCL compiladas, lo cual dio como resultado el histograma de la Figura 5.5.

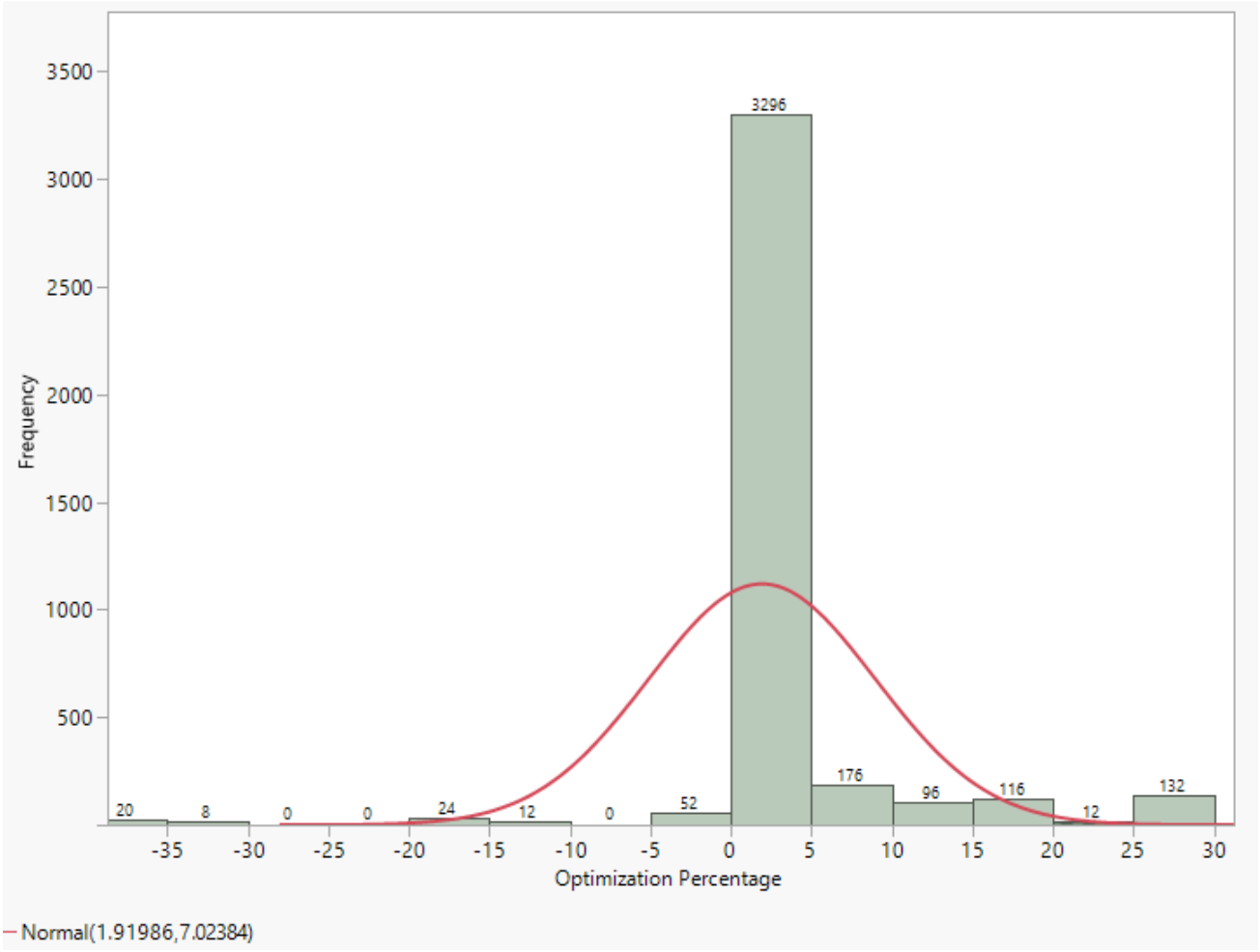


Figura 5.1: Histograma - Porcentaje de optimización general

La Figura 5.5 contiene los datos de todos los experimentos realizados, las cuatro iteraciones para los tres proyectos. Para estos experimentos se tomó la opción de parámetro de referencia del algoritmo de compilación de la Figura 4.3, cada parámetro por proyecto brinda una combinación distinta para el TCL, la cual puede tener más, menos, o igual cantidad de líneas que el caso base; por lo que queremos encontrar los parámetros que al ser usados como referencia optimizan la configuración.
A partir del histograma 5.5 se tiene que del total de experimentos, en el 2.941 % empeora, en el 97.059 % iguala u optimiza, y que en el 13.489 % se optimiza en un 5 % o más, los cuales se distribuyen como se muestra en la Figura 5.2.

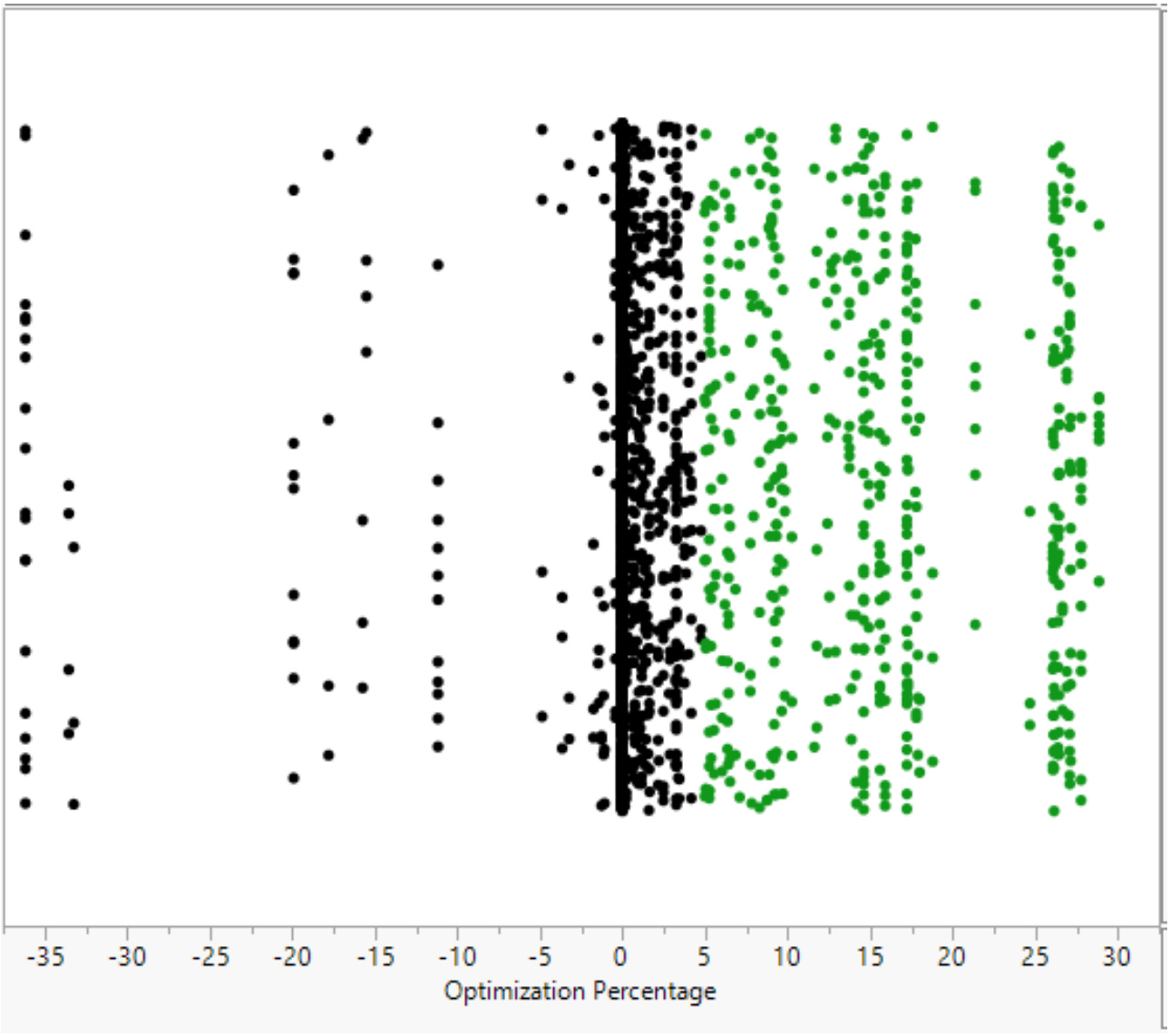


Figura 5.2: Diagrama de dispersión - Porcentaje de optimización general

En el diagrama de dispersión 5.2 se distinguen de color verde los parámetros que al utilizarse como referencia consiguen una optimización igual o mayor al 5 %, de igual manera se observa que que la gran mayoría de los parámetros de referencia mantienen igual el TCL de salida, siendo este el 71.805 % de los casos.
Los proyectos difieren en sus parámetros, estos pueden existir en solo un proyecto, en dos, o en los tres analizados; como lo que buscamos son parámetros comunes en los tres proyectos que se puedan usar como referencia para optimización, basados en la Figura 5.2 se deseaba filtrar los parámetros comunes entre proyectos, para lo que se planteó una gráfica de dispersión en la que se distinguiera de un color los parámetros comunes entre proyectos, y de otro los que solo se encontraban en uno o en dos, lo que dio como resultado la Figura 5.3

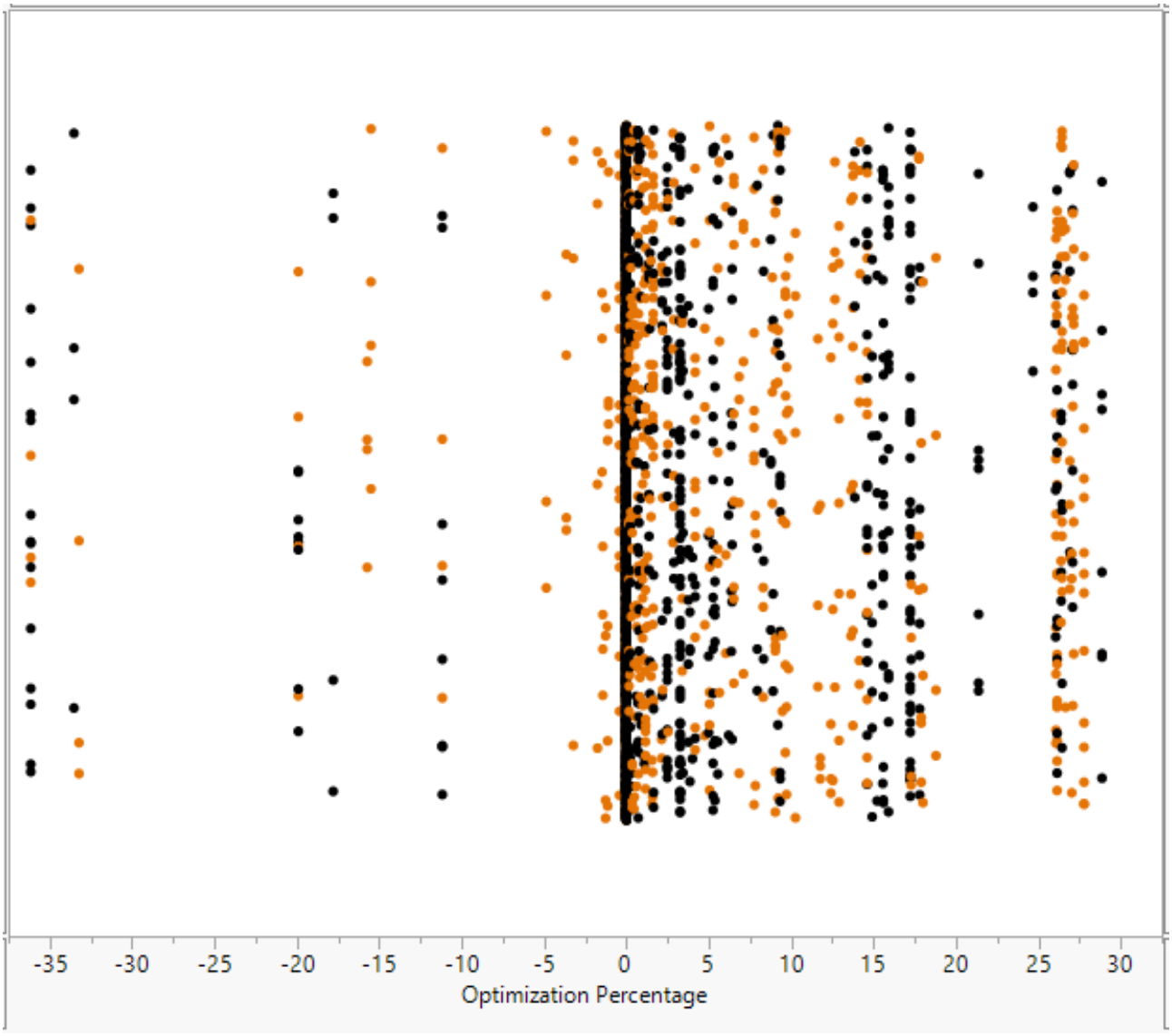


Figura 5.3: Diagrama de dispersión - Parámetros comunes

En la gráfica de dispersión 5.3 se tienen de color naranja aquellos parámetros comunes entre los tres proyectos, estos significan el 38.64 % de los parámetros totales, mediante el algoritmo de verificación se revisaron tanto los archivos de configuración provenientes de parámetros comunes como los que no. Como se muestra en la gráfica 5.3 hay múltiples parámetros que al utilizarse como referencia en el algoritmo de compilación, cumplen con el objetivo general del proyecto el cual es obtener una optimización de al menos un 5 % ya que cualquier parámetro naranja del gráfico de dispersión 5.3 que se encuentre sobre la línea de 5 o hacia la derecha, cumple con este propósito.

5.0.2. Factores y niveles

El análisis de factores y niveles está hecho con el objetivo de determinar qué componentes del sistema impactan en mayor o menor mediada el resultado final de la compilación. Montgomery en [27] describe que en muchos experimentos interviene el estudio de los efectos de dos o más factores. En general, los diseños factoriales son los más eficientes para este tipo de experimentos. En el diseño factorial se entiende que en cada ensayo o réplica completa del experimento se investigan todas las combinaciones posibles de los niveles de los factores. Por ejemplo, si el factor A tiene a niveles y el factor B tiene b niveles, cada réplica contiene todas las ab combinaciones de los niveles.

Los factores que posee el algoritmo de compilación como entrada son el proyecto y el corner de referencia, también se requiere verificar la existencia o no de cambios a la salida del sistema dependiendo del número de iteraciones en las que este se ejecute; esto sumado a los niveles que posee cada factor, definen al experimento como multifactorial [27].Como lo indica la Figura 5.3 hay parámetros que no se comparten entre proyectos, y aunque los archivos de configuración TCL fueron verificados, para las pruebas de normalidad nos centraremos en aquellos experimentos que tengan parámetros comunes, ya que estos son los que serán recomendados como referencia para la compilación, lo que genera la Tabla 5.1 de factores y niveles.

Factor	**Nivel**
Proyecto	Proyecto_1, Proyecto_2, Proyecto_3
Parámetro común	Parámetro_1, Parámetro_2, Parámetro_3 ... Parámetro_127
Iteración	Iteración_1, Iteración_2, Iteración_3, Iteración_4

Tabla 5.1: Factores y Niveles

Al ser tres factores, con tres niveles de proyecto, 127 parámetros en común en cuatro iteraciones, se cuenta con 1524 experimentos, que están contenidos dentro de los 3708 ya compilados y verificados. Como primer paso para determinar si los factores son significativos, se realizaron pruebas de normalidad, en el histograma 5.4 se observa la distribución del porcentaje de optimización para los factores y niveles 5.1.

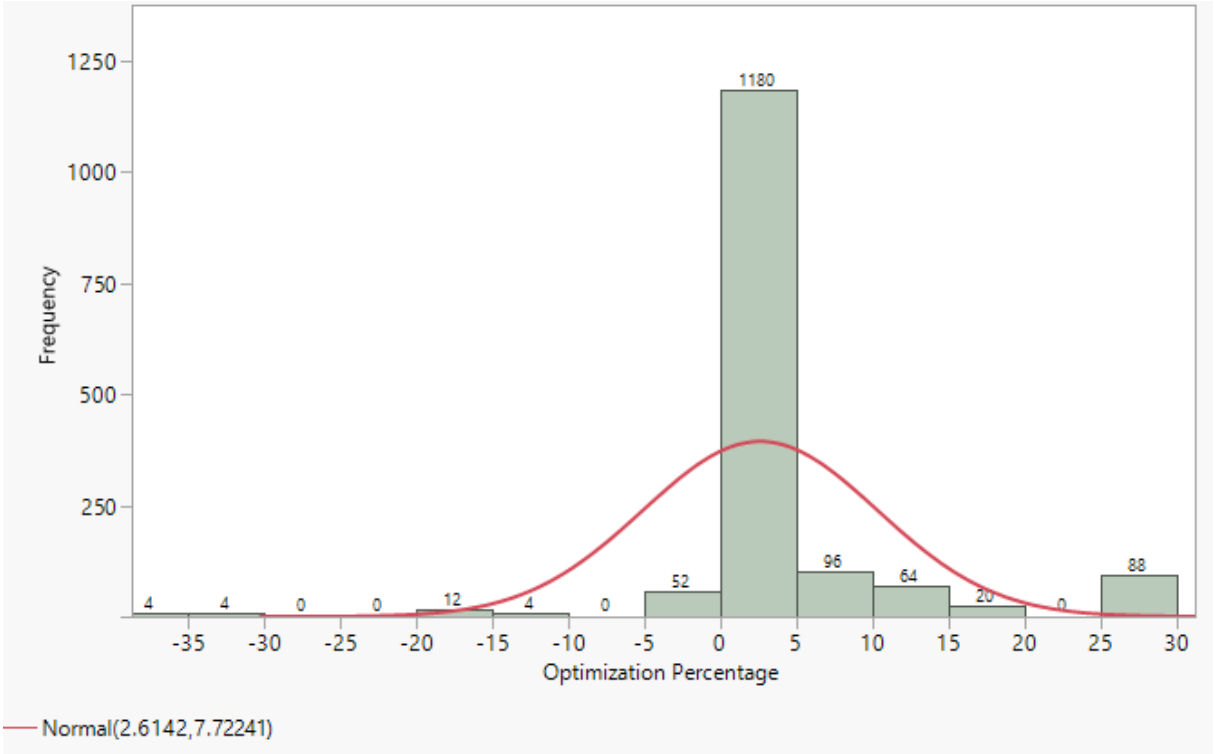


Figura 5.4: Porcentajes de optimización de parámetros comunes

A partir del histograma 5.4 se tiene que del total de experimentos, en el 4.986 % de los casos empeora, en el 77.427 % iguala u optimiza, y que en el 17.585 % se optimiza en un 5 % o más. Una vez calculados los porcentajes de optimización de parámetros comunes, se realiza el análisis de normalidad para lo que se eligió la prueba de Shapiro-Wilk, que como se menciona en [28] es una prueba de hipótesis que puede ayudar a determinar si una muestra ha sido extraída de una distribución normal. La hipótesis nula para la prueba establece que la muestra proviene de una distribución normal, y la hipótesis alternativa establece que no lo es. Otra opción para determinar la normalidad es la prueba de Kolmogorov-Smirnov, como se explica en [28], esta ayuda a determinar si una muestra es extraída de una distribución teórica particular; tiene la hipótesis nula de que la muestra proviene de la distribución y la hipótesis alternativa de que no es así, pero esta fue descartada ya que como se menciona en el mismo libro, para realizar la prueba se requiere que no existan datos con valores iguales, lo que sucede en nuestro caso cuando dos parámetros optimizan de la misma manera.

Para la prueba de Shapiro-Wilk la hipótesis nula (H_0) establece que los datos provienen de una distribución normalmente distribuida [28], si el valor p es menor que el nivel alfa elegido, en nuestro caso del 5 %, entonces se rechaza la hipótesis nula y hay evidencia de que los datos probados no siguen una distribución normal. El algoritomo de Shapiro-Wilk se ejecutó mediante el software JPM, lo cual dio como resultado la Figura 5.5.

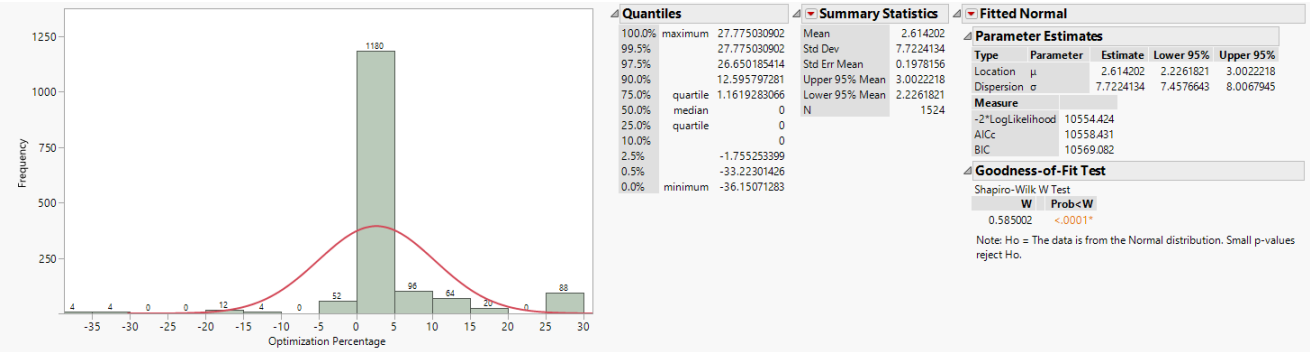


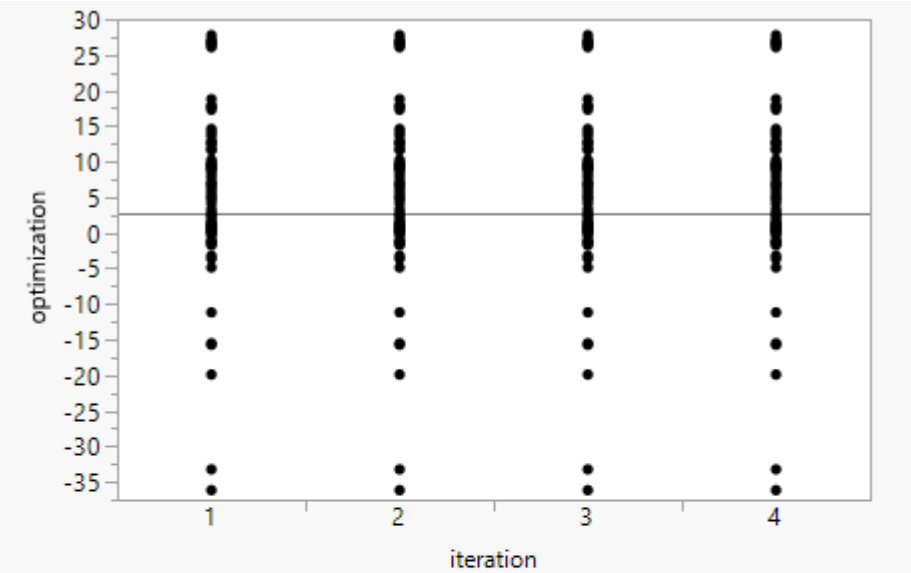
Figura 5.5: Análisis de normalidad

En la Figura 5.5 se muestra un resumen de la distribución estadística de los datos, siendo el apartado de *Goodness-of-Fit Test* el que presenta los resultado de la prueba de Shapiro-Wilk, donde se representa de color naranja el valor p de 0.001, siendo este mucho menor al de significancia 0.005, por lo tanto se rechaza la hipótesis nula y se concluye que los datos no siguen una distribución normal.

Los datos al no seguir una distribución normal, se procedió a analizar como un procedimiento estadístico no paramétrico, con el método de Kruskal-Wallis, que como se describe en [29] este se utiliza para comparar más de dos muestras que son independientes o no están relacionadas. El equivalente paramétrico para esta prueba es el análisis de varianza (ANOVA).

Se realizaron las pruebas de Kruskal-Wallis en JMP cada uno de los factores de la tabla 5.1 para un valor de significia del 0.05, la hipótesis nula establece que todos los factores afectan de igual manera, por lo tanto para que un factor de la tabla 5.1 sea significativo, su valor debe ser menor al de significancia.

El primer factor al que se le realizaron pruebas de Kruskal-Wallis, fue al número de iteración, donde cada iteración implica compilar todos los proyectos con los parámetros comunes como referencia. En la Figura 5.6 se muestra el impacto de cada una de las tres iteraciones realizadas sobre la optimización de lineas de TCL:



Wilcoxon / Kruskal-Wallis Tests (Rank Sums)

Level	Count	Score Sum	Expected Score	Score Mean	(Mean-Mean0)/Std0
1	381	290513	290513	762.500	0.000
2	381	290513	290513	762.500	0.000
3	381	290513	290513	762.500	0.000
4	381	290513	290513	762.500	0.000

1-Way Test, ChiSquare Approximation

ChiSquare	DF	Prob>ChiSq
0.0000	3	1.0000

Figura 5.6: Kruskal-Wallis para el factor iteración

En la Figura 5.6 se observa en la gráfica superior que los valores de optimización, siguen el mismo comportamiento a través de las iteraciones, lo que se comprueba con los valores de la tabla *1-Way Test, ChiSquare Approximation* donde el valor p en la casilla $Prob > Chisq$ es de 1, lo que además de no rechazar la hipótesis nula, implica que este del todo no afecta el resultado del algoritmo de optimización, por lo tanto no importa cuantas veces se ejecute el algoritmo, dadas las mismas entradas, se asegura la misma salida.

El segundo factor analizado fue el de parámetro de referencia, que posee 127 niveles, en el algoritmo de compilación cada parámetro de referencia dependiendo de los valores que tenga disponibles por proyecto va a definir los corners de referencia aparte del default. Al aplicar las pruebas de Kruskal-Wallis al factor de parámetro común, se tiene como resultado la Figura 5.7.

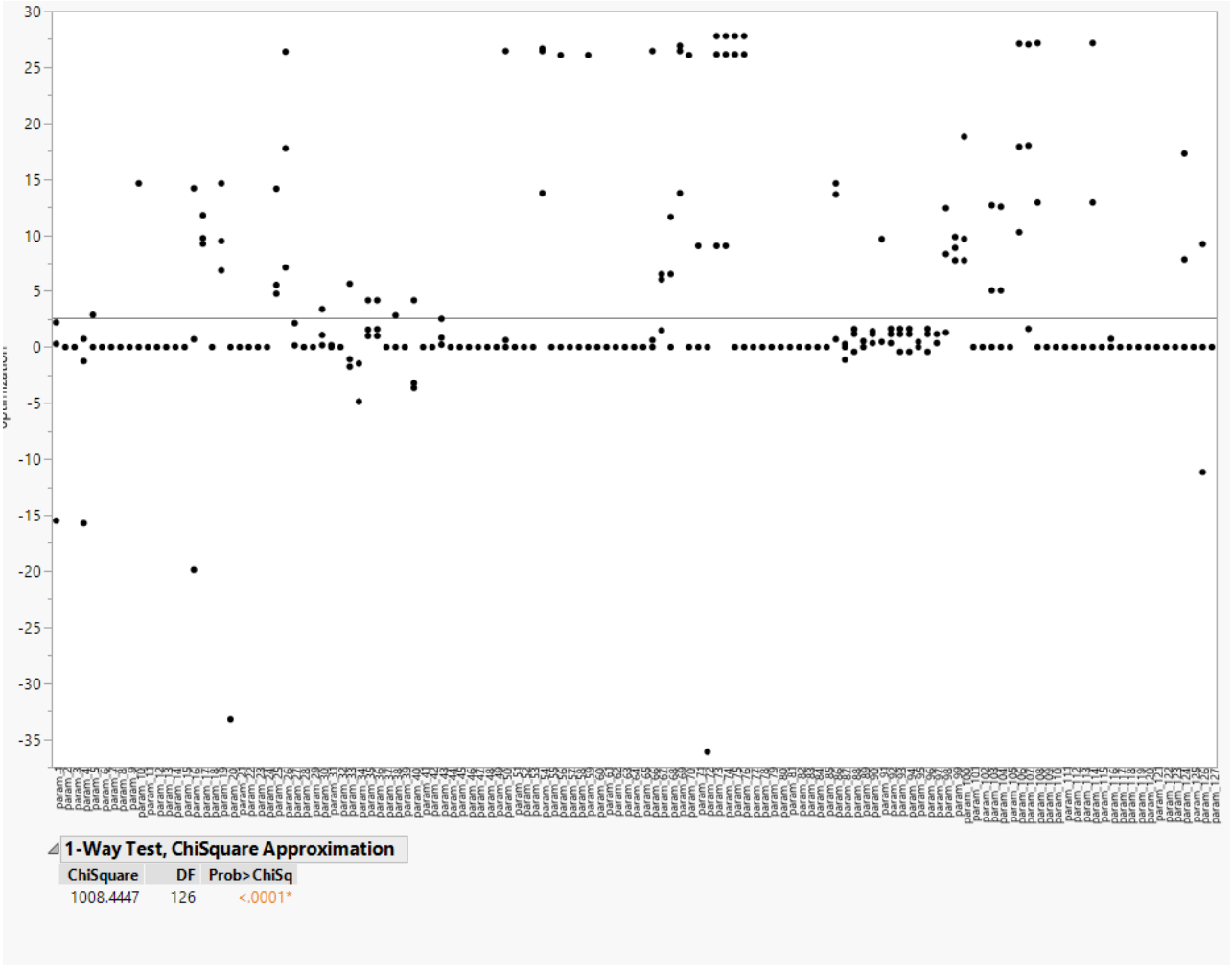
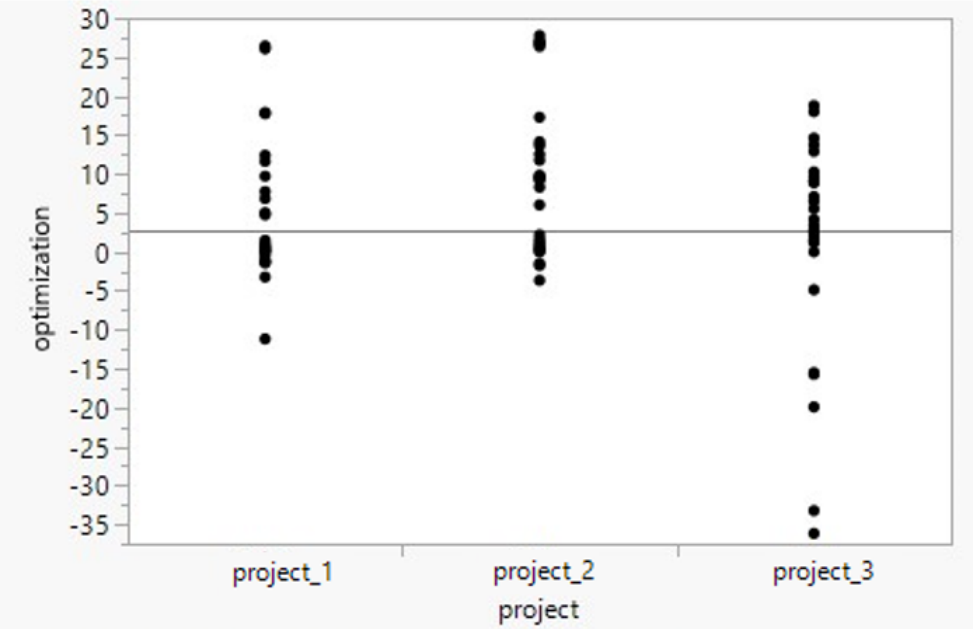


Figura 5.7: Kruskal-Wallis para el factor parámetro

En la Figura 5.7 se compone de dos elementos, el primero es una gráfica que modela el impacto del parámetro de referencia sobre la optimización de lineas de TCL, a diferencia de la gráfica en el Kruskal-Wallis de iteración 5.6, los puntos poseen distribuciones distintas para cada parámetro, lo que implica que el factor afecta la optimización, lo que se confirma con los valores de la tabla inferior 5.7, donde el valor p en $Prob > Chisq$ es 0.001, esto rechaza la hipótesis nula y al ser un número mucho menor a 0.05, el parámetro de referencia es un factor de alta significancia.

El tercer y último factor al que se le realizaron pruebas de Kruskal-Wallis fue al de proyecto, para este factor contábamos con tres proyectos disponibles, se considera un proyecto como disponible si se contaba con la base de datos completa, el compilador utilizado, el ambiente de variables respectivo, el kit de desarrollo correspondiente y una partición para pruebas; al aplicar las pruebas de Kruskal-Wallis se obtuvieron los resultados de la Figura 5.8.



Wilcoxon / Kruskal-Wallis Tests (Rank Sums)

Level	Count	Score Sum	Expected Score	Score Mean	(Mean-Mean0)/Std0
project_1	508	375206	387350	738.594	-1.735
project_2	508	407846	387350	802.846	2.928
project_3	508	378998	387350	746.059	-1.193

1-Way Test, ChiSquare Approximation

ChiSquare	DF	Prob>ChiSq
8.6728	2	0.0131*

Figura 5.8: Kruskal-Wallis para el factor proyecto

Al identificar el valor p calculado $Prob > Chisq$, se observa que este caso el valor p 0.0131 se encuentra por debajo del valor de significancia estándar 0.05, lo cual al igual que en el caso anterior nos indica que el proyecto tiene un peso importante sobre la optimización, pero no tan alto como el parámetro que se use como referencia; esto es algo esperado, ya que cada proyecto cuenta con distintos valores de parámetros y corners habilitados.

A partir del histograma 5.4 se determinó que, en el 17.585 % de los experimentos que utilizaban parámetros comunes entre todos los corners se optimizan las líneas del TCL compilado en un 5 % o más, sumado a esto los resultados de Kruskal-Wallis donde los factores que tienen impacto sobre la optimización son el proyecto y el parámetro de referencia, se graficó en la Figura 5.9 el número de grupos que realiza el algoritmo de compilación en base a el proyecto y parámetro de referencia, con el fin de encontrar patrones en común entre parámetros que brinden una alta mejora.

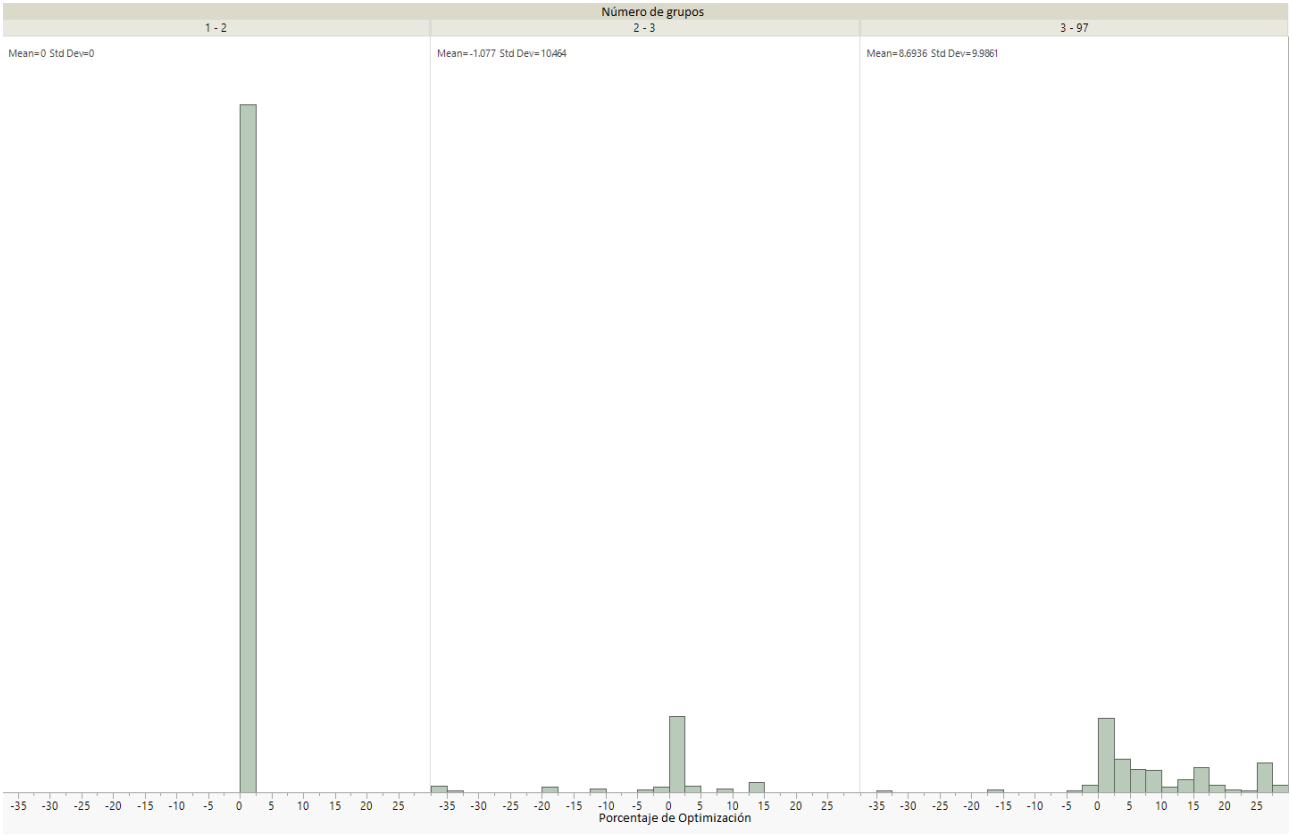


Figura 5.9: Porcentajes de optimización por número de grupos

En síntesis, en la Figura 5.4 se concluyó que el algoritmo de compilación optimizó en un 17.585 % de los experimentos realizados con tres proyectos, con 127 parámetros en común, en cuatro ocasiones, luego al determinar mediante el métotodo de Shapiro-Wilk que el resultado de dicho experimento no seguía una distribución normal, se procedió a analizar como un procedimiento no paramétrico mediante pruebas de Shapiro-Wilk, las cuales indicaron que los factores de peso para la compilación son el parámetro de referencia y el proyecto; estos dos factores interactúan con el algoritmo de compilación como se mostró en descripción detallada de la solución 4.3, para crear los grupos de corners, los que se están implícitamente correlacionados con la optimización tal y como se muestra en la Figura 5.9.

De la gráfica de optimización por número de grupos 5.9 se puede concluir que cuando solo existen entre uno y dos valores disponibles del parámetro de referencia en el proyecto, la optimización tiende a cero, y a medida de que aumentan los grupos, también lo hace la optimización, pero no de manera lineal. El comportamiento no lineal se debe a que no todos los grupos tienen la misma cantidad de corners, a la hora de compilar el TCL no es lo mismo cuatro grupos con el 25 % de corners, que los mismos cuatro grupos con el 85 % de los corners en el primer grupo, y los otros tres restantes con 5 % cada uno, sumado a esto, los parámetros difieren en las líneas de TCL que requieren para definirse, ya que la cantidad de condicionales de un parámetro puede variar entre un proyecto y otro.

Ya que el factor de parámetro de referencia es por lejos el más significativo a la hora de optimizar, lo que debemos hacer para asegurar una optimización igual o mayor al 5 % es definir una lista de parámetros de referencia comunes entre proyectos que hayan optimizado constantemente a través de los experimentos, los que se resumen en la tabla 5.2.

Parámetro referencia	**Proyecto**	**Optimización [%]**	**Optimización promedio [%]**
Interconexión	proyecto_1 proyecto_2 proyecto_3	9.72 11.77 9.22	10.23
Proceso	proyecto_1 proyecto_2 proyecto_3	6.84 9.47 14.61	10.31
Voltaje	proyecto_1 proyecto_2 proyecto_3	17.75 26.38 7.1038	17.08
Biblioteca para celdas de memoria estática	proyecto_1 proyecto_2 proyecto_3	26.44 26.65 13.75	22.28
Condición de operación para análisis de setup	proyecto_1 proyecto_2 proyecto_3	26.14 27.78 9.04	20.98
Condición de operación para análisis de hold	proyecto_1 proyecto_2 proyecto_3	26.14 27.78 9.04	20.98
Dominios de voltaje del escenario	proyecto_1 proyecto_2 proyecto_3	17.89 27.10 10.26	18.41

Tabla 5.2: Lista de parámetros comunes con optimización superior al 5 %

En la tabla 5.2 encontramos 7 parámetros que al usarse como referencia en el algoritmo de compilación de cualquiera de los proyectos, superan el 5 % de optimización buscado, con mejoras de hasta el 25 % en ocho casos. Los parámetros que están en la tabla 5.2 dependen directa o indirectamente en los factores que modelan el escenario, los cuales son proceso, voltaje, temperatura, interconexión y escenario de temporización; esto genera que al usar parámetros de referencia que están directamente relacionados a esos factores, son los que generan mayores similitudes, y hacen que modelen más grupos y compartan más valores en común que el resto, lo que se ve reflejado en el porcentaje de optimización.

A partir de los datos presentados en la 5.2, se identificó una heurística para distinguir los parámetros que ofrecen las optimizaciones más destacadas. Con base en esta heurística,

se desarrolló un algoritmo que compila que el proyecto seleccionado con base en cada uno de los parámetros de la tabla y elige la configuración óptima, basado en las lineas totales de configuración como el resultado final.

La función interpreta los parámetros de referencia a partir de una lista que se encuentra en el mismo orden que la tabla: primero interconexión, segundo proceso, tercero voltaje y así sucesivamente con cada uno de ellos; el parámetro se toma en cuenta para la solución final solo si mejora la optimización en comparación a los parámetros anteriores de la lista. Lo que genera los siguientes resultados:

- Parámetro de referencia para el proyecto_1: Biblioteca para celdas de memoria estática, para una optimización del 26.44 %.
- Parámetro de referencia para el proyecto_2: condición de operación para análisis de setup, para una optimización del 27.78 %.
- Parámetro de referencia para el proyecto_3: proceso, para una optimización del 14.61 %.

Capítulo 6

Conclusiones

En primer lugar, el compilador implementado es más flexible a la hora de definir condicionales para los parámetros, ya que acepta condicionales directamente en TCL, en lugar de la alternativa anterior, en la que cada nuevo condicional debía ser codificado en Perl por parte del usuario.

Por otro lado, la implementación del compilador a partir de una base de datos organizada por capas, facilita el diseño de nuevos proyectos, ya que puede tomar como referencia configuraciones globales, de fábrica y de proceso para definir nuevos proyectos, en lugar de un archivo en Microsoft Excel.

Así mismo, entre el control de versiones y la flexibilidad de configuración se estandariza el compilador, ya que con la metodología anterior podían existir versiones locales modificadas del compilador, que si no se compartían correctamente se podían perder entre proyecto y proyecto.

Además, el porcentaje de optimización de líneas a la salida del compilador, se ve reflejado en el espacio de memoria del archivo en formato TCL de manera proporcional.

A su vez, la interfaz gráfica en conjunto con la opción de importar y exportar archivos en formato csv, brinda las ventajas de arquitectura, búsqueda y visualización de una interfaz web, como las funcionalidades de edición de las aplicaciones comerciales que manejan archivos en formato csv.

Por último, el control de versiones en conjunto con la interfaz gráfica, permiten compartir y generar áreas de trabajo para la configuración de corners y parámetros de un contexto de manera ágil y segura.

Capítulo 7

Recomendaciones y Trabajo Futuro

En primer lugar, la interfaz gráfica al ser una herramienta que se proyecta para ser utilizada como plataforma de desarrollo de escenarios PVT en futuros proyectos, se recomienda darle mantenimiento y actualizaciones para cubrir las necesidades de los clientes a medida que se requieran nuevas funcionalidades o mejoras a las ya desarrolladas.

Así mismo, algunas mejoras que pueden agilizar el proceso de diseño son tablas editables, filtros dinámicos u optimización de los menús relacionados a corners y parámetros que permitan la visualización a través de las capas de configuración.

A su vez, las mejoras relacionadas al control de versiones van de la mano con la interfaz gráfica, ya que para este proyecto se implementó la arquitectura de los repositorios a nivel de Git, así como el pull y push de configuraciones de un contexto, como trabajo futuro se puede realizar un menú que muestre el historial de cambios ya sea en un área de trabajo o en el repositorio principal, para así aprovechar o recuperar diseños previos, desde la interfaz.

Como complemento de la interfaz gráfica, se sugiere la habilitación de la línea de comandos de Unix como interfaz con el usuario, con el fin de brindar una alternativa de funciones rápidas de visualización y edición que no dependan de la solución web.

Finalmente, el compilador podría ser agregado como parte del flujo de verificación, de modo que pueda leer la base de datos en tiempo de ejecución. De esta manera, se automatizaría el proceso de generación y consumo de la configuración de los corners.

Bibliografía

[1] Eric Matthes. *Python crash course: A hands-on, project-based introduction to programming*. no starch press, 2019.

[2] Ann Mutschler. *Process Corner Explosion*. 2018. URL: https://semiengineering.com/process-corner-explosion/ (visitado 26-04-2023).

[3] S. Churiwala y S. Garg. *Principles of VLSI RTL Design: A Practical Guide*. SpringerLink : Bücher. Springer New York, 2011. ISBN: 9781441992963. URL: https://books.google.co.cr/books?id=gyDS6xiqnakC.

[4] Ali Dasdan e Ivan Hom. «Handling inverted temperature dependence in static timing analysis». En: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 11.2 (2006), págs. 306-324.

[5] Weste Neil H E. y David Money Harris. *CMOS VLSI Design: A circuits and systems perspective*. Pearson India, 2015.

[6] Ying-Wen Bai y Fu-En Tsai. «Design and Implementation of a Table-based GUI for MP3 Players». En: *2006 IEEE International Symposium on Consumer Electronics*. 2006, págs. 1-6. DOI: 10.1109/ISCE.2006.1689433.

[7] K. Bindu Madhavi et al. «Derating Analysis for Reliability of Components». En: *Department of ECE, HITAM, Hyderabad* 2 (2015), pág. 39.

[8] C.C. Huang, F. Wong y D. Kim. «Different electrical measuring techniques of package and interconnect parasitics for high speed VLSI devices». En: *Proceedings of NORTHCON '94*. 1994, págs. 137-143. DOI: 10.1109/NORTHC.1994.643327.

[9] «IEEE Standard for Integrated Circuit (IC) Open Library Architecture (OLA)». En: *IEEE STD 1481-2009* (2010), págs. 1-658.

[10] Kiran Kumar y Neha Thakur. *VLSI Design*. Malla Reddy College, 2019.

[11] Norman Einspruch. *VLSI handbook*. Academic Press, 2012.

[12] A.B. Kahng et al. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer International Publishing, 2022. ISBN: 9783030964153. URL: https://books.google.com/books?id=Ey51EAAAQBAJ.

[13] Christopher Hogstrom. *Why learn tcl as a Digital Circuit Designer*. 2018. URL: https://grittyengineer.com/why-learn-tcl-tool-command-language-as-a-digital-circuit-designer/ (visitado 26-05-2023).

[14] John K Ousterhout y Ken Jones. «Tcl and the Tk Toolkit Second Edition». En: ().

[15] URL: https://wiki.tcl-lang.org/page/Preprocessor.

[16] Jia Huang y Hamid Shahnasser. «A preprocessor Tcl script generator for NS-2 communication network simulation». En: *2011 International Conference on Communications and Information Technology (ICCIT)*. 2011, págs. 184-187. DOI: 10.1109/ICCITECHNOL.2011.5762676.

[17] Nazatul Nurlisa Zolkifli, Amir Ngah y Aziz Deraman. «Version Control System: A Review». En: *Procedia Computer Science* 135 (2018), págs. 408-415.

[18] Steffan Otte. «Version Control Systems». En: ().

[19] Jon Loeliger y Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.", 2012.

[20] Andrew Dillon. «User Interface Design». En: *Encyclopedia of Cognitive Science*. 2006, págs. 453-458.

[21] Soren Lauesen. *User interface design: a software engineering perspective*. Pearson Education, 2005.

[22] Christian Gram. «A Software Engineering View at User Interface Design». En: (mar. de 1995). DOI: 10.1007/978-0-387-34907-7_16.

[23] Parita Jain, Arun Sharma y Laxmi Ahuja. «The Impact of Agile Software Development Process on the Quality of Software Product». En: *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. 2018, págs. 812-815. DOI: 10.1109/ICRITO.2018.8748529.

[24] Bilal Gonen y Dipali Sawant. «Significance of Agile Software Development and SQA Powered by Automation». En: *2020 3rd International Conference on Information and Computer Technologies (ICICT)*. 2020, págs. 7-11. DOI: 10.1109/ICICT50521.2020.00009.

[25] Gopalkrishna Waja, Jill Shah y Pankti Nanavati. «AGILE SOFTWARE DEVELOPMENT». En: *International Journal of Engineering Applied Sciences and Technology* 5 (abr. de 2021). DOI: 10.33564/IJEAST.2021.v05i12.011.

[26] Loo Khong et al. «Software Development Life Cycle AGILE vs Traditional Approaches». En: feb. de 2012.

[27] Douglas C. Montgomery. *Diseno y anaélisis de experimentos: Douglas C. Montgomery*. Grupo Editorial Iberoameérica, 1991.

[28] S. Baldock. *Using R for Statistics*. Books for professionals by professionals. Apress, 2014. ISBN: 9781484201398. URL: https://books.google.com/books?id=kFcnCgAAQBAJ.

[29] G.W. Corder y D.I. Foreman. *Nonparametric Statistics: A Step-by-Step Approach*. Wiley, 2014. ISBN: 9781118840313. URL: https://books.google.com/books?id=hYVYAwAAQBAJ.