UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería Electrónica

Análisis temporal estático y optimización del flujo de trabajo en un diseño digital

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

AUTOR/A: Navas Morales, Juan Pablo

Tutor/a: Pérez Pascual, Mª Asunción

Cotutor/a externo: Salvador Edo, Rubén

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

ANÁLISIS TEMPORAL ESTÁTICO Y OPTIMIZACIÓN DEL FLUJO DE TRABAJO EN UN DISEÑO DIGITAL

Autor: Juan Pablo Navas Morales

Tutor: María Asunción Pérez Pascual

Tutor de Empresa: Rubén Salvador Edo

Trabajo Fin de Máster presentado en el Departamento de Ingeniería Electrónica de la Universitat Politècnica de València para la obtención del Título de Máster Universitario en Ingeniería de Sistemas Electrónicos

Curso 2023-24

Valencia, Septiembre de 2024

Resumen

El objetivo fundamental del presente trabajo es realizar un estudio exhaustivo sobre los conocimientos, técnicas y procesos relacionados con el análisis temporal estático (STA), de forma que un diseñador digital encargado de realizar dicha tarea, sea capaz de llevarla a cabo con éxito, pues no se trata de una rama de conocimiento generalmente impartida en los planes de estudio.

Para ello, se establecerán unas bases de conocimiento digital necesarias para la comprensión de ciertos términos y entornos empleados a lo largo de todo el trabajo. Posteriormente, se explicarán todos los chequeos temporales que las herramientas de STA llevan a cabo, así como la información relativa a esta disciplina incluida en las librerías de fabricantes. A continuación, se explican todas las reglas de diseño que se deben incluir para modelar y restringir adecuadamente el proyecto.

Por último, se analizarán casos reales de un proyecto realizado en Analog Devices, que permitan al lector ejemplificar los contenidos aprendidos a través de todas las secciones, así como entender como funciona las herramientas de análisis. De igual modo, se estudiará un ejemplo de automatización llevado a cabo en el proyecto para mejorar el flujo de diseño.

Resum

L'objectiu fonamental del present treball és realitzar un estudi exhaustiu sobre els coneixements, tècniques i processos relacionats amb l'anàlisi temporal estàtic (STA), de manera que un dissenyador digital encarregat de realitzar aquesta tasca sigui capaç de dur-la a terme, ja que no es tracta d'una branca del coneixement generalment impartida als plans d'estudi.

Per això, s'establiran unes bases de coneixement digital necessàries per a la comprensió de certs termes i entorns emprats al llarg de tot el treball. Posteriorment, s'explicaran tots els controls temporals que les eines de STA duen a terme, així com la informació relativa a aquesta disciplina inclosa a les biblioteques dels fabricants. A continuació, s'expliquen totes les regles de disseny que s'han d'incloure per modelar i restringir adequadament el projecte.

Finalment, s'analitzaran casos reals d'un projecte realitzat a Analog Devices, que permetran al lector exemplificar els continguts apresos a través de totes les seccions, així com entendre com funcionen les eines d'anàlisi. De la mateixa manera, s'estudiarà un exemple d'automatització realitzat en el projecte per a millorar el flux de disseny.

Abstract

The fundamental objective of this work is to conduct a thorough study of the knowledge, techniques, and processes related to Static Timing Analysis (STA), so that a digital designer responsible for performing this task is capable of doing it, as it is not a field of knowledge generally covered in curriculum plans.

To achieve this, a set of necessary digital knowledge bases will be established for understanding certain terms and environments used throughout the work. Subsequently, all the timing checks performed by STA tools will be explained, as well as the information related to this discipline included in manufacturer libraries. Next, all the design rules that should be included to model and

properly constrain the project will be explained.

Finally, real cases from a project carried out at Analog Devices will be analyzed, allowing the reader to exemplify the content learned through all sections, as well as to understand how the analysis tools work. Similarly, an example of automation implemented in the project to improve the design flow will be studied.

A mi familia, mi pareja y todas las personas que me han acompañado y apoyado durante estos años, tanto a los que están como a los que estuvieron.

Gracias.

Índice general

Índice de figuras

Listado de siglas empleadas

ASIC Application Specific Integrated Circuit.

AT Arrival Time.

CDC Clock Domain Crossing.

CTS Clock Tree Synthesis.

DRC Design Rule Check.

DUA Device Under Analysis.

FPGA Field Programmable Gate Array.

IPs Intellectual Properties.

LEC Logic Equivalent Check.

LUT Look Up Table.

LVS Layout Versus Schematic.

MMMC Multi-Mode Multi-Corner.

OTP One Time Programmable.

PnR Place and Route.

RAT Required Arrival Time.

RTL Register Transfer Level.

STA Static Timing Analysis.

Parte I

Memoria

Capítulo 1

Introducción y Objetivos

1.1. Introducción

El proceso de verificación de un circuito integrado de propósito específico (ASIC) es una tarea compleja y vital durante la fase de diseño, que permite asegurar el correcto funcionamiento del ASIC antes del proceso de fabricación. Una vez el diseño ha sido cerrado, se procede a crear las máscaras encargadas de transferir los patrones al sustrato de silicio, siendo este proceso uno de los más costosos en el desarrollo y fabricación de un ASIC. Por esta razón, los procesos de verificación se han convertido en la última década en uno de los puntos de mayor interés para las empresas del sector, aumentando los recursos destinados a estas disciplinas.

Algunas de las técnicas de verificación más comunes son la simulación funcional y de potencia, la verificación formal, el estudio de *Clock Domain Crossing* (CDC) con el que pueden detectarse problemas de metaestabilidad o el análisis de estabilidad de alimentación, sin embargo, el presente trabajo busca centrarse en explicar cómo realizar de forma exitosa un análisis temporal estático (*Static Timing Analysis*, STA).

1.1.1. Qué es el análisis temporal estático y sus beneficios

El análisis temporal estático STA es una técnica que se centra en verificar que todas las rutas de temporización en un diseño cumplan con las restricciones de tiempo establecidas. Además, ofrece información sobre las posibles violaciones que podrían producirse en el circuito. A pesar de que no se trata de la única forma de realizar estas comprobaciones, se ha convertido en la técnica más especializada y directa de hacerlo, pues aunque otras técnicas como las simulaciones funcionales pueden arrojar violaciones temporales, no están diseñadas para realizar un análisis temporal exhaustivo.

Una de las ventajas del STA en comparación con las simulaciones funcionales, es que no necesita de estímulos de entrada específicos para llevar a cabo el estudio, es decir, el STA no tiene en cuenta la funcionalidad del diseño. En cambio, esta técnica tan solo se centra en realizar el estudio temporal a nivel de transferencia de registros (RTL) diseñado y haciendo uso tanto de las reglas de diseño especificadas por el diseñador, como de los ficheros de librerías de fabricantes, de forma que se pueda verificar si el diseño es capaz de operar bajo ciertas condiciones específicas de voltaje, temperatura o velocidad.

Además, las herramientas de STA permiten al diseñador realizar unos estudios denominados Multi-Mode Multi-Corner (MMMC), lo cual agiliza enormemente el proceso de estudio al analizar, en una sola simulación, todas las posibles condiciones a las que puede estar expuesto el ASIC, ya que se trata de una metodología avanzada utilizada para verificar el rendimiento temporal de un circuito digital bajo diversas condiciones operativas y ambientales.

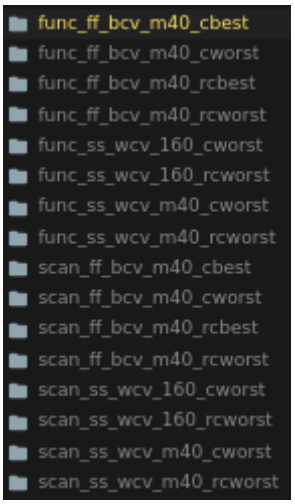


Figura 1.1: Resultados STA en modo MMMC

En la figura 1.1 se puede apreciar un árbol de archivos reportados por la herramienta de STA configurada en modo MMMC. A continuación, se detallan las posibles combinaciones que la herramienta ha llevado a cabo.

- La primera gran división elige modo de funcionamiento, escaneo (“scan”) es un modo de evaluación empleado para comprobar el correcto estado del RTL una vez ha sido fabricado y funcional (“func”) es el modo en el que se analiza el RTL que desempeña la propia funcionalidad del diseño digital.
- El siguiente hace distinción entre el modo rápido (“ff”) y lento (“ss”). El primero de ellos generalmente incluye temperaturas más bajas y voltajes más altos, además de características de fabricación que favorecen los transistores más rápidos. Este modo evalúa casos de tiempos de propagación más cortos, como las violaciones de mantenimiento (*hold*) que más tarde serán analizadas. El modo de funcionamiento lento tiene en cuenta los casos opuestos al rápido, y se centra en analizar violaciones de configuración (*setup*).
- El resto de parámetros hacen alusión a las condiciones que se acaban de comentar: “bcv” y “wcv” son respectivamente los mejores y peores casos de voltaje; “m40” y “160” hacen referencia a la temperatura de operación, mientras que “cbest”, “cworst”, “rcbest” y “rcworst” nos permiten operar en configuraciones más o menos favorables en cuanto a la carga y resistencia de las interconexiones.

Gracias a la metodología MMMC que integran las herramientas de STA es posible hacer este estudio tan exhaustivo y centrado en el análisis temporal que lo diferencia del resto de opciones de verificación. Además, algunas de las características que ofrece son [1]:

- Robustez y Fiabilidad: Asegura que el circuito funcione correctamente bajo todas las condiciones especificadas.
- Cobertura Integral: Proporciona una verificación completa del rendimiento temporal del diseño.
- Optimización Multiescenario: Permite optimizar el diseño para múltiples escenarios, reduciendo el riesgo de fallos de temporización.

1.1.2. Proceso de diseño digital de un ASIC

Como sucede con cualquier disciplina compleja, realizar un proceso de estas características requiere especialización y división de funciones, para garantizar de esta forma que se tienen en cuenta diferentes enfoques durante la realización del proyecto, lo cual fomenta la resolución de errores y la mejora del diseño.

Por lo tanto, el proceso de diseño de un ASIC puede dividirse en las siguientes fases:

- Diseño RTL: en esta etapa se realiza el diseño digital empleando un lenguaje de descripción hardware como el VHDL, aunque actualmente los más empleados son Verilog y System Verilog.
- Síntesis lógica: el siguiente paso es sintetizar una lista de nodos (*netlist*) a partir del RTL diseñado en la etapa anterior. Las herramientas encargadas de esta tarea realizan optimizaciones del diseño para mejorar las prestaciones en cuanto a área, consumo o velocidad. Existen diferentes configuraciones específicas que permiten ajustar dichas optimizaciones de cara a las necesidades del diseño.
- Planificación del diseño (*Floorplan*): una vez se dispone de la lista de nodos (*netlist*) generada por el sintetizador, hay que asignar una ubicación física específica a cada uno de los bloques del diseño, así como áreas restringidas (E/S o las áreas del núcleo). El objetivo es minimizar las distancias de interconexión.
- Emplazado (*Placement*): a continuación, se asigna el emplazamiento final teniendo en cuenta factores como la densidad de interconexión y la minimización de los retardos de señal.
- Rutado (*Routing*): en esta etapa, se emplean algoritmos que determinan las mejores rutas de interconexión, de forma que sea posible cumplir con los requisitos temporales establecidos para el diseño.
- Procesos de verificación posteriores al diseño: como se verá en posteriores secciones, este no es el único punto en el que se aplican procesos de verificación, pero si que es el más completo al haber pasado por todas las etapas de diseño.
- Comprobación de las reglas de diseño (DRC): aseguran que se cumplen las especificaciones permitidas por la tecnología de fabricación.
- Comparación de los diseños físico y lógico (LVS): compara la red de conexiones del esquemático con la red extraída del diseño físico para verificar que ambas son equivalentes.
- Máscaras y fabricación: el último paso consiste en generar las máscaras de litografía que se utilizarán en el proceso de fabricación.

1.1.3. Fases del proceso de STA y puntos de aplicación

El estudio de STA se divide en tres grandes fases que se detallarán en las proximas secciones, siendo estás:

- Uso de librerías proporcionadas por los fabricantes de las celdas empleadas en el diseño del ASIC. Aquí se incluye toda la información temporal que posteriormente la herramienta va a verificar.
- Definición de reglas de diseño o restricciones que especifican ciertas características importantes del diseño que la herramienta debe tener en cuenta a la hora de realizar el análisis.
- Proceso de depuración y reescritura de restricciones tras la obtención de resultados proporcionados por las herramientas de análisis.

Las herramientas de análisis temporal estático, tales como Tempus de Cadence [2], permiten al diseñador realizar estudios en diferentes partes del proceso de diseño. Esta versatilidad se debe a que las herramientas son capaces de realizar el estudio con diferentes ficheros de información, creando la posibilidad de realizar análisis desde las primeras etapas del proceso de diseño. En la figura 1.2 se puede apreciar un ejemplo de la configuración que la herramienta de análisis temporal de cadence (Tempus) ofrece a sus diseñadores para elegir los archivos de entrada que se usarán en el estudio. Esto ofrece versatilidad a la hora de realizar los análisis de tempranas fases del diseño, de forma que no sea necesario contar con toda la información del rutado y emplazado para el análisis temporal.

```
################
# ANALYSIS TYPE
################

# syn              : Analyze syn netlist + syn generated SPEF     + tcons constraints
# pnr              : Analyze pnr netlist + pnr generated SPEF     + pnr generated constraints
# signoff (default): Analyze pnr netlist + extract generated SPEF + tcons constraints
STA_ANALYSIS_TYPE = signoff
```

Figura 1.2: Configuración Tempus

1.1.4. Otros procesos de verificación

Como se comentó en secciones anteriores, la verificación del diseño digital de un ASIC se trata de un proceso complejo que se divide en diferentes análisis para cercionarse del correcto funcionamiento del mismo desde diferentes ámbitos de estudio. El principal aliado del STA es la verificación funcional [3], pues anteriormente este era el único método empleado para comprobar que las especificaciones temporales estaban siendo cumplidas.

En la actualidad, los proyectos de diseño digital reutilizan bloques de propiedad intelectual (IPs) de proyectos anteriores con los que han tenido buenos resultados, pero esto suele entrar en conflicto con las nuevas metodologías de verificación. Los proveedores de IPs suelen incluir una guía de usuario que cuenta con recomendaciones para comprobar las especificaciones temporales del diseño, es por ejemplo en esta sección, donde las más recientes IPs albergan las reglas de

diseño que se deben incluir en el análisis de STA. Sin embargo, debido a la reutilización de IPs en los procesos de diseño, aún se dan casos en los que diferentes metodologías de verficación deben convivir, ya que los fabricantes prepararon sus IPs para que fuesen analizadas tan solo desde una verificación funcional.

Además de estas herramientas de verificación, otras comúnmente utilizadas son:

- Verificación formal [4]: emplea ciertos métodos matemáticos para probar algunas propiedades del diseño que no han podido ser completamente cubiertas con la verificación funcional.
- Verificación de potencia [5]: esta metodología evalúa el consumo energético del ASIC bajo ciertas condiciones de funcionamiento.
- Verificación de lógicas equivalentes (LEC) [6]: esta técnica crucial se encarga de comprobar que dos versiones de diseño son completamente iguales a nivel funcional. Este proceso es de gran utilidad dado que a lo largo del diseño del RTL se llevan a cabo optimizaciones que pueden llegar a cambiar el funcionamiento inicial del diseño.
- Emulación hardware: recientemente se ha comenzado a hacer uso de las FPGA para comprobar que el RTL diseñado para el ASIC desempeña correctamente su funcionalidad.

Todos estos métodos conviven y se complementan a la hora de verificar que se cumplen todas las especificaciones demandadas por el cliente antes de ser fabricado. Como se comentó al inicio de este trabajo, el coste de tener que repetir las máscaras de fotolitografía supondría numerosas pérdidas a la empresa diseñadora, por lo que los esfuerzos de verificación durante su etapa de diseño están más que justificados.

1.2. Objetivos

Los principales objetivos que busca cubrir este trabajo son:

- Dominar el análisis temporal estático de circuitos electrónicos, comprendiendo profundamente sus principios y aplicaciones. Identificar y aplicar las mejores prácticas para realizarlo correctamente, con el fin de optimizar el rendimiento y la eficiencia de los circuitos. Además, evaluar y comunicar efectivamente los beneficios y ventajas de su correcta implementación, contribuyendo así al avance y la innovación en el campo de la electrónica.
- Explicar cómo un diseñador electrónico digital puede llevar a cabo un análisis de STA, desde el estudio de los conceptos relativos a esta disciplina, estableciendo las restricciones necesarias al diseño y analizando los informes ofrecidos por las herramientas con el objetivo de solventar las violaciones temporales.
- Adquirir un dominio experto en el funcionamiento de las herramientas de STA, comprendiendo a fondo cómo realizan los estudios y análisis. Identificar y entender detalladamente la información necesaria para su correcta implementación.
- Proponer una mejora en el flujo de trabajo de un diseñador encargado del estudio de STA.

Estos conocimientos permitirán a un diseñador electrónico digital encargarse de todas las tareas relacionadas con el estudio de STA a lo largo del proceso de diseño de un ASIC, los cuales aportan gran valor a un grupo de trabajo. Además, estos conocimientos se imparten de manera superficial en los planes de estudio actuales, por lo que este trabajo busca cubrir un área de conocimiento muy valorada por las empresas de microelectrónica.

1.3. Estructura de la memoria

El presente trabajo fin de máster consta de cinco secciones de contenido teórico-práctico además de la introducción y objetivos que ya han sido expuestos.

En primera instancia se abordarán ciertos conceptos de análisis temporal estático que sentarán las bases para las posteriores secciones, tanto a nivel de conceptos teóricos como de cierta nomenclatura que se empleará a lo largo de todo el documento. Se estudiarán conceptos de lógica secuencial, tipos de caminos datos y el cálculo de *slack*.

En la siguiente sección relativa al análisis y cálculo temporal se analizarán todos los chequeos temporales que las herramientas de STA llevan a cabo durante sus estudios del diseño, tales como los tiempos de configuración (*setup*), mantenimiento (*hold*) o la verificación de cierre de reloj (*clock gating check*). Además, se explicará la interacción que tiene la herramienta con las librerías otorgadas por los fabricantes/diseñadores externos.

A través de la cuarta sección de contenido se estudiarán todas las reglas de diseño (*constraints*) que el ingeniero de diseño digital debe conocer para poder implementar con éxito un buen análisis temporal, tanto para modelar ciertos aspectos externos/internos del proyecto, como para solventar las violaciones reportadas.

La penúltima sección del trabajo expone varios ejemplos de violaciones reportadas en un proyecto de diseño real, así como la forma en la que han sido solventadas. Además, en este apartado se explica con detalle cómo las herramientas presentan la información en los informes temporales.

En última instancia, se exponen varias optimizaciones al flujo de diseño llevadas a cabo en forma de programas escritos con python, que han permitido mejorar el proceso de diseño de un proyecto real llevado a cabo en Analog Devices.

Capítulo 2

Conceptos Análisis Temporal Estático

En esta sección se van a detallar algunos conceptos esenciales para comprender las explicaciones del trabajo, como los diferentes tipos de caminos que componen la lógica del diseño, puntos específicos sobre los que se realizan los análisis temporales o cómo calcular los caminos críticos del diseño.

2.1. Lógica combinacional y secuencial

En el diseño digital, la lógica combinacional y la lógica secuencial son dos conceptos fundamentales que permiten la creación y operación de circuitos electrónicos complejos. Cada tipo de lógica tiene características y funciones específicas, y su correcta aplicación es crucial para el desarrollo eficaz de sistemas digitales.

En la lógica combinacional el estado de las salidas depende únicamente del valor de las entradas actuales, es decir, no tiene en cuenta los estados anteriores. Dentro de este grupo se encuentran puertas lógicas como las AND, OR, NOT, XOR, etc. Por otro lado, la lógica secuencial sí que tiene en cuenta el estado anterior del sistema a la hora de definir la salida actual, por lo que se podría decir que tienen una noción de memoria. Elementos como los registros (*flip-flops*) o los biestables (*latches*) se encuentran en este grupo. La combinación de ambos tipos de lógica da lugar a los complejos diseños digitales que se emplean en la actualidad.

A nivel de funcionalidad, la lógica combinacional suele emplearse para calcular operaciones tales como sumas o comparaciones, mientras que la lógica secuencial se usa para almacenar los resultados de dichas operaciones en registros. Otro enfoque podría ser el de una máquina de estados, en el que la lógica secuencial se emplea para guardar el estado actual de la máquina, y se hace uso de la lógica combinacional como señales de control para moverse entre estados.

En la figura 2.1 puede verse un ejemplo de diseño digital básico que servirá como base para explicar, en posteriores secciones, la mayoría de chequeos temporales que el proceso de STA verifica durante su ejecución. Consta de dos registros, uno de inicio y otro de captura, entre los que se realiza alguna operación o cálculo mediante lógica combinacional. Ambos registros están controlados por una señal de reloj, que puede o no, ser la misma, y además cuentan con una señal de *reset* (por lo general asíncrona) que permite llevar la salida del registro a un valor por defecto. En algunos casos, también cuentan con una entrada de *set*, que crea la posibilidad de llevar la salida

del registro a un valor concreto, puede ser útil en máquinas de estados por ejemplo, aunque su uso se reserva para situaciones aisladas.

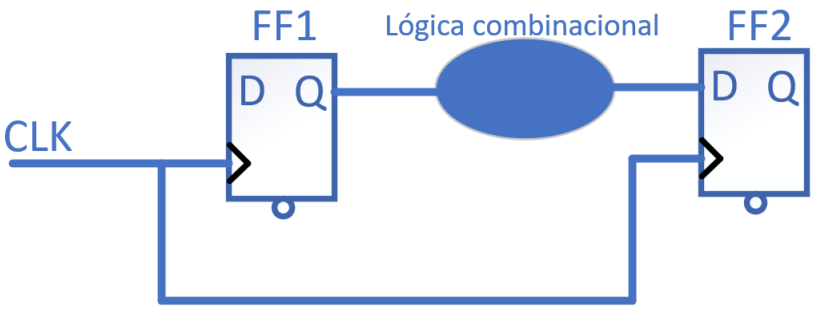


Figura 2.1: Circuito combinacional y secuencial

2.2. Tipos de caminos en el diseño digital

En esta sección se van a identificar los posibles caminos que pueden recorrer las señales dentro de un diseño digital. Las principales diferencias entre estos caminos son las características del mismo, el punto de inicio (*startpoint*) y de finalización (*endpoint*) e incluso la funcionalidad de la señal que propaga.

2.2.1. Ruta de datos

Dentro de un diseño secuencial se distinguen cuatro rutas de datos en función de los puntos de inicio y finalización de las mismas, que abarcarán todos los casos analizados posteriormente por la herramienta de STA.

- Pin de entrada a registro (*in2reg*).
- Registro a registro (*reg2reg*).
- Registro a pin de salida (*reg2out*).
- Pin de entrada a pin de salida (*in2out*).

En la ruta de los datos existen dos posibles puntos de inicio: un pin de entrada o el reloj (CLK) de un registro. En cambio, para los puntos de finalización las posibilidades son los pines de salida o la entrada de datos (D) de un registro. En la figura 2.2 pueden apreciarse las cuatro rutas de datos posibles, siendo la primera el *in2reg*, la segunda el *reg2reg*, la tercera el *reg2out* y la última el denominado *in2out*.

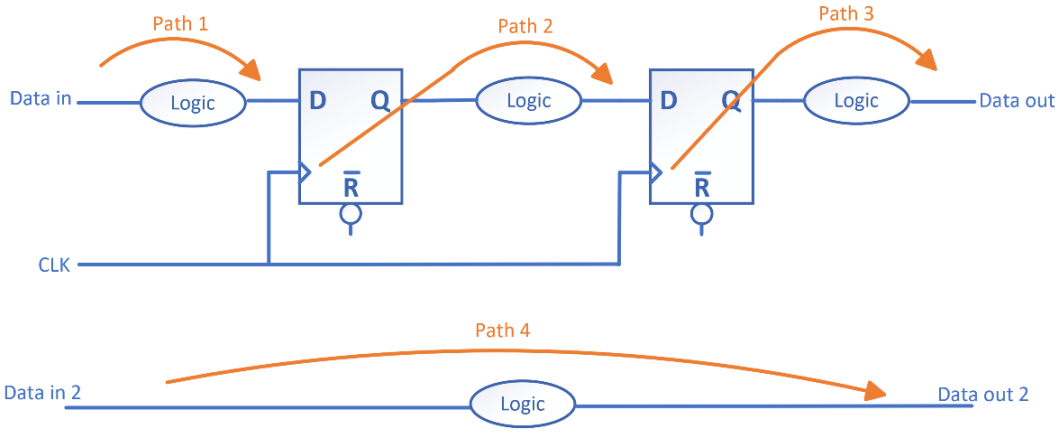


Figura 2.2: Ruta de datos

2.2.2. Rutas de reloj

Como se comentó anteriormente, una de las diferencias entre la lógica secuencial y la combinacional es la presencia de una noción de "memoria" en la primera de ellas. El reloj del sistema es el que realiza la temporización de los *data paths*, determinando en que momento se cambia el valor de salida del diseño.

Los estudios de STA siempre se realizan desde un punto de entrada a uno de finalización, tal y como se vio en las rutas de datos, pero además, estos deben tener siempre un reloj asociado. En el caso de los registros, el reloj será aquel que llegue a su pin CLK, mientras que en el caso de las entradas y salidas del diseño digital será necesario modelarlo a través de ciertas restricciones que se detallarán en posteriores secciones. De esta forma, el sistema se encuentra completamente acotado para poder realizar el análisis correctamente.

Este tipo de camino puede a su vez dividirse en dos más, aquel que llega al registro del punto de inicio que será llamado reloj de lanzamiento (*launch clock*), y el que llega al de destino denominado reloj de captura (*capture clock*), pudiendo ver estas diferencias en la figura 2.3.

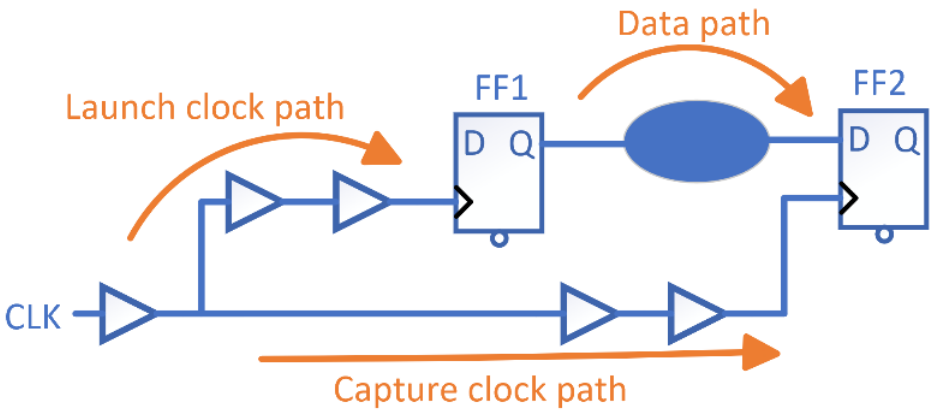


Figura 2.3: Rutas de reloj

Esta nomenclatura será importante ya que los informes arrojados por las herramientas de STA ofrecen información relativa a la ruta de lanzamiento y a la de captura.

2.2.3. Ruta de cierre de reloj

El camino descrito en este apartado no puede ser incluido ni dentro de las rutas de reloj ni en las rutas de datos, pues no cumple con ninguna de las especificaciones de puntos de inicio o finalización descritas en las secciones anteriores.

Se trata de un camino empleado para alterar las características de un reloj y su principal funcionalidad es la de “apagar” el reloj. Dado que la fuente de reloj siempre se encuentra en funcionamiento, para poder ahorrar consumo es necesario que la lógica del sistema deje de cambiar de un estado a otro, y para ello, evitar que les llegue la señal de reloj es una de las formas más comunes y sencillas. Como puede verse en la figura 2.4, estos caminos actúan como señales de control para alterar la propagación del reloj.

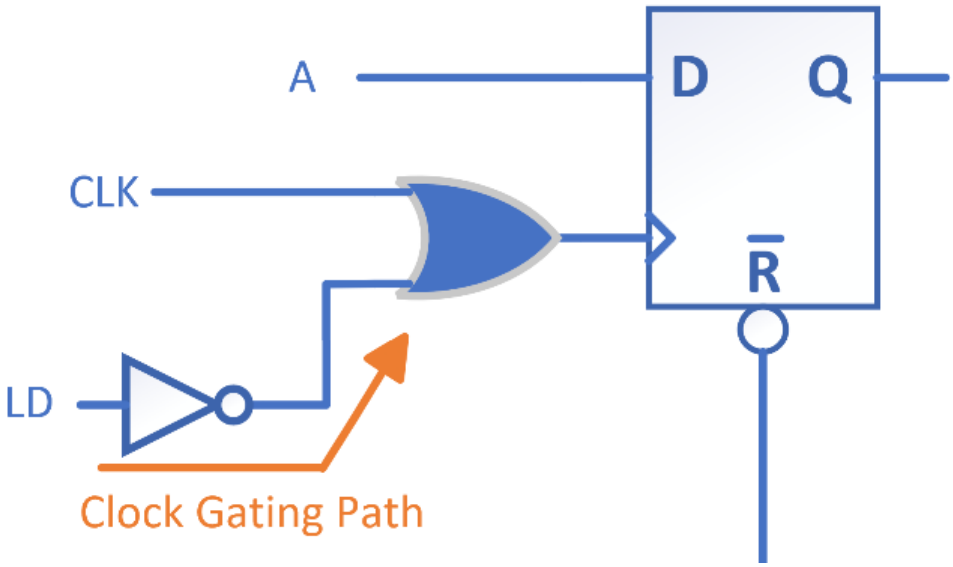


Figura 2.4: Ruta de cierre de reloj

En posteriores secciones se analizará cuales son los chequeos temporales aplicados en esta clase de caminos, pues tal y como se verá, no es posible cortar la propagación de reloj en cualquier momento.

2.2.4. Rutas asíncronas

Además de los caminos síncronos o las propias rutas de reloj, es necesario que exista cierta lógica que permita alterar el estado del diseño síncrono en cualquier momento, teniendo prioridad sobre éste. Las señales de *reset* y *set* que pueden llegar a los registros son dos ejemplos de señales asíncronas que tienen prioridad sobre el resto de caminos, y que pueden llegar en cualquier momento, cambiando el estado de salida del sistema.

En la siguiente figura 2.5 puede apreciarse un ejemplo de este tipo de rutas.

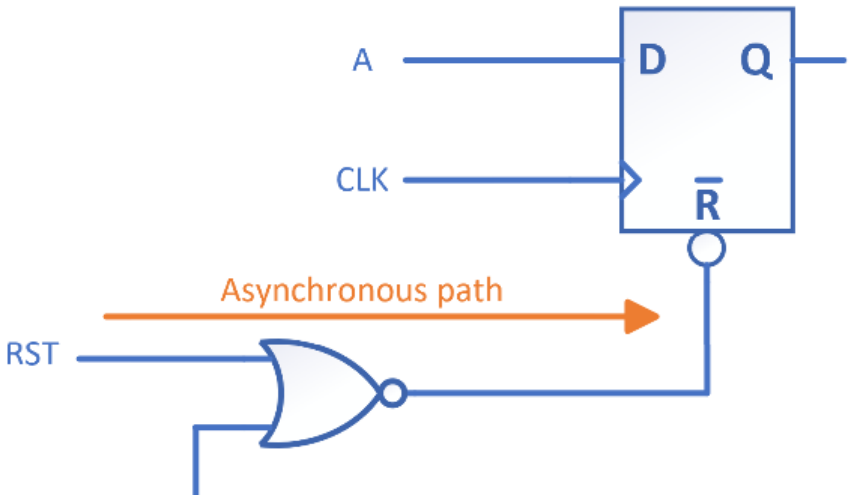


Figura 2.5: Rutas asíncronas

En el diseño digital existen otros caminos que también son denominados asíncronos, aunque en este caso es debido a la relación que existe entre el origen y el destino de la señal. En ciertos diseños con varios dominios de reloj, es muy probable que se de el caso de que una señal cruce de un dominio a otro, y si la relación entre ambos relojes es asíncrona, será necesario incluir en esta ruta un sincronizador. Dependiendo de la naturaleza de dicha señal, será necesario emplear una estructura de sincronización u otra, tales como doble o triple *flip-flop*, FIFOs o protocolos con *handshake* [7].

2.2.5. Otros tipos de rutas

A continuación, se van a explicar ciertas rutas que por definición si que se podrían ubicar en alguno de los ya mencionados, pero que debido a ciertas características diferenciales, es necesario dedicarles una mención especial.

- El primero de ellos se denomina “camino crítico” y se trata aquel camino del RTL más desfavorable desde el punto de vista de la temporización. En caso de que haya violaciones temporales, el camino crítico será aquel que posea mayor retardo de propagación, mientras que si no las hay será el que más cerca esté de cometer una violación.
- El segundo se trata de la ruta falsa (*false path*). Mientras que la mayoría de los caminos presentes en el diseño digital necesitan de una verificación temporal por parte de la herramienta de STA, hay ciertos casos en los que no es necesario realizar análisis alguno. Esto es debido a que física o funcionalmente ese camino no necesita dicho estudio temporal. Un sencillo ejemplo puede apreciarse en la figura 2.6, dado que ambos multiplexores tienen la misma entrada de selección, no es necesario estudiar el camino temporal de la entrada “in0” del “mux0” a la ´´in1” del “mux1”, pues en ningún caso se podría dar dicha combinación.

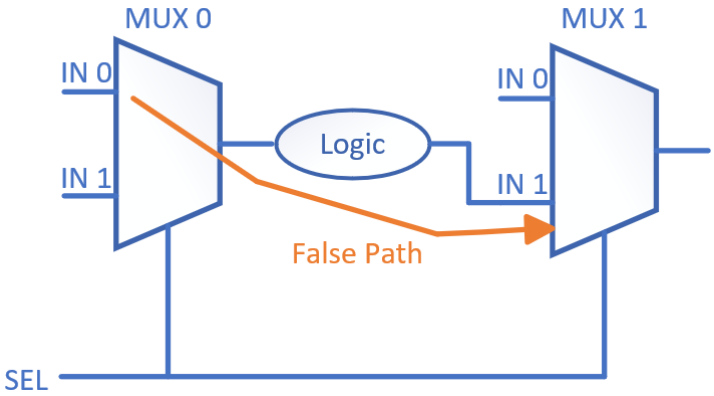


Figura 2.6: Rutas falsas

- Las rutas multiciclo (*multicycle path*) son aquellas que debido a la funcionalidad del diseño necesitan más de un ciclo de reloj para propagar cierta señal desde el punto de inicio hasta el de finalización. En caso de no definirlo, se produciría una violación temporal pues, por defecto, la herramienta realiza los chequeos temporales como si cada ruta de datos necesitase un solo ciclo para propagarse. En posteriores secciones se explicará con mayor detalle dicha regla de diseño y el análisis de estas rutas multiciclo.

2.3. Cálculo de *slack*

Antes de explicar el concepto de *slack*, es necesario tener en cuenta dos cálculos previos: el tiempo de llegada (*Arrival Time*, AT) y el tiempo requerido de llegada (*Required Arrival Time*, RAT). El AT es el momento en el que la señal llega al punto de finalización, teniendo en cuenta los retardos de la ruta del reloj de lanzamiento y de la ruta de datos. Por otro lado, el RAT es el instante en el que el dato debería llegar al punto de finalización teniendo en cuenta los márgenes de seguridad implicados en cada cálculo en específico, por ejemplo, los márgenes de mantenimiento, configuración o recuperación (*recovery*) que serán estudiados posteriormente.

El *slack* se trata de la diferencia temporal entre el RAT y el AT. Un *slack* positivo significa que el dato ha llegado antes de lo esperado, teniendo cierto margen de seguridad con respecto al valor necesario. En cambio, un valor de *slack* negativo significa que no se están cumpliendo los chequeos temporales, y por lo tanto, se reportará como violación.

En la figura 2.7 puede apreciarse visualmente un ejemplo del cálculo de *slack* que resultaría en una violación temporal debido a que el tiempo de llegada es mayor el requerido.

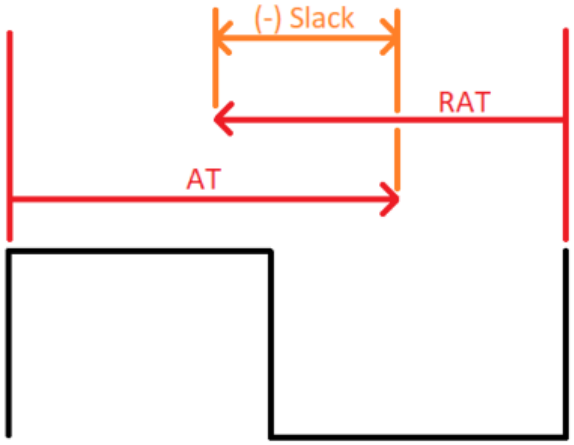


Figura 2.7: Slack negativo

Sin embargo, en la figura 2.8 se expone el caso contrario, en el que el dato llega antes de lo necesario y existe cierto margen de seguridad.

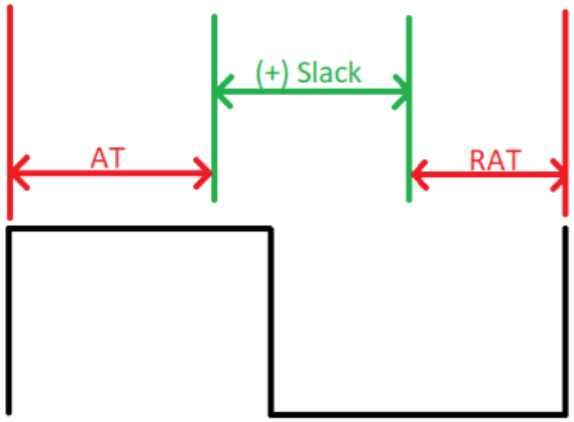


Figura 2.8: Slack positivo

Capítulo 3

Análisis y cálculo temporal

3.1. Chequeos temporales

En esta sección se van a detallar los principales chequeos temporales que ejercitan las herramientas de STA durante el análisis de un diseño digital.

3.1.1. Tiempo de configuración

El chequeo de configuración (*setup check*) se realiza entre los pines de reloj y datos de un registro. En este análisis se comprueba que el dato llegue al registro con suficiente antelación con respecto al flanco activo de reloj, es decir, que si se está capturando un dato con el flanco positivo del reloj, el dato debe llegar como mínimo un tiempo de configuración antes que dicho flanco.

Matemáticamente es posible calcular el chequeo de configuración como:

$$T_{\text{launch}} + T_{\text{ck2q}} + T_{\text{data path}} < T_{\text{capture}} + T_{\text{cycle}} - T_{\text{setup}} \tag{3.1}$$

De la ecuación anterior tan solo dos parámetros pueden ser modificados por el diseñador para solventar una violación de configuración: el tiempo de la ruta de datos, que puede ser reducido, o el periodo de reloj, que puede aumentarse, lo que implica utilizar un reloj con una frecuencia menor. Empleando las reglas de diseño concretas también es posible indicarle a la herramienta ciertos requerimientos, para que trate de optimizar la lógica de dicho camino.

En la figura 3.1 puede apreciarse un ejemplo gráfico del análisis de configuración entre dos registros. Cabe destacar que este chequeo se realiza en el registro de captura de la ruta de datos en estudio.

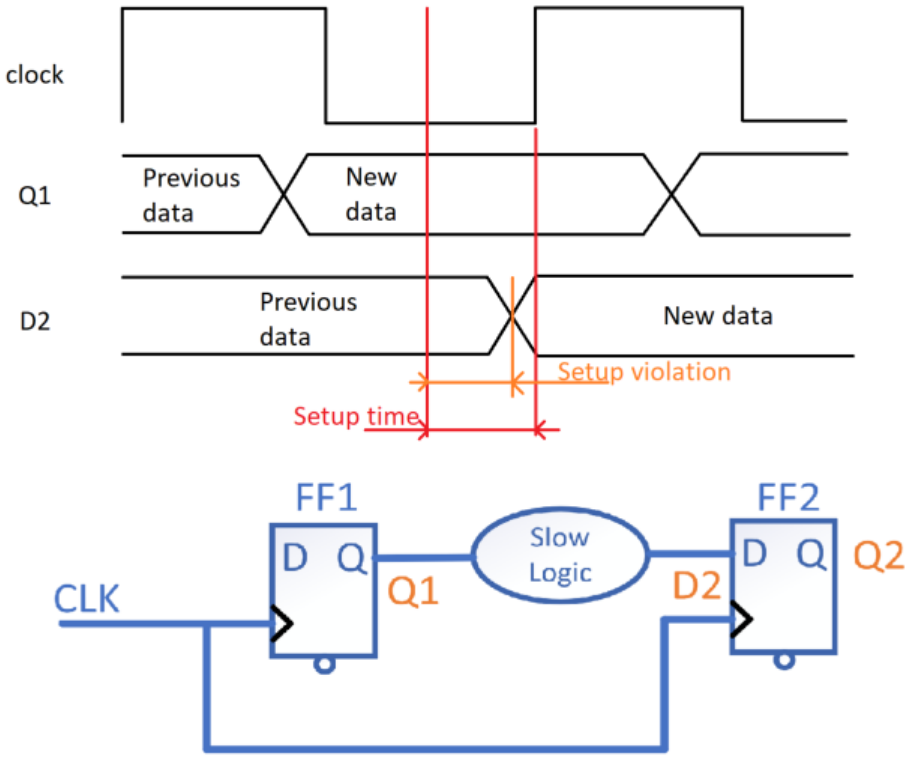


Figura 3.1: Verificación tiempo de configuración (*setup*)

Analizando la imagen anterior pueden sacarse varias conclusiones respecto a este chequeo. La primera de ellas se trata de la comprensión del propio tiempo de configuración, quedando muy bien reflejado que el dato no puede cambiar (en D2) durante un tiempo previo al flanco de captura en el segundo registro (FF2). También es posible identificar las soluciones anteriormente aportadas que podrían solventar la violación: conseguir una ruta de datos más rápida (el dato llegaría antes a D2) o aumentar el periodo del reloj (el *setup time* se desplazaría a la derecha y el dato ya no causaría la violacion).

3.1.2. Tiempo de retención

Al igual que el chequeo de configuración, el de retención (*hold*) se realiza entre los pines de reloj y datos, sin embargo, el de retención es el tiempo mínimo durante el cual los datos no pueden cambiar después del flanco activo del reloj. Por lo tanto, los caminos de datos con una lógica combinacional muy rápida serán los más críticos de cara a este chequeo.

Matemáticamente, el estudio del tiempo de retención (*hold*) se calcula como:

$$T_{\text{launch}} + T_{\text{ck2q}} + T_{\text{data path}} > T_{\text{capture}} + T_{\text{hold}} \tag{3.2}$$

Como se aprecia en la ecuación anterior, no es posible ajustar la frecuencia del reloj para solventar violaciones de retención una vez el diseño ha sido realizado, por lo que es de vital importancia identificarlas durante las fases de diseño. Es posible incluir algunos *buffers* en la ruta de datos para

hacerlo más lento y cumplir con la especificación de retención, para lo cual se pueden emplear ciertas restricciones que afectan al tiempo mínimo de propagación en una ruta de datos, o bien insertar directamente los *buffers* necesarios desde la herramienta de emplazado y rutado (PnR).

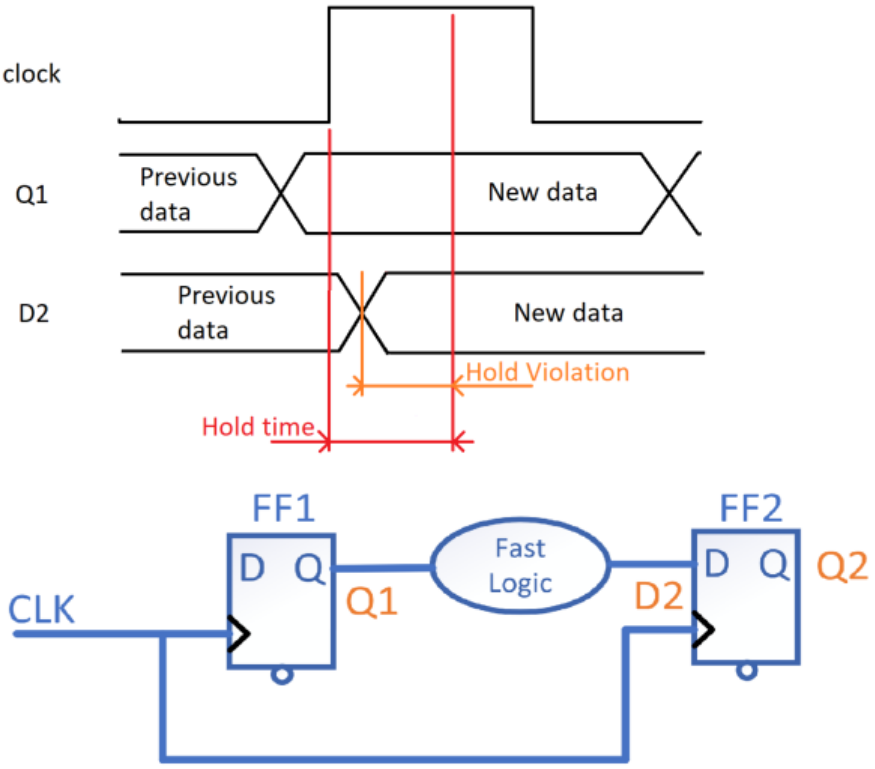


Figura 3.2: Verificación tiempo de retención (*hold*)

En la figura 3.2 se presenta el estudio de una violación de tiempo de retención, en el que el dato cambia demasiado rápido después del flanco activo de reloj, pudiendo provocar estados equivocados del diseño, errores de cálculo o aumentar el consumo de circuito.

Una diferencia con respecto al estudio del tiempo de configuración sería el flanco activo en el que se analiza cada uno. Para un mismo ciclo, el tiempo retención se chequearía con respecto al primer flanco activo, y el de configuración se realizaría previo al flanco del siguiente ciclo de reloj. Además, podría decirse que el chequeo del tiempo de retención se aplica al registro de lanzamiento.

En la figura 3.3 se muestran las formas de onda llegando al FF1 (*launch clock*) y al FF2 (*capture clock*), con ambos chequeos aplicados.

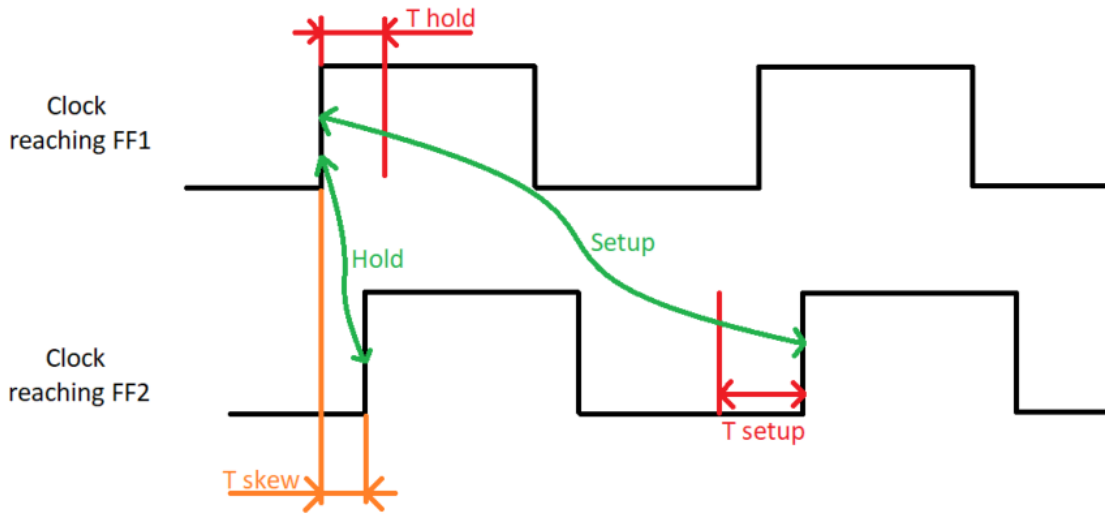


Figura 3.3: Verificación tiempos de configuración y retención

3.1.3. Tiempo de recuperación y eliminación

Los chequeos de recuperación (*recovery*) y de eliminación (*removal*) comparten ciertas características con los de configuración y retención, que posteriormente serán identificadas, pero su principal diferencia radica en los pines sobre los que se realiza dicho estudio. Los chequeos de recuperación y eliminación tan solo estudian la relación de pines asíncronos de un registro, como el *reset*, con respecto a la entrada de reloj. Concretamente, se estudia el momento en el que se produce la desactivación de la señal asíncrona, pues a partir de este momento el sistema debe recuperar el control en el próximo flanco activo de reloj.

Para su explicación, se estudiará un chequeo de recuperación y eliminación entre los pines de reloj y *reset*, concretamente entre el flanco activo de reloj (*posedge*) y la desactivación del *reset* (*negedge* pues será activo a nivel alto). Esta desactivación del reset no puede suceder en cualquier instante temporal, y es por ello que deben cumplirse dos condiciones: que se respete un tiempo mínimo de eliminación después del flanco activo de reloj (similar al de retención) y un tiempo de recuperación previo al próximo flanco (análogo al de configuración).

En la figura 3.4 pueden apreciarse ambos chequeos, y como la desactivación del *reset* debe producirse en algún punto intermedio entre ellos.

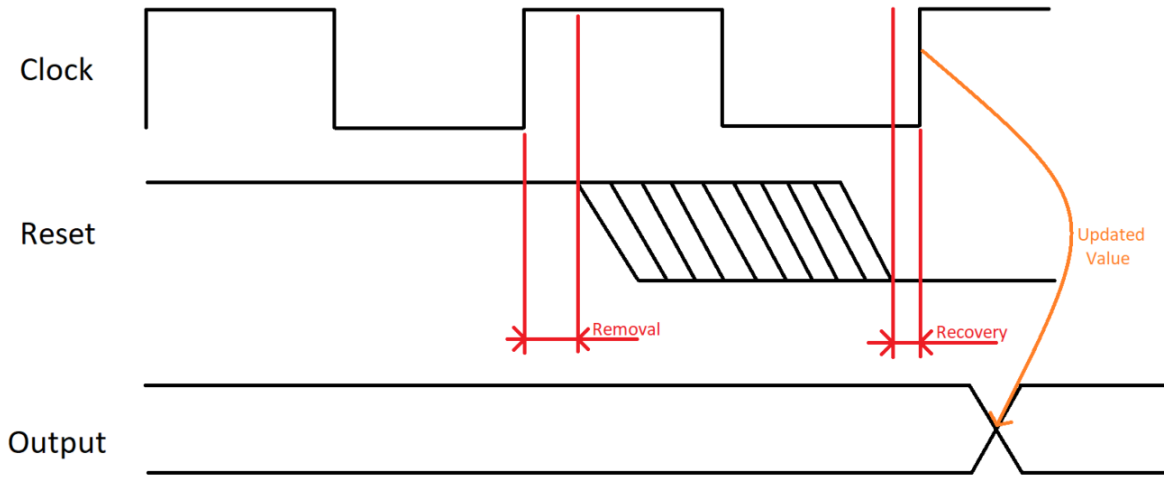


Figura 3.4: Verificación tiempos de recuperación y eliminación

3.1.4. Anchura mínima de pulso

El chequeo del ancho mínimo de pulso se realiza sobre los pines asíncronos y de reloj de las celdas secuenciales del diseño, por ejemplo los registros. A diferencia de los estudios anteriores, este análisis no se realiza entre varios pines de entrada, si no que tan solo se encarga de medir el ancho del pulso de entrada a cada pin.

Algunos de los aspectos a tener en cuenta de cara a prevenir dichas violaciones en base la latencia del árbol de reloj son:

- Emplear celdas lógicas más rápidas.
- Establecer valores agresivos para las transiciones de reloj.
- Verificar la estructura del reloj antes de la síntesis del árbol de reloj (CTS).
- En caso de multiplexar un reloj de alta frecuencia con uno de baja para test, asegurarse de que la latencia de los relojes de alta frecuencia no se ve afectada durante la síntesis.

En la sección de librerías 3.2 se detallará donde encontrar la información de estos chequeos para cada pin de entrada que lo requiera.

3.1.5. Verificación de cierre de reloj

Los caminos de reloj por lo general tan solo deben contener *buffers* o inversores añadidos por la herramienta para ajustar la propagación del reloj a lo largo del diseño, de forma que llegue a cada celda secuencial en el momento oportuno. Cuando en una ruta de reloj existe otro tipo de lógica, así como una puerta AND, la herramienta realizará la verificación de cierre de reloj (*clock gating check*).

Tal y como se comentó en la sección 2.2.3, es muy aconsejable incluir cierta lógica en los caminos de reloj que nos permitan controlar su propagación, pues si se evita que el reloj llegue a un bloque concreto del diseño que no se encuentra funcionalmente activo, se ahorrá consumo. Los cambios de estado a nivel de transistor requieren energía, por lo que evitar que un reloj cambie constantemente de valor, será una de las formas más eficaces de ahorro energético.

En la actualidad, las herramientas de síntesis están preparadas para realizar las adaptaciones necesarias para incluir celdas de *clock gating* allá donde lo considere oportuno, aunque se considera una buena práctica que los diseñadores las incluyan en sus diseños RTL.

Este chequeo se realiza entre la señal de control (*gating signal*) y el reloj que va a ser sometido a modificación (*clock signal*). Las figuras 3.5 y 3.6 pueden tomarse como referencia para las siguientes explicaciones.

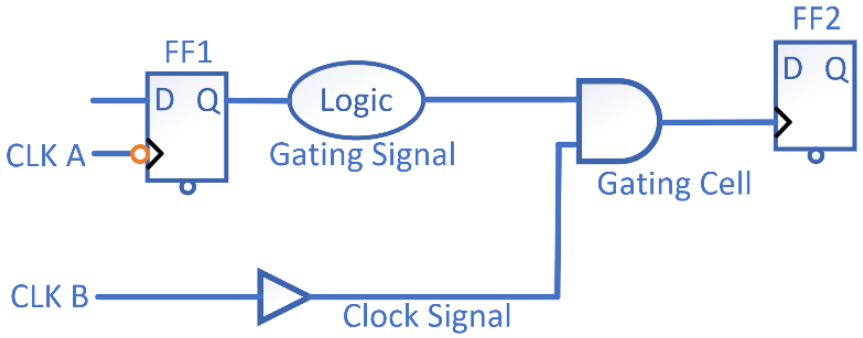


Figura 3.5: Clock Gating con puerta AND

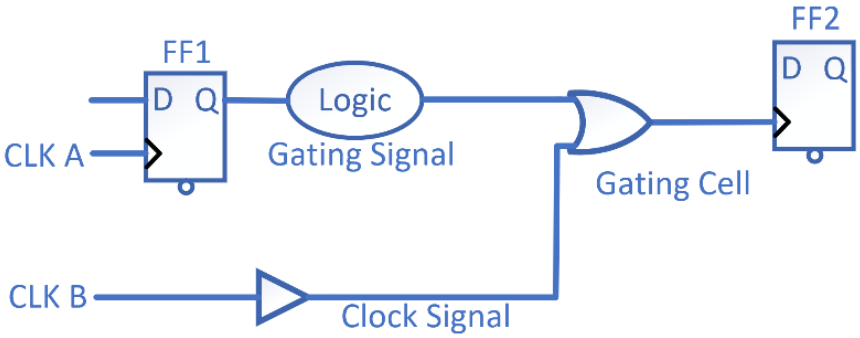


Figura 3.6: Clock Gating con puerta OR

El problema que se pretende evitar con este análisis es que se corte la señal de reloj durante su semiperiodo activo, provocando con no cumpla con las especificaciones de mínimo ancho y dando lugar a posibles *glitches*. Para ello habrá que fijarse tanto en los registros controladores de la señal de control, que indicarán si el flanco activo es positivo (*posedge*) o negativo (*negedge*), y en la propia lógica de la puerta.

En caso de que el comportamiento de la puerta sea como el de una AND, un 0 en la señal de control será dominante frente a la señal de reloj, mientras que un 1 dejará pasar el reloj sin modificaciones. Es por ello, que la señal de control debe llegar tan solo cuando el semiperiodo de

la señal de reloj sea 0, tal y como puede verse en la figura 3.7. Para conseguir este comportamiento es necesario fijarse en el detalle de la figura 3.5, en el que el reloj del FF1 es activo en el flanco negativo. Podría haberse evitado esta modificación haciendo que la lógica de la señal de control fuese más lenta (añadir *buffers*) pero sin duda la primera se trata de una solución más elegante.

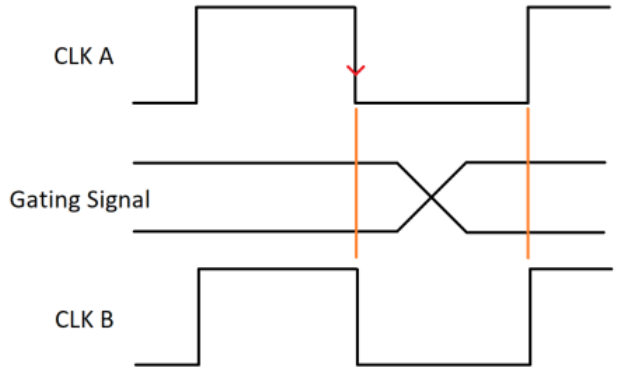


Figura 3.7: Formas de onda *Clock Gating* con puerta AND

Por el contrario, si el funcionamiento de la puerta está basado en una OR, el comportamiento será el opuesto, siendo necesario que la señal de control llegue durante el semiperiodo a 1, tal y como puede verse en la figura 3.8. Al ser necesario que esta señal llegue durante el primer semiperiodo, el registro controlador de la señal de control debe ser activo al flanco positivo, tal y como puede verse en la figura 3.6.

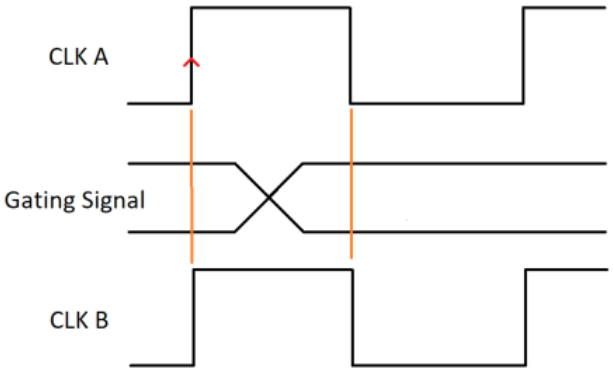


Figura 3.8: Formas de onda *Clock Gating* con puerta OR

3.1.6. Otros chequeos temporales

A parte de los principales chequeos temporales que se han explicado en las anteriores subsecciones, existen otros análisis que las herramientas de STA llevan a cabo en los diseños.

3.1.6.1. Verificación del desfase (*skew*)

Este chequeo limita el retardo máximo permitido entre dos señales, debido a que podría provocar un comportamiento inestable del diseño. Generalmente son usados en celdas con múltiples relojes. Anteriormente, se comentó que las herramientas optimizan los caminos de reloj para conseguir que estén balanceados y que lleguen a su pin de entrada en el momento oportuno. Debido a que es imposible conseguir que todos los caminos sean iguales, la verificación del desfase se encarga de comprobar que no se superan ciertos límites temporales.

3.1.6.2. Periodo de reloj

Al igual que ocurre con el mínimo ancho de pulso, las celdas secuenciales como los registros o los biestables cuentan con especificaciones en sus librerías que especifican un valor mínimo para el periodo de reloj. Para saber si se cumple este requisito, la herramienta emplea la siguiente ecuación, debiendo obtener como resultado un valor de *slack* positivo.

$$Slack = Clock\_period - Min\_Period\_Constraint - Skew \tag{3.3}$$

3.1.6.3. Tiempos de transición

Mide el tiempo que tarda una señal en cambiar de 0 a 1 o de 1 a 0, y dependiendo de las especificaciones de las librerías el chequeo se puede realizar en diferentes escalas, como del 10 % al 90 % o del 20 % al 80 %. Cabe destacar que no tienen el mismo tiempo una transición ascendente que una descendente.

3.1.6.4. Fan in y Fan out

Cada celda del diseño contiene una especificación en sus librerías acerca del número de receptores máximo que pueden ir conectados a uno de sus pines de salida. Lo mismo ocuure para los pines de entrada, y estos chequeos aseguran que no se superan ciertos límites máximos que podrían provocar un mal funcionamiento del sistema, pues la celda no está capacitada para suministrar suficiente corriente.

3.1.6.5. Verificación del retardo de ruta

Como se ha comentado en anteriores secciones, en ciertos casos se usan restricciones de diseño para añadir retardos (*delays*) a algunos caminos de conveniencia. Este chequeo se encarga de comprobar que se cumplen dichos retardos añadidos por el diseñador mediante la regla de diseño: *set_max_delay* y *set_min_delay* (en la seccion 4 se detallan las reglas de diseño).

3.2. Información temporal librerías

En esta sección se va a explicar como se presenta la información que los fabricantes entregan en sus librerías, pues las herramientas de STA la emplean para realizar los análisis temporales. Por lo tanto, es de vital importancia conocer el tipo de información que aparece en las librerías, así como la forma en la que se encuentra presentada. Además de información temporal, las librerías incluyen valores relativos al área, al consumo y la funcionalidad de las celdas que la componen.

Para la explicación de esta sección se va a tomar como ejemplo la librería de una memoria de una sola programación (*One Time Programmable*, OTP). En la primera sección de las librerías se suele incluir información acerca de las unidades que se han empleado para cada magnitud, las condiciones de operación para las cuales se han calculado los valores de dicha librería, los umbrales o *threshold* que se han tomado como referencia y ciertas tablas o *Look Up Table* (LUT), existiendo una por cada chequeo que se necesite, así como el de configuración, retención, eliminación, recuperación, mínimo ancho de pulso, etc. En la sección de código 3.1 puede verse un ejemplo de lo mencionado.

```
1  /* Units Attributes */
2   time_unit                    : "1ns";
3   leakage_power_unit           : "1pW";
4   voltage_unit                 : "1V";
5   current_unit                 : "1uA";
6   pulling_resistance_unit      : "1kohm";
7   capacitive_load_unit           (1,pf);
8
9   /* Operation Conditions */
10  nom_process                  : 1.00;
11  nom_temperature              : -40.00;
12  nom_voltage                  : 1.08;
13
14  /* Threshold Definitions */
15  slew_lower_threshold_pct_fall     : 20.00 ;
16  slew_lower_threshold_pct_rise     : 20.00 ;
17  slew_upper_threshold_pct_fall     : 80.00 ;
18  slew_upper_threshold_pct_rise     : 80.00 ;
19
20  lu_table_template (Pulse_width_4) {
21    variable_1 : related_pin_transition;
22    index_1 ("0.0010,0.0020,0.0030,0.0040");
23  }
24
25  lu_table_template (Recovery_4_4) {
26    variable_1 : constrained_pin_transition;
27    variable_2 : related_pin_transition;
28    index_1 ("0.0010,0.0020,0.0030,0.0040");
29    index_2 ("0.0010,0.0020,0.0030,0.0040");
30  }
```

Listado de código 3.1: Información general librerías

A continuación, el resto de la librería se divide por celdas (en caso de que haya más de una) y

cada una de estas por pines o buses, siendo la única diferencia que los primeros son de un solo bit mientras que los segundos son una agrupación de éstos.

En el código 3.2 puede apreciarse un ejemplo completo de la información asociada a uno de los pines de salida de la memoria. Para este bus se dispone de cierta información general así como su dirección, capacidad o aquella relacionada con la alimentación, entre las líneas 5 y 18. A partir de la línea 19 se encuentra la información temporal que las herramientas de STA emplean para sus estudios.

El primer aspecto de interés será indentificar con respecto a que pin se está realizando el estudio, en este caso será entre el pin de salida Q y el pin de entrada CK (pin del reloj).

El siguiente punto de interés se encuentra en le línea 21, pues indica el tipo de chequeo para el cual se ha preparado la próxima información. Podemos encontrar los siguientes casos:

- *falling_edge* o *rising_edge*: hacen referencia al estudio de *Clock to Q*, diferenciando entre si la celda secuencial es activa a nivel bajo (*falling*) o a nivel alto (*rising*).
- *min_pulse_width*: chequeo de ancho mínimo de pulso.
- *setup_rising* o *setup_falling*: análisis de *setup* diferenciando si está asociado a un flanco activo a nivel alto, o bajo de reloj.
- *hold_rising* o *hold_falling*: mismo caso aplicado al estudio de *hold*.
- *combinational_fall* o *combinational_rise*: valores empleados para las transiciones de las celdas combinacionales del diseño.
- *recovery_rising*, *recovery_falling*, *removal_rising* o *removal_falling*: todos los posibles chequeos aplicados a los pines asíncronos de las celdas secuenciales.

A continuación, cada pin o bus cuenta con cuatro tablas que incluyen la siguiente información temporal: *cell_rise*, *cell_fall*, *rise_transition* y *fall_transition*. Las tablas de *cell* indican el tiempo que tarda una celda desde que su entrada supera uno de los umbrales (*threshold*) (80 % en caso de ser una entrada ascendente o 20 % en caso de ser descendente) hasta que la salida hace lo propio. En cambio las tablas de transición (*transition*) indican cuanto tarda la salida en cambiar de un nivel a otro.

Para interpretar estas tablas, las librerías cuentan con índices específicos para cada tipo de chequeo. Por lo general, el primer índice se trata del valor para la transición de entrada, y el segundo para la capacidad de salida de la celda.

```
1     bus (Q) {
2       bus_type : bus_DATA;
3       direction : output;
4       max_capacitance : 0.13;
5       related_power_pin : "VDD";
6       related_ground_pin : "VSS";
7       power_down_function : "!VDD + VSS";
8       pin (Q[15:0]) {
9         internal_power () {
10          related_pg_pin : "VDD";
11          rise_power (scalar) {
```

```
12              values("0.0");
13            }
14            fall_power (scalar) {
15              values("0.0");
16            }
17          }
18        }
19        timing () {
20          related_pin : "CK";
21          timing_type : falling_edge;
22          cell_rise (delay_transition) {
23            values("6.0,   6.001, 6.002, 6.003",\
24                   "6.001, 6.002, 6.003, 6.004",\
25                   "6.002, 6.003, 6.004, 6.005",\
26                   "6.003, 6.004, 6.005, 6.006");
27          }
28          cell_fall (delay_transition) {
29            values("6.0,   6.001, 6.002, 6.003",\
30                   "6.001, 6.002, 6.003, 6.004",\
31                   "6.002, 6.003, 6.004, 6.005",\
32                   "6.003, 6.004, 6.005, 6.006");
33          }
34          rise_transition (delay_transition) {
35            values("0.005, 0.039, 0.159, 0.597",\
36                   "0.004, 0.039, 0.159, 0.596",\
37                   "0.005, 0.039, 0.159, 0.596",\
38                   "0.005, 0.039, 0.159, 0.596");
39          }
40          fall_transition (delay_transition) {
41            values("0.004, 0.033, 0.134, 0.478",\
42                   "0.005, 0.034, 0.134, 0.478",\
43                   "0.004, 0.033, 0.133, 0.478",\
44                   "0.004, 0.033, 0.133, 0.477");
45          }
46        }
47      }
```

Listado de código 3.2: Ejemplo 1 librería OTP

En la sección de código 3.3 puede verse otra metodología más descriptiva de visualizar las tablas temporales de las librerías, en las que se aprecia la forma matricial de emplear los índices, siendo 0.16, 0.35 y 1.43 para las columnas y 0.1, 0.3 y 0.7 para las filas.

```
1
2 cell_fall(delay_template_3x3) {
3     index_1 ("0.1, 0.3, 0.7"); /* Input transition */
4     index_2 ("0.16, 0.35, 1.43"); /* Output capacitance */
5     values ( /* 0.16 0.35 1.43 */ \
6     /* 0.1 */ "0.0617, 0.1537, 0.5280", \
7     /* 0.3 */ "0.0918, 0.2027, 0.5676", \
8     /* 0.7 */ "0.1034, 0.2273, 0.6452");
```

Listado de código 3.3: Ejemplo 2 librería OTP

Capítulo 4

Reglas de diseño

En este capítulo se da paso a una parte más práctica del proceso de análisis temporal estático, y se trata de la definición de reglas de diseño que acoten y preparen correctamente el diseño para que la herramienta de STA pueda llevar a cabo todos los chequeos ya explicados.

Sin embargo, estas reglas de diseño no se emplean únicamente en STA, si no que afectan a diferentes partes del proceso de diseño, ya sea a la síntesis, al emplazado y rutado o al estudio del cruce entre dominios de reloj. Debido a que no es posible realizar ningún análisis temporal estático sin al menos haber realizado la síntesis del RTL (figura 1.2), en este capítulo se van a incluir todas las restricciones (*constraints*) que un diseño real debe tener, incluso aquellas que necesitan los procesos de diseño previos al STA.

4.1. Definición de relojes

Después de haber realizado el proceso de síntesis del árbol de reloj (CTS) se cuenta con información real de los retardos de propagación, latencia u otras características propias del árbol de reloj. Sin embargo, previo a dicho proceso, algunas reglas de diseño son necesarias para poder realizar la síntesis, tales como *create_clock* o *create_generated_clock*, y otras como *set_clock_latency* proporcionan cierta información útil antes de calcular valores más precisos en PnR.

Para poder realizar un análsis de STA de todos los caminos internos (los analizados entre dos registros), tan solo es necesario definir el reloj que controle los puntos de inicio y de finalización. Para ello, es necesario establecer una regla de diseño que contenga al menos: la fuente del reloj, el periodo y la forma de onda, ya sea a través del ciclo de trabajo (*duty cycle*) o del instante en el que se producen los flancos. Dicha regla de diseño se denomina *create_clock*.

```
1
2 set clk_period 100
3
4 set clk_period_derated [expr ${clk_period} *
      ${setup_overcon_derate}]
5
6 create_clock [get_ports CLK]
7     -name CLK
```

```
8      -period ${clk_period_derated}
9      -waveform [list 0 [expr ${clk_period_derated} * 0.5]]
10
11 NOTA: set setup_overcon_derate [expr [expr 100.0 -
       $env(TCONS_SETUP_OVERCON)] / 100.0] donde TCONS_SETUP_OVERCON
       varía según el proceso, siendo cada vez menos restrictivo, por
       ejemplo: 5% SYN, 3% PnR y 1% STA
```

Listado de código 4.1: Definición de reloj principal

En la sección de código 4.1 puede verse un ejemplo de la definición de un reloj. Cabe destacar que siempre es positivo mantener cierto nivel de abstracción que ayude a la comprensión de las reglas de diseño introducidas, como al crear una variable para el valor del periodo: *clk_period*.

Tal y como se mencionó anteriormente, la definición de un reloj necesita de una fuente, en este caso el puerto CLK, un periodo, donde se considera buena práctica el sobredimensionamiento, y la definición de la forma de onda mediante el parámetro *waveform* (puede apreciarse en la línea 9 que se trata de un *duty cycle* del 50 %)

Por lo general, los diseños cuentan con más de un dominio de reloj, debido a que diferentes módulos necesitan cierta frecuencia de trabajo. Es por ello que se emplean relojes derivados de uno principal, dividiendo o multiplicando su frecuencia. Para definir estos relojes se emplea la regla de diseño: *create_generated_clock*, la cual necesita como mínimo: el pin sobre el que será aplicado, la fuente de referencia y un parámetro adicional que defina la modificación que se ha realizado con respecto al reloj original. En el código 4.2 puede verse un ejemplo de lo explicado.

```
1
2   create_generated_clock \
3     [get_pins U_digital_core/U_otp_cntrl/otp_ck_reg/Q] \
4     -name CLK_DIV_2 \
5     -source [get_ports CLK] \
6     -divide_by 2 \
7     -comment "clock divided by 2 for OTP controler"
```

Listado de código 4.2: Definición de reloj derivado

Además del parámetro *-divided_by*, existen otras opciones para indicarle a la herramienta la modificación realizada con respecto al reloj original, como *multiply_by*, o *edges*, la cual coge como referencia los flancos del reloj principal y genera la forma de onda resultante.

En la figura 4.1 puede verse un ejemplo de la utilización del parámetro *-edges*{1 5 8} para la definición de un reloj generado, la cual sería equivalente a dividir entre 3.5 el reloj de origen. Cuando se pretende conseguir un ciclo de trabajo muy preciso, esta opción permite ajustar al máximo las especificaciones del reloj.

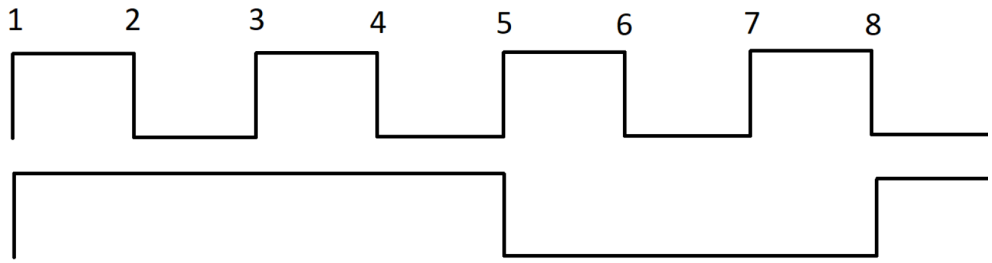


Figura 4.1: Parámetro *edges* en reloj generado

Cabe mencionar, que también es posible emplear un parámetro denominado *-duty_cycle*, pero tan solo es compatible con la opción de *multiply_by*, por lo que en el resto de casos es necesaria la utilización de los recursos anteriormente mencionados.

Una vez se han definido los relojes del diseño, es el momento de añadir ciertas restricciones que ofrezcan unas características más reales al árbol de reloj.

La primera de ellas establece un valor de transición para los flancos del reloj. En el código 4.3 puede verse un ejemplo de dicha regla de diseño. Tan solo se tendrá en cuenta para procesos previos al CTS, como la síntesis o el *pre_CTS* de PnR, posteriormente se emplearán para los análisis los valores reales resultantes del emplazado y rutado.

```
1 set transition_rise 0.6
2 set transition_fall 0.65
3
4 set_clock_transition -rise transition_rise ${all_real_clocks}
5 set_clock_transition -fall transition_fall ${all_real_clocks}
```

Listado de código 4.3: Transición del reloj

De igual modo, es necesario indicar una latencia máxima y mínima para los caminos de reloj, que posteriormente será descartada cuando sea posible calcular la real en base al estado del árbol de reloj. En el código 4.4 puede verse un ejemplo de esta regla de diseño.

```
1 set latency_min 0.4
2 set latency_max 0.6
3
4 set_clock_latency -min latency_min  [all_clocks]
5 set_clock_latency -max latency_max  [all_clocks]
```

Listado de código 4.4: Latencia de reloj

Además de las reglas de diseño anteriormente explicadas, será necesario modelar el *jitter* de reloj, que se trata de aquellas fluctuaciones aleatorias o periodicas en el periodo de reloj. Para ello, se usará la restricción de *clock_uncertainty*, tanto para *setup* como para *hold*.

```
1 set setup_clock_jitter 0.3
2 set hold_clock_jitter 0.2
3
```

```
4 set_clock_uncertainty  -setup ${setup_clock_jitter} [all_clocks]
5 set_clock_uncertainty -hold [expr  hold_clock_jitter +
    $env(TCONS_HOLD_OVERCON)] [all_clocks]
```

Listado de código 4.5: Incertidumbre periodo reloj

En la sección de código 4.5 puede verse un ejemplo de lo mencionado, haciendo de nuevo hincapié en que es posible añadir variables a todas las restricciones que permitan al diseñador sobredimensionar ciertas partes del diseño de forma diferente en cada etapa. Estos sobredimensionamientos conllevan un esfuerzo extra por parte de la herramienta para cumplir las especificaciones, asegurando en los casos más críticos un correcto desempeño del diseño.

Para finalizar con la definición de relojes es preciso definir la relación existente entre ellos. Como ya se ha comentado, un diseño real cuenta con numerosos relojes, siendo algunos de ellos derivados de uno principal, y en otros casos provienen de fuentes de reloj diferentes. Dado que estos dominios no están completamente aislados unos de otros y existe cierta interacción, es necesario incluir unas reglas de diseño que los acoten.

La regla de diseño empleada para indicar la relación entre relojes se trata del *set_clock_groups*, la cual debe incluir al menos un parámetro de caracterización y el grupo al que se aplica. En la sección de código 4.6 puede verse un ejemplo de como se definen los diferentes grupos de reloj. Para este hipotético caso, se puede apreciar en la línea 2 un grupo de relojes síncronos, ya que los divididos provienen de la fuente CLK, mientras que todos los demás grupos son asíncronos entre si. Además, se ha definido que los relojes CLK_REF y CLK_FSW son lógicamente excluyentes, lo que significa que por diseño no pueden estar ambos activos al mismo tiempo.

```
1     set_clock_groups -name i2C_clk_group -async -group {I2C_CLK}
2     set_clock_groups -name clk_group     -async -group {CLK
         CLK_DIV_2 CLK_DIV_5 CLK_SYS}
3     set_clock_groups -name fsw_clk_group -async -group {CLK_FSW}}
4     set_clock_groups -name ref_clk_group -async -group {CLK_REF}
5     set_clock_groups -name virtual_group -async -group
         {VIRTUAL_OSC_CLK}
6     set_clock_groups -name fsw_ref_clk   -logically_exclusive
         -group {CLK_REF} -group {CLK_FSW}
```

Listado de código 4.6: Dominios de reloj

Las herramientas de STA tan solo analizarán los dominios de reloj que hayan sido definidos como síncronos, es decir, los que se encuentren definidos dentro de un mismo grupo. Por otro lado, las herramientas de CDC serán las encargadas de analizar la relación entre relojes asíncronos, asegurándose de que existen mecanismos de sincronización apropiados para cada caso.

4.2. Modelado externo del ASIC

En la actualidad, el diseño digital suele ser una pequeña parte de los ASIC, donde la parte analógica sigue abarcando la mayoría del espacio disponible. A la hora de comprobar el correcto funcionamiento de la parte digital es necesario modelar aquellas partes externas con las que existe interacción, para lo cual será necesario especificar los retardos temporales que teóricamente habrá

a la entrada y a la salida, así como la fuerza de conducción (*driving strength*) de las fuentes externas conectadas a las entradas y las cargas de las salidas, es decir, la capacidad que tienen las celdas de manejar (suministrar o absorver corriente eléctrica) las salidas.

4.2.1. Retardo de entrada

Las entradas del diseño digital pueden estar conectadas externamente a diferentes partes del diseño analógico, o bien a determinados submódulos digitales del proyecto. Es por ello que para analizar correctamente los caminos *in2reg* es necesario añadir cierta información sobre las características externas de la ruta. En la figura 4.2 puede verse un ejemplo en el que un dispositivo bajo análisis (Device Under Analysis, DUA) posee lógica externa conectada a una de sus entradas.

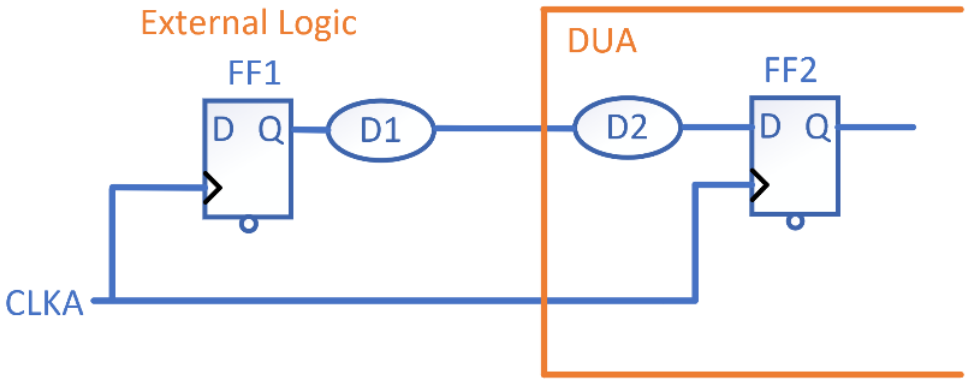


Figura 4.2: Esquema retardo de entrada

Al igual que sucedía con los *reg2reg*, es necesario que la lógica externa tenga asociado una fuente de reloj, de forma que las herramientas de STA puedan llevar a cabo los cálculos temporales. En la práctica existen varias posibilidades a la hora de elegir dicho reloj, aunque las dos más comunmente empleadas son las siguientes.

- Hacer uso del reloj con mayor frecuencia del diseño, lo cual supondría poner a todas las entradas bajo las condiciones de diseño más exigentes. Como ya se ha mencionado anteriormente, sobredimensionar las especificaciones del entorno (dentro de unos límites razonables) aumenta la seguridad funcional del diseño. Otra posibilidad complementaria a esta metodología es la de usar el mismo reloj de destino, de forma que el camino *in2reg* a analizar sea completamente síncrono.
- Otra posibilidad es emplear lo que se conoce como relojes virtuales (*virtual clocks*), los cuales se caracterizan por no tener una fuente de reloj. Debido a dicha ausencia no se incluiran en la lista de nodos del diseño y tan solo tendrán como finalidad ayudar al análisis de ciertas rutas. En algunos casos, como el estudio de CDC, cuando se emplean sincronizadores después de las entradas al diseño, la herramienta no reconocerá estas estructuras como tales a menos que el reloj del sincronizador y el reloj empleado como referencia en la regla de diseño sean asíncronos, por lo que el uso de *virtual clocks* facilita en este caso el estudio de CDC.

La regla de diseño para modelar el retardo de las entradas se conoce como *set_input_delay* y debe incluir como mínimo una fuente de reloj, un puerto de entrada del diseño y un valor de retardo. Lo correcto en esta clase de restricciones es hacer uso de los parámetros *-min* y *-max*, de forma que se modele una especie de *setup* y *hold*, es decir, un espacio temporal en torno al flanco activo de reloj en el que el valor de entrada permanezca estable.

En la sección de código 4.7 puede verse un ejemplo de como definir el retardo externo para un pin de entrada. Cabe destacar que en ciertas reglas de diseño es necesario añadir un *-add_delay* para que no se sobrescriba una restricción anterior aplicada al mismo puerto.

```
1
2 set_input_delay -min 4 -clock VIRTUAL_CLK [get_ports in_data]
3 set_input_delay -max 7 -clock VIRTUAL_CLK [get_ports in_data]
      -add_delay
```

Listado de código 4.7: Retardo externo de entrada

En la figura 4.3 se muestra un ejemplo gráfico del efecto que tendrían las reglas de diseño anteriores. Puede observarse que antes del tiempo mínimo y después del máximo no puede cambiar el valor de la señal de entrada, asegurándo que se cumplen los tiempos de *setup* y *hold* cercanos al flanco de captura.

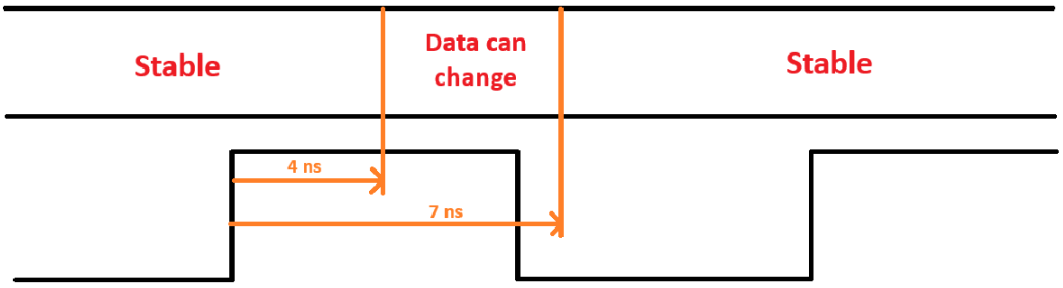


Figura 4.3: Formas de onda retardo de entrada

4.2.2. Retardo de salida

En el caso de las salidas del diseño digital, ocurre exactamente lo mismo que con las entradas, siendo necesario un modelado externo de los retardos que encontrarán los caminos de salida. En la figura 4.4 se ve de forma esquemática un ejemplo del modelado aplicado a una salida del diseño.

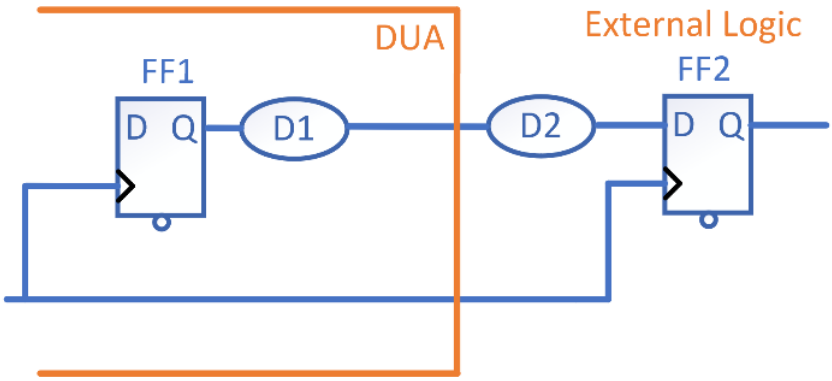


Figura 4.4: Esquema retardo de salida

La regla de diseño para el retardo de las salidas contiene los mismos parámetros que la del apartado anterior, una fuente de reloj, un puerto de salida y un retardo, tal y como se ejemplifica en la sección de código 4.8. Sin embargo, también es posible apreciar que para modelar el tiempo mínimo ha sido necesario emplear un valor negativo.

```
1
2 set_output_delay -min -4 -clock VIRTUAL_CLK [get_ports out_data]
3 set_output_delay -max 10 -clock VIRTUAL_CLK [get_ports out_data]
      -add_delay
```

Listado de código 4.8: Retardo externo de salida

Para entender dicho valor negativo es necesario tener en cuenta dos conceptos:

- La herramienta de STA tomará el flanco activo del reloj como referencia y siempre restará el valor del *output_delay*, mientras que para el *input_delay* siempre se lo sumará.
- Como regla general se puede simplificar el hecho de que los datos siempre deben permanecer estables antes del tiempo mínimo y después del tiempo máximo establecidos por las restricciones.

Teniendo en cuenta lo anterior, se puede apreciar en la figura 4.5 como se ha modelado un tiempo en torno al flanco activo de reloj que impide que se produzcan cambios, y cómo es necesario dicho valor negativo para conseguir que la estabilidad del dato perdure cierto tiempo después del flanco positivo del reloj.

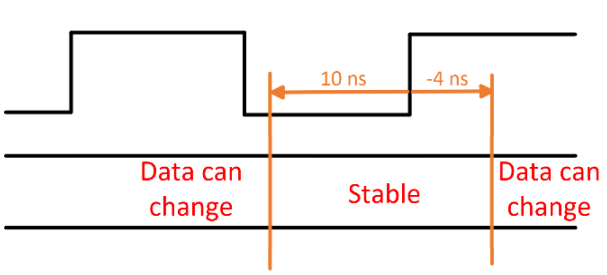


Figura 4.5: Formas de onda retardo de salida

4.2.3. Fuerza de conducción y carga de salida

Esta regla de diseño no es estrictamente necesaria para el análisis temporal estático, pero establecerla mejorará la fiabilidad de los resultados. En caso de no indicar la fuerza de conducción (*driving strength*) la herramienta asumirá un valor infinito, o lo que es equivalente, un tiempo de transición de cero en las entradas.

Para especificar la regla de diseño en cuestión es necesario establecer un tiempo de subida y bajada independiente, la celda específica que se desea emplear y los puertos de entrada a los que se va a conectar. En la sección de código 4.9 puede verse un ejemplo de lo comentado.

```
1 set_driving_cell -input_transition_rise 0.5
     -input_transition_fall 0.6 -lib_cell $env(CONS_DRIVING_CELL)
     ${all_data_ports}
2 set_driving_cell -input_transition_rise 0.4
     -input_transition_fall 0.5 -lib_cell
     $env(CONS_CLK_DRIVING_CELL) ${all_clock_ports}
3
4 set_load 1.0 [all_outputs]
5 set_load 0.2 [get_ports OTP_TM]
```

Listado de código 4.9: Fuerza de conducción y carga de salida

En la figura 4.6 puede verse un ejemplo de la modelización externa que se está llevando a cabo con estas reglas de diseño, definiendo tanto los valores de las transiciones de entrada (subida y bajada) de una celda concreta que se aplican a un conjunto de puertos, como los valores de las cargas de salida.

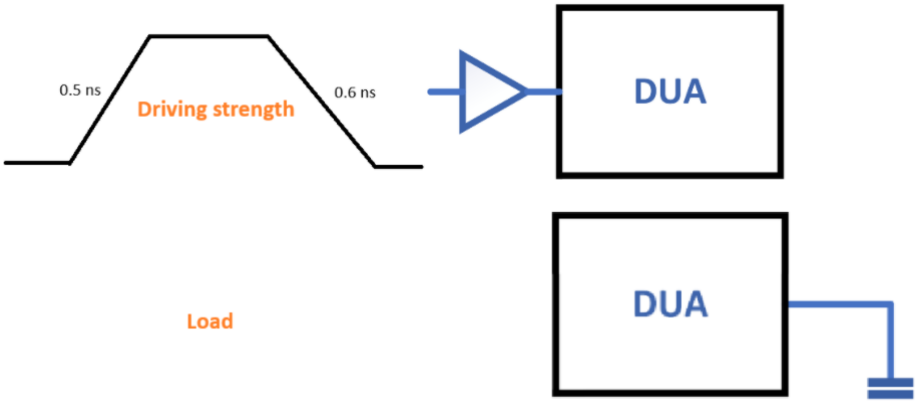


Figura 4.6: Fuerza de conducción y carga de salida

4.3. Excepciones temporales

Las herramientas de análisis temporal estático no tienen en cuenta la funcionalidad del diseño, tal y como se explicó en secciones anteriores, lo que puede provocar que en determinados escenarios se obtengan resultados adulterados debido a dicha falta de información. Por ello, los diseñadores deben especificar ciertas reglas de diseño que otorguen a la herramienta un conocimiento mayor acerca del diseño que se está analizando.

En esta subsección se explicarán cuatro restricciones que modifican el comportamiento de análisis establecido por defecto en las herramientas de STA.

4.3.1. Valores constantes

En cualquier diseño digital se emplean valores constantes o quasi-estáticos como señales de control o configuración, y dada la naturaleza de estas señales, no se suelen emplear grandes mecanismos para tratarlas a nivel funcional. Herramientas como las de STA o CDC pueden detectar que no se cumplen los estándares establecidos en estos casos, provocando violaciones o *warnings* en los informes ofrecidos. La tarea del diseñador (conocedor de la funcionalidad del diseño) es dotar a la herramienta de dicha información mediante la regla de diseño *set_case_analysis*.

Esta restricción tan solo necesita dos parámetros: el puerto de entrada o el pin de la celda sobre la que se va aplicar, y el valor que se le desea asignar. En la sección de código 4.10 se muestran varios ejemplos de la regla.

```
1
2 set_case_analysis 0 U_digital_top/U_mux0/sel[0]
3
4 set_case_analysis 1 U_digital_top/U_mux1/sel[1]
```

Listado de código 4.10: Set Case Analysis

En las reglas de diseño reales, siempre existe una etiqueta que define el caso de diseño (*case_analysis*) empleado para elegir el modo de funcionamiento del sistema. Como se comentó en la sección 1.1.1, podemos distinguir entre el modo funcional o el de escaneo, y para hacer esta selección se emplean multiplexores a través de todo el diseño, controlados por una señal de control que permite conmutar entre ambos. Por lo tanto, es común que exista un *case_analysis* en las reglas de diseño muy similar al expuesto en la sección de código 4.11.

```
1
2 set_case_analysis 0 [get_pins
      *U_digital_top/U_scan_control/scan_mode_reg/Q]
```

Listado de código 4.11: Modo funcional

4.3.2. Falsos caminos

En la sección 2.2.5 se explicó el fundamento de este tipo de caminos que en determinados casos se incluyen en el diseño, pero que debido a su irrelevancia a nivel del análisis temporal estático, no tienen que ser estudiados por la herramienta. La regla de diseño que nos permite especificar este fenómeno se denomina *set_false_path*.

Para hacer uso de esta restricción se disponen de numerosos parámetros que otorgan versatilidad al diseño, como por ejemplo la posibilidad de especificar un punto de inicio o de final, desde flanco descendente o ascendente, para *setup* o *hold*, etc. En la siguiente sección de código 4.12 se incluye un ejemplo de todos los parámetros disponibles para esta restricción.

```
1
2 set_false_path [-help] [-hold | -setup] [-rise] [-fall] {[{-from |
      -rise_from | -fall_from} <from_list>] [{-through |
      -rise_through | -fall_through} <through_list>] [{-to | -rise_to
      | -fall_to} <to_list>]}
```

Listado de código 4.12: Configuración de los falsos caminos

Durante el proceso de PnR se lleva a cabo la inserción de celdas de reserva (*spare cells*) con el objetivo de mejorar la calidad y robustez del diseño del ASIC. Existen dos tipos de celdas de reserva:

- Redundancia: estas celdas se insertan con la finalidad de suplir a otras con las mismas características que hayan resultado defectuosas, ya sea a nivel lógico o de almacenamiento.
- Desplazamiento: se utilizan para ajustar el diseño en caso de que se detecten problemas de rendimiento después de la fabricación.

Por lo tanto, dado que estas celdas se emplearan en el futuro tan solo en casos de emergencia, durante el análisis de STA no deben tenerse en cuenta. En la sección de código 4.13 puede verse el ejemplo concreto de una ruta falsa con celdas de reserva.

```
1
2 set_false_path -to [get_db pins *spr_gate*/D]
3 set_false_path -to [get_db pins *spr_gate*/CLRN]
```

Listado de código 4.13: Spare cells

Otro ejemplo aplicable a las rutas falsas se trata de la corrección de ciertas violaciones que, una vez han sido analizadas, se determina que no son reales. En la sección 5 se analizarán con detalle varias violaciones reportadas por la herramienta, siendo una de ellas las relacionadas con la activación del *reset*.

Cuando se produce la condición de *reset*, todo el sistema o una gran parte de él se lleva a un estado por defecto, por lo que resulta evidente que no es necesario cumplir chequeos como los de *setup* justo después de dicha aserción. En cambio, la herramienta a veces reporta algunas violaciones que encajan con el caso descrito, por lo que es necesario incluir en la ruta analizada un *set_false_path*. En la sección de código 4.14 puede apreciarse un ejemplo de lo comentado.

```
1
2 set_false_path -fall_through
     U_digital_top/U_clock_reset_top/i_clk_tree_clk/rst_sync_out/sync2ff
3 /u_sync_reg/Q
```

Listado de código 4.14: Activación del *reset*

Es importante apreciar que no se aplica simplemente el *set_false_path* a través de dicha celda, si no que se desactiva el estudio solamente a través de la bajada, es decir, durante la activación del reset.

4.3.3. Deshabilitación temporal

La regla *set_disable_timing* en diseño de ASICs se utiliza para desactivar el análisis temporal en ciertos caminos o bloques de un diseño. Esta directiva le dice al sintetizador o a la herramienta de análisis de temporización que ignore ciertas rutas de señal durante la verificación de tiempos. Es útil cuando se sabe que una ruta no afectará el rendimiento global del diseño o cuando se desea simplificar el análisis eliminando caminos irrelevantes.

Se trata de una regla que puede parecer muy similar al *set_false_path* analizado anteriormente, ya que en cuanto a los parámetros necesarios, cuenta con las mismas opciones. Sin embargo, sus implicaciones en el análisis temporal son diferentes, pues la regla de *set_false_path* tan solo desactivará la información de tiempo de llegada para que la herramienta no lo analice, mientras que la de *set_disable_timing* corta físicamente la propagación de los arcos de temporización, de forma que ni los valores constantes se propagarán.

Con este tipo de excepciones temporales es conveniente tener sumo cuidado a la hora de aplicarlas a un estudio temporal, pues emplearlas de forma errónea puede enmascarar problemas reales. Siempre que sea posible, se recomienda emplear otro tipo de reglas que caractericen el diseño de forma correcta, y tan solo emplear las restricciones de excepciones cuando sea extrictamente necesario.

4.3.4. Caminos de múltiples ciclos

La regla *set_multicycle_paths* se aplica para especificar que ciertos caminos en un diseño no deben ser analizados con el mismo ciclo de reloj que los caminos normales (por defecto la herramienta supone que todos los caminos necesitan un solo ciclo para propagarse). En otras palabras, indica que el camino especificado necesita más de un ciclo de reloj para propagarse desde el registro de origen hasta el registro de destino.

La sintaxis típica para *set_multicycle_paths* incluye la especificación del número de ciclos adicionales necesarios. Por ejemplo, si un camino necesita dos ciclos de reloj adicionales, se especifica el número "2", además del registro de origen y destino, tal y como puede verse en la sección de código 4.15. Para cada camino de estas características será necesario especificar a parte del tiempo de *setup* uno de *hold*, para hacer un correcto modelaje del *path*.

```
1
2 set_multicycle_paths -setup 2 -from [origen] -to [destino]
```

Listado de código 4.15: Camino de múltiples ciclos

En la figura 4.7 puede verse un ejemplo gráfico de una ruta de múltiples ciclos, dos concretamente, apreciándose el desplazamiento del punto de finalización en un ciclo más con respecto al establecido por defecto.

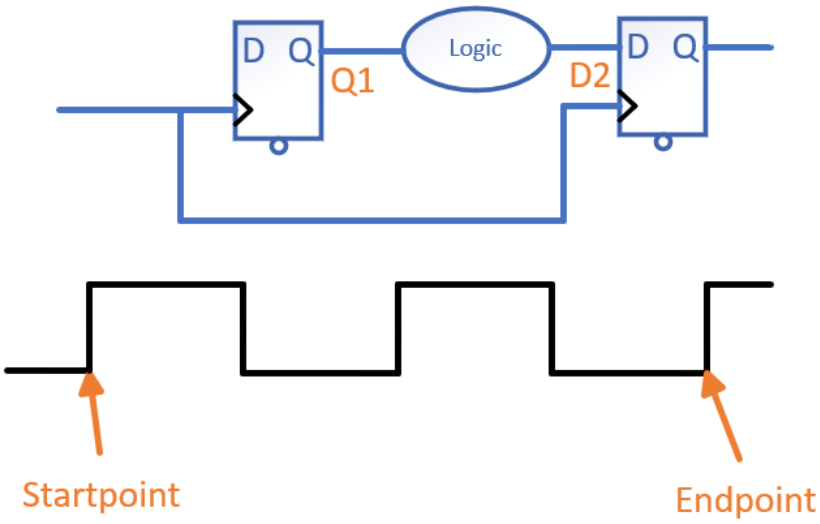


Figura 4.7: Rutas de múltiples ciclos

Capítulo 5

Análisis de informes

En este capítulo se va a detallar qué información puede ofrecer la herramienta Tempus de Cadence para realizar análisis de STA, así como la forma de obtenerla. Posteriormente se analizarán varias violaciones de un proyecto real y se presentarán soluciones a las mismas.

5.1. Informes generados por Tempus

Los informes temporales que reporta la herramienta de Tempus presentan la información siempre siguiendo un mismo estilo. Para mejorar la calidad de la explicación se va a dividir el informe en dos partes: la cabecera y los caminos detallados. En la figura 5.1 puede verse un ejemplo de la cabecera que incluyen todos los informes.

```
Path 1: MET (91.535 ns) Setup Check with Pin U_digital_core/U_nvm_cntrl/nvm_unlock_latched_q_reg/CLK->D
              View: func_ff_bcv_m40_cbest
             Group: clk_sys
        Startpoint: (F) NVM_UNLOCK_LAT_TO[2]
             Clock: (R) OSC_CLK
          Endpoint: (F) U_digital_core/U_nvm_cntrl/nvm_unlock_latched_q_reg/D
             Clock: (R) clk_sys

                      Capture        Launch
       Clock Edge:+  945.000       840.000
       Drv Adjust:+    0.082         0.057
      Src Latency:+    1.183         0.000
      Net Latency:+    0.647 (P)     0.000 (I)
          Arrival:=  946.911       840.057

            Setup:-    0.152
      Uncertainty:-    0.300
      Cppr Adjust:+    0.000
    Required Time:=  946.459
     Launch Clock:=  840.057
      Input Delay:+   12.000
        Data Path:+    2.867
            Slack:=   91.535
```

Figura 5.1: Cabecera informes

Esta cabecera primeramente informa sobre el resultado del informe, pudiendo ser: *MET* o *VIOLATED*, así como el *slack* resultante, el chequeo realizado y el pin sobre el que se ha llevado a cabo. En el ejemplo de la figura 5.1 se puede apreciar un informe con *slack* positivo en el que se ha realizado un chequeo de *setup*.

A continuación, se incluye el caso límite (*corner case*) del MMMC sobre el que se ha realizado el análisis e información relativa a los puntos de entrada y salida, así como los relojes asociados a dichos puntos del estudio.

La última parte de esta cabecera incluye un resumen temporal de las rutas de captura y lanzamiento, siendo un sumatorio de los principales retardos de cada camino. Algunos puntos clave a fijarse en este resumen son: *required time* (tiempo máximo o mínimo de llegada del dato, dependiendo del tipo del chequeo), *setup* (debido a que es una estudio de *setup* es importante fijarse en el valor que se está tratando de alcanzar), *input delay* (ya que el *startpoint* es un punto de entrada al diseño), *data path* (retardo que introduce la lógica entre los dos puntos) y el *slack* (valor final que se calcula como la diferencia entre el *required time* y el *arrival time*).

Otros tipos de chequeos incluyen valores diferentes en el sumatorio, pero todos siguen la misma metodología.

```
#-------------------------------------------------------------------------------------------------------------------------------------------
# Timing Point                                          Cell                    Arc  Fanout   Load  Trans  User      Incr   Delay   Arrival  Edge
#                                                                            Annotation          (pf)   (ns)  Derate   Delay    (ns)      (ns)
#-------------------------------------------------------------------------------------------------------------------------------------------
  NVM_UNLOCK_LAT_TO[2]                                  (arrival)               SPEF       2  0.011  0.115  -        0.012   0.012   852.069  F
  xana_control/xana_tmx/xana_tmx_scan/p80915A/ZN        NAND2X1_NLPLHS          SPEF       1  0.031  0.115  1.000    0.038   0.234   852.303  R
  xana_control/xana_tmx/xana_tmx_scan/PLI_FE_OFC11347_n_5/Z  BUFFX1_NLPLHS      SPEF       1  0.029  0.206  1.000    0.086   0.251   852.553  R
  xana_control/xana_tmx/xana_tmx_scan/p24868A/ZN        CLKNAND2X1_NLPLHS       SPEF       1  0.021  0.180  1.000    0.000   0.180   852.734  F
  xana_control/xana_tmx/xana_tmx_fix/p24947A/Z          CLKAND2X1_NLPLHS        SPEF       5  0.050  0.176  1.000    0.009   0.301   853.035  F
  U_digital_core/U_nvm_cntrl/p25150A/Z                  OR2X1_NLPLHS            SPEF       3  0.011  0.241  1.000    0.000   0.188   853.223  F
  U_digital_core/U_nvm_cntrl/p25338A/ZN                 NOR2X1_NLPLHS           SPEF       4  0.020  0.055  1.000    0.031   0.254   853.477  R
  U_digital_core/U_nvm_cntrl/Fp26896A/ZN                INVX1_NLPLHS            SPEF       4  0.014  0.280  1.000    0.000   0.071   853.548  F
  U_digital_core/U_nvm_cntrl/p26987A/ZN                 CLKNAND2X1_NLPLHS       SPEF       2  0.008  0.089  1.000    0.000   0.081   853.629  R
  U_digital_core/U_nvm_cntrl/p27090A/ZN                 OAI21X1_NLPLHS          SPEF       1  0.006  0.077  1.000    0.002   0.049   853.678  F
  U_digital_core/U_nvm_cntrl/p27097A/ZN                 XNOR2X1_NLPLHS          SPEF       1  0.003  0.099  1.000    0.000   0.104   853.782  F
  U_digital_core/U_nvm_cntrl/p27099A/Z                  AO211X1_NLPLHS          SPEF       1  0.003  0.048  1.000    0.000   0.234   854.016  F
  U_digital_core/U_nvm_cntrl/p27498A/ZN                 AOI211X1_NLPLHS         SPEF       1  0.003  0.051  1.000    0.000   0.160   854.176  R
  U_digital_core/U_nvm_cntrl/p27908A/ZN                 OAI211X1_NLPLHS         SPEF       1  0.003  0.157  1.000    0.001   0.061   854.237  F
  U_digital_core/U_nvm_cntrl/p27942A/Z                  AO211X1_NLPLHS          SPEF      13  0.039  0.074  1.000    0.004   0.365   854.601  F
  U_digital_core/U_nvm_cntrl/p29159A/ZN                 NOR2X1_NLPLHS           SPEF       2  0.004  0.149  1.000    0.000   0.117   854.718  R
  U_digital_core/U_nvm_cntrl/g39294/ZN                  NOR2BX1_NLPLHS          SPEF       1  0.006  0.098  1.000    0.000   0.107   854.825  R
  U_digital_core/U_nvm_cntrl/p29493A/Z                  CLKMUX2X1_NLPLHS        SPEF       1  0.002  0.084  1.000    0.000   0.100   854.924  F
  U_digital_core/U_nvm_cntrl/nvm_unlock_latched_q_reg/D  SDFFCNQX1_NLPLHS       SPEF       1  0.002  0.035  1.000    0.000   0.000   854.924  F
#-------------------------------------------------------------------------------------------------------------------------------------------
    Other End Path:

#-------------------------------------------------------------------------------------------------------------------------------------------
# Timing Point                                                                                                Cell                    Arc  Fanout   Load  Trans  User      Incr   Delay   Arrival  Edge
#                                                                                                                                Annotation          (pf)   (ns)  Derate   Delay    (ns)      (ns)
#-------------------------------------------------------------------------------------------------------------------------------------------
  osc_9m_clk_out_vddd                                                                                         (arrival)               SPEF       2  0.020  0.128  -        0.000   0.000   945.082  R
  CTI_ccl_buf_00331/Z                                                                                         CLKBUFFX16_NLPLHS       SPEF       1  0.132  0.128  1.000    0.000   0.121   945.203  R
  p214748365A29/ZN                                                                                            CLKNAND2X4_NLPLHS       SPEF       1  0.016  0.094  1.000    0.000   0.051   945.254  F
  CTI_cid_buf_00530/Z                                                                                         CLKBUFFX12_NLPLHS       SPEF       1  0.093  0.048  1.000    0.000   0.094   945.348  F
  p214748365A22/ZN                                                                                            CLKNAND2X4_NLPLHS       SPEF       1  0.019  0.049  1.000    0.000   0.054   945.402  R
  U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/tree_clock_mux_clk_in_sel/CTI_cid_buf_00577/Z  CLKBUFFX16_NLPLHS  SPEF  1  0.061  0.047  1.000  0.000  0.069  945.471  R
  U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/tree_clock_mux_clk_in_sel/p214748365A/Z   CLKMUX2X4_NLPLHS        SPEF       1  0.020  0.039  1.000    0.000   0.123   945.594  R
  U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/tree_clock_mux_scan_clk_int/CTI_cid_buf_00601/Z  CLKBUFFX16_NLPLHS  SPEF  1  0.079  0.051  1.000  0.000  0.079  945.673  R
  U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/tree_clock_mux_scan_clk_int/p214748365A/Z  CLKMUX2X2_NLPLHS  SPEF  1  0.015  0.050  1.000  0.000  0.111  945.784  R
  CTI_ccl_buf_00328/Z                                                                                         CLKBUFFX12_NLPLHS       SPEF       5  0.084  0.057  1.000    0.000   0.087   945.871  R
  U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_sys_tree_clk/tree_clock_mux_scan_clk_int/p214748365A/Z  CLKMUX2X2_NLPLHS  SPEF  1  0.015  0.061  1.000  0.000  0.113  945.984  R
  U_digital_core/U_clock_reset_top/U_clock_reset/CTI_cid_buf_00625/Z                                          CLKBUFFX12_NLPLHS       SPEF       4  0.034  0.058  1.000    0.000   0.063   946.047  R
  U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_sys_div_clk/clk_out_reg/Q                              SDFFCNQX1_NLPLHS        SPEF       2  0.019  0.030  1.000    0.000   0.218   946.265  R
  U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_sys_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_clk_out/p214748365A/Z  CLKMUX2X4_NLPLHS  SPEF  2  0.047  0.119  1.000  0.000  0.170  946.435  R
  U_digital_core/CTI_ccl_buf_00296/Z                                                                          CLKBUFFX16_NLPLHS       SPEF       3  0.172  0.091  1.000   -0.002   0.089   946.524  R
  U_digital_core/CTI_ccl_a_buf_00273/Z                                                                        CLKBUFFX16_NLPLHS       SPEF       7  0.201  0.083  1.000    0.000   0.131   946.655  R
  U_digital_core/CTI_ccl_a_buf_00213/Z                                                                        CLKBUFFX12_NLPLHS       SPEF      13  0.151  0.126  1.000    0.000   0.116   946.770  R
  U_digital_core/CTI_ccl_a_buf_00093/Z                                                                        CLKBUFFX12_NLPLHS       SPEF      79  0.259  0.094  1.000    0.000   0.141   946.911  R
  U_digital_core/U_nvm_cntrl/nvm_unlock_latched_q_reg/CLK                                                     SDFFCNQX1_NLPLHS        SPEF      79  0.259  0.156  1.000    0.000   0.000   946.911  R
#-------------------------------------------------------------------------------------------------------------------------------------------
```

Figura 5.2: Caminos del informe

En la figura 5.2 puede verse un ejemplo de la segunda parte, aunque no se trata de una configuración por defecto. En ella puede apreciarse un recorrido detallado de las rutas de lanzamiento y captura.

Para reportar estos análisis temporales se emplea la directiva: *report_timing*, la cual incluye numerosos parámetros de configuración que permiten realizar un estudio detallado de los caminos temporales. Un ejemplo sería el utilizado para reportar el *capture path* junto al *launch path*: *-path_type full_clock*.

En la sección de código 5.1 puede verse un resumen de los principales parámetros usados para reportar los informes temporales.

```
1
2 report_timing
3     -check_type: por defecto, los típicos estudios temporales son
          analizados, como setup, hold o removal, pero este parámetro
          permite analizar otros como el clock_period o el
          pulse_width.
4
5     -debug unconstrained: permite analizar caminos que no tienen
          un reloj asociado al startpoint o al endpoint.
6
7     -late | -early: por defecto los estudios temporales se
          realizan sobre el setup (late), y para poder visualizar el
          análisis de hold es necesario incluir la directiva -early.
8
9     -fields: este parámetro sirve para personalizar las tablas de
          caminos temporales, siendo algunas de las más relevantes:
          edge, arrival, cell, delay o phase.
10
11    -from | from_rise | from_fall: definen el startpoint del
          estudio.
12
13    -to | to_rise | to_fall: definen el endpoint del análisis.
14
15    -max_slack 0: para visualizar únicamente los paths violados.
16
17    -path_exceptions: interesante para comprobar las excepciones
          temporales (false_paths, multicycles_paths,
          disable_timings..) aplicadas.
18
19    -split_delay, añade informacion sobre cada arco temporal del
          camino.
```

Listado de código 5.1: Informes temporales

5.2. Violaciones

En esta sección se van a estudiar diferentes violaciones reportadas por la herramienta de Tempus, explicando por qué se ha producido y como es posible solucionarla, ya sea implementando cambios en el diseño (en caso de ser real) o deshabilitándola (en caso de ser una violación no relevante).

5.2.1. Violación de tipo *hold*

La primera violación que se va a analizar se trata de una tipo *hold*. Tal y como se explicó en el apartado 3.1.2, esta violación se produce debido a que el dato ha llegado demasiado rápido al registro de captura, por lo que existe la posibilidad de capturar este dato en el flanco del dato anterior. En la figura 3.2 se ejemplificaba dicha explicación.

En la figura 5.3 puede apreciarse el informe de una violación tipo *hold*, con un *slack* negativo

de 0.042 ns. De igual modo, es posible apreciar en la imagen que el *startpoint* de este análisis se trata de un puerto de entrada al diseño, por lo que ya se puede intuir que ajustando los requisitos del *set_input_delay* será posible tratar esta violación.

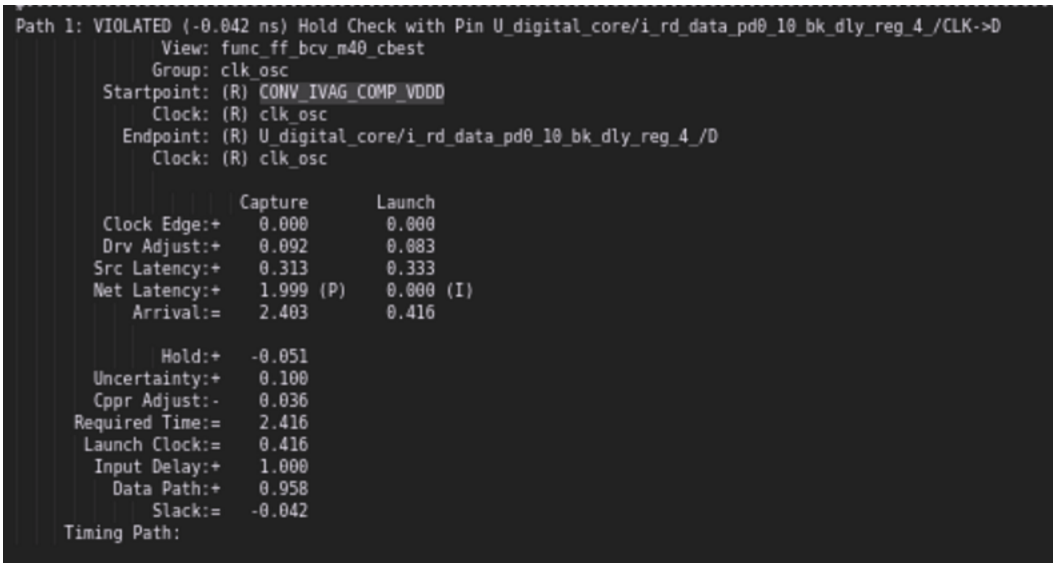


Figura 5.3: Violacion *hold*

Por lo general, durante el proceso de PnR la herramienta de rutado trata de ajustar estos caminos para que no se produzcan violaciones, pero en determinados casos no es capaz de solventar todas. Para ello, es posible sobredimensionar ciertos parámetros para exigirle un requisito mayor a la herramienta, de forma que con ese sobresfuerzo, cumplamos con todos los chequeos. En el caso de la figura anterior, la solución más pertinente será aumentar el valor del *input delay* para asegurar que el dato no llega demasiado rápido al registro que captura el dato.

En la figura 5.4 puede verse un nuevo *report* del camino anterior, pero en este caso la violación ha sido resuelta gracias al incremento del *Input Delay*, y a los *buffers* añadidos por la herramienta para cumplir dicha regla de diseño, que impiden que el dato llegue en el flanco previo de reloj.

```
Path 1: MET (8.430 ns) Hold Check with Pin U_digital_core/i_rd_data_pd0_10_bk_dly_reg_4_/CLK->D
              View: func_ff_bcv_m40_cbest
             Group: clk_osc
       Startpoint: (R) CONV_IVAG_COMP_VDDD
            Clock: (R) clk_osc
         Endpoint: (R) U_digital_core/i_rd_data_pd0_10_bk_dly_reg_4_/D
            Clock: (R) clk_osc

                    Capture       Launch
       Clock Edge:+   0.000        0.000
       Drv Adjust:+   0.000        0.062
      Src Latency:+   1.059        0.000
      Net Latency:+   0.834 (P)    0.000 (I)
          Arrival:=   1.894        0.061

                Hold:+  -0.053
       Uncertainty:+   0.200
       Cppr Adjust:-   0.000
     Required Time:=   2.040
      Launch Clock:=   0.061
       Input Delay:+  10.000
         Data Path:+   0.409
             Slack:=   8.430
```

Figura 5.4: Violacion *hold* resuelta

5.2.2. Violación de tipo *removal*

En este segundo caso se presenta una violación del tipo *removal*, la cual fue explicada con detalle en el apartado 3.1.3. Análoga a una violación de tipo *hold*, en la que el *reset* se desactiva demasiado pronto después del flanco activo de reloj, por lo que de nuevo, la solución será incluir de alguna forma retardos en el camino de *reset*.

```
Path 1: VIOLATED (-0.309 ns) Removal Check with Pin xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLK->CLRN
              View: func_ff_bcv_m40_cbest
             Group: async_default
       Startpoint: (R) xss_conv_control/U_fsw_clock_divider/rst_sync_out/falsegenblk_sync2ff/u_sync_reg/CLK
            Clock: (R) CLK_REF1
         Endpoint: (R) xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLRN
            Clock: (F) fsw_clk_div

                    Capture       Launch
       Clock Edge:+  380.000      380.000
      Src Latency:+    1.730        0.092
      Net Latency:+    1.151 (P)    2.111 (P)
          Arrival:=  382.881      382.204

          Removal:+    0.233
      Uncertainty:+    0.200
      Cppr Adjust:-    0.145
     Required Time:=  383.169
      Launch Clock:=  382.204
         Data Path:+    0.656
             Slack:=   -0.309

#-------------------------------------------------------------------------------------------------------------------------------------------------
# Timing Point                                                                                                  Cell              Delay   Arrival  Edge
#                                                                                                                               (ns)     (ns)
#-------------------------------------------------------------------------------------------------------------------------------------------------
  xss_conv_control/U_fsw_clock_divider/rst_sync_out/falsegenblk_sync2ff/u_sync_reg/CLK                          (arrival)                -   382.204  R
  xss_conv_control/U_fsw_clock_divider/rst_sync_out/falsegenblk_sync2ff/u_sync_reg/Q                            DFFCNX2_NLPLHS       0.158   382.362  R
  xss_conv_control/U_fsw_clock_divider/rst_sync_out/falsegenblk_sync2ff/Q                                       sync2ff_nlplhs         -   382.362  R
  xss_conv_control/U_fsw_clock_divider/rst_sync_out/CKI_FE_PHC934_n_35/Z                                       BUFFX1_NLPLHS        0.082   382.444  R
  xss_conv_control/U_fsw_clock_divider/rst_sync_out/p214748365A/Z                                              CLKMUX2X1_NLPLHS     0.101   382.545  R
  xss_conv_control/U_fsw_clock_divider/CKI_FE_PHC5853_rst_clk_fsw_n/Z                                          BUFFX1_NLPLHS        0.080   382.625  R
  xss_conv_control/U_fsw_clock_divider/CKI_FE_PHC4551_rst_clk_fsw_n/Z                                          BUFFX1_NLPLHS        0.150   382.775  R
  xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/p214748365A/Z  CLKMUX2X1_NLPLHS     0.086   382.860  R
  xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLRN  SDFFCSNQX1_NLPLHS    0.000   382.860  R
#-------------------------------------------------------------------------------------------------------------------------------------------------
```

Figura 5.5: Violacion *removal*

En la figura 5.5 se puede ver un ejemplo de informe temporal en el que se reporta una violación de *removal* con un *slack* de -0.309 ns. Para retrasar temporalmente la llegada del *reset* al *endpoint*

es necesario la incorporación de *buffers* a este camino, para lo cual se ha empleado la regla de diseño que puede verse en la sección de código 5.2.

```
1
2 set_min_delay -from [get_pins
      xss_conv_control/U_fsw_clock_divider/rst_sync_out/
3 falsegenblk_sync2ff/Q] -to [get_pins -hierarchical
      *ocp_pls1_reg/CLRN] 2
```

Listado de código 5.2: Constraint solución al removal

La regla de diseño *set_min_delay* introduce dos cambios relevantes: el primero de ellos se produce durante el PnR, momento en el que se añaden los *buffers* necesarios para cumplir con la especificación impuesta, y la segunda se trata de un nuevo chequeo que se realizará en STA para comprobar que ha tenido efecto la restricción, este chequeo se denomina *Path Delay Check*, el cual puede apreciarse en la figura 5.6.

```
Path 1: MET (0.104 ns) Path Delay Check with Pin xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLRN
              View: func_ff_bcv_m40_cbest
             Group: async_default
       Startpoint: (R) xss_conv_control/U_fsw_clock_divider/rst_sync_out/falsegenblk_sync2ff/Q
             Clock:
         Endpoint: (R) xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLRN
             Clock: (R) CLK_REF1

                      Capture       Launch
        Path Delay:+    2.000            -
       Src Latency:+    0.000        0.000
       Net Latency:+    0.000 (I)    0.000 (I)
           Arrival:=    2.000        0.000

           Removal:+    0.233
       Cppr Adjust:-    0.000
     Required Time:=    2.233
      Launch Clock:=    0.000
         Data Path:+    2.337
             Slack:=    0.104
```

Figura 5.6: Violacion *removal* resuelta

Ahora se puede apreciar que se está cumpliendo el chequeo y que el *slack* se ha convertido en positivo, esto ha sido gracias a los *buffers* añadidos por la herramienta durante el emplazado y rutado, y que pueden apreciarse en la lista de celdas a través de camino recorrido por el análisis en la figura 5.7.

# Timing Point	Cell
xss_conv_control/U_fsw_clock_divider/rst_sync_out/falsegenblk_sync2ff/Q	sync2ff_nlplhs
xss_conv_control/U_fsw_clock_divider/rst_sync_out/CKI_FE_PHC934_n_35/Z	BUFFX1_NLPLHS
xss_conv_control/U_fsw_clock_divider/rst_sync_out/p214748365A/Z	CLKMUX2X1_NLPLHS
xss_conv_control/U_fsw_clock_divider/CKI_FE_PHC5853_rst_clk_fsw_n/Z	BUFFX1_NLPLHS
xss_conv_control/U_fsw_clock_divider/CKI_FE_PHC4551_rst_clk_fsw_n/Z	BUFFX1_NLPLHS
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/p214748365A/Z	CLKMUX2X1_NLPLHS
xss_conv_control/CKI_FE_PHC6409_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC6488_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC6897_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7067_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7195_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7299_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7383_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7463_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7545_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7614_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7683_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7743_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7804_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7864_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7921_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC7977_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8033_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8091_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8148_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8200_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8252_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8302_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8347_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8388_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8422_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8452_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8477_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8497_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8513_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8525_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8534_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8542_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8549_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8555_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8559_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8564_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8568_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/CKI_FE_PHC8572_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS
xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLRN	SDFFCSNQX1_NLPLHS

Figura 5.7: Violacion *removal* resuelta (*path*)

5.2.3. Violación de tipo *setup*

Similar a la violación descrita en el caso 1, en este subapartado se presenta una violación de *setup*. En la sección 3.1.1 se explicaba con detalle que en algunos caminos de datos demasiado lentos, no se llegaba con suficiente antelación al flanco activo de reloj, dando lugar a que la herramienta genere una violación como la mostrada en la figura 5.8. Es posible apreciar un *slack* negativo de 0.446 ns y también es destacable que los relojes de inicio y final no son los mismos, pero debido a que son síncronos (el *clk_sys* es un reloj derivado del OSC_CLK) la herramienta puede realizar dichos análisis.

Figura 5.8: Violacion *setup*

Dado que el final del camino analizado se trata de una salida del diseño digital, será posible hacer uso de la regla de diseño *set_output_delay* para forzar a la herramienta a cuadrar la temporización de este camino. En la sección de código 5.3 puede apreciarse la regla que ha solventado la violación de tiempo de configuración, en la que se ha disminuido 2 ns el valor del retardo, relajando de esta forma la especificación.

```
1
2 set_output_delay -max 83 -clock OSC_CLK [get_ports ibb2_fccm_vddd]
```

Listado de código 5.3: Constraint solución al *removal*

Tras haber realizado un nuevo estudio de STA, en la figura 5.9 puede verse como el mismo análisis anterior es ahora favorable, con un *slack* de 1.108 ns.

Figura 5.9: Violacion *setup* resuelta

5.2.4. Violación errónea 1

Tras haber visto algunos casos de violaciones reales, en las que hay que aplicar ciertas reglas de diseño para impactar directamente el diseño RTL, en los siguientes casos se van a detallar algunas violaciones reportadas por la herramienta que o bien, no son reales por un motivo coherente, o son fallos de la herramienta de Tempus.

En este subapartado se va a ejemplificar una supuesta violación en la que una misma celda

realiza un flanco positivo en uno de los caminos y un flanco negativo en el otro, en el mismo instante temporal. Esto se trata de un error de la herramienta de Cadence, que tiene en cuenta el peor caso posible y en ciertos ocasiones se producen este tipo de errores.

En la figura 5.10 puede verse la cabecera de dicha violación, en la que se reporta una violación de eliminación (*removal*) con un *slack* negativo de 1.744 ns. A priori, no existe ninguna diferencia aparente con otras violaciones que ya han sido analizadas.

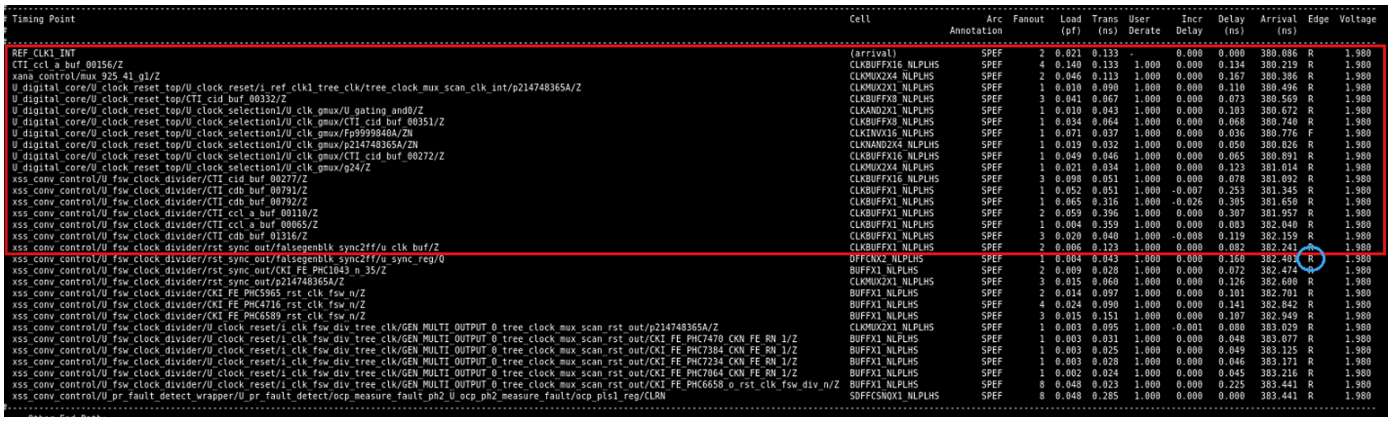
```
Path 1: VIOLATED (-1.744 ns) Removal Check with Pin xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLK->CLRN
              View: func_ff_bcv_m40_cbest
             Group: async_default
       Startpoint: (R) xss_conv_control/U_fsw_clock_divider/rst_sync_out/falsegenblk_sync2ff/u_sync_reg/CLK
            Clock: (R) CLK_REF1
         Endpoint: (R) xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLRN
            Clock: (F) fsw_clk_div

                     Capture       Launch
       Clock Edge:+  380.000      380.000
       Drv Adjust:+    0.094        0.086
      Src Latency:+    3.714        0.000
      Net Latency:+    1.318 (P)    2.156 (P)
          Arrival:=  385.126      382.241

          Removal:+    0.240
      Uncertainty:+    0.200
      Cppr Adjust:-    0.382
    Required Time:=  385.185
     Launch Clock:=  382.241
        Data Path:+    1.199
            Slack:=   -1.744
```

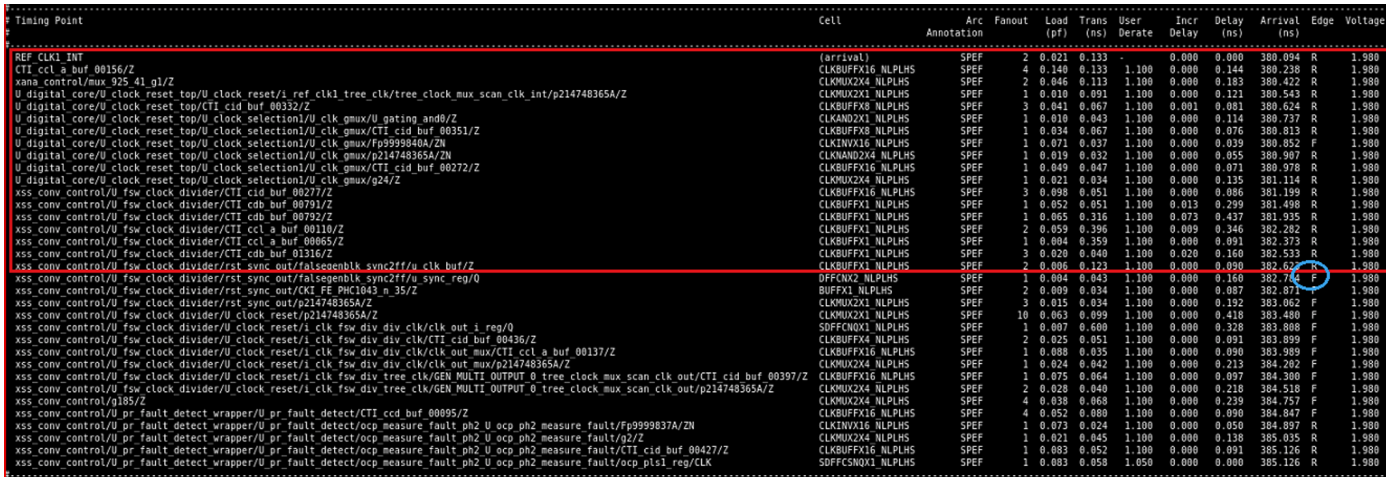
Figura 5.10: Cabecera violacion errónea

Es labor del diseñador digital entender y analizar los caminos temporales de una violación antes de empezar a incluir restricciones en el proyecto, ya que en determinados casos puede no ser una violación real. En la figura 5.11 se muestra la ruta de lanzamiento de esta violación, en el que hay que fijarse que las celdas encerradas en el cuadrado rojo son idénticas a las de la figura 5.12 y que la siguiente celda es sobre la que se produce un flanco positivo en la ruta de lanzamiento y un flanco negativo en la ruta de captura. También es posible apreciar que los tiempos de llegada no son exactamente los mismos y esto es debido a que la herramienta ha decidido aplicar un retardo más pesimista en el segundo de ellos, pero en la práctica esto no sucedería en caminos exactamente coincidentes que parten de un mismo flanco de reloj.



Timing Point	Cell	Arc Annotation	Fanout	Load (pf)	Trans (ns)	User Derate	Incr Delay	Delay (ns)	Arrival (ns)	Edge	Voltage
REF_CLK1_INT	(arrival)	SPEF	2	0.021	0.133	-	0.000	0.000	380.086	R	1.980
CTI_ccl_a_buf_00156/Z	CLKBUFFX16_NLPLHS	SPEF	4	0.140	0.133	1.000	0.000	0.134	380.219	R	1.980
xana_control/mux_925_41_g1/Z	CLKMUX2X4_NLPLHS	SPEF	2	0.046	0.113	1.000	0.000	0.167	380.386	R	1.980
U_digital_core/U_clock_reset_top/U_clock_reset/i_ref_clk1_tree_clk/tree_clock_mux_scan_clk_int/p214748365A/Z	CLKMUX2X1_NLPLHS	SPEF	1	0.010	0.090	1.000	0.000	0.110	380.496	R	1.980
U_digital_core/U_clock_reset_top/CTI_cid_buf_00332/Z	CLKBUFFX8_NLPLHS	SPEF	3	0.041	0.067	1.000	0.000	0.073	380.569	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/U_gating_and0/Z	CLKAND2X1_NLPLHS	SPEF	1	0.010	0.043	1.000	0.000	0.103	380.672	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/CTI_cid_buf_00351/Z	CLKBUFFX8_NLPLHS	SPEF	1	0.034	0.064	1.000	0.000	0.068	380.740	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/Fp9999840A/ZN	CLKINVX16_NLPLHS	SPEF	1	0.071	0.037	1.000	0.000	0.036	380.776	F	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/p214748365A/ZN	CLKNAND2X4_NLPLHS	SPEF	1	0.019	0.032	1.000	0.000	0.050	380.826	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/CTI_cid_buf_00272/Z	CLKBUFFX16_NLPLHS	SPEF	1	0.049	0.046	1.000	0.000	0.065	380.891	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/g24/Z	CLKMUX2X4_NLPLHS	SPEF	1	0.021	0.034	1.000	0.000	0.123	381.014	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_cid_buf_00277/Z	CLKBUFFX16_NLPLHS	SPEF	3	0.098	0.051	1.000	0.000	0.078	381.092	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_cdb_buf_00791/Z	CLKBUFFX1_NLPLHS	SPEF	1	0.052	0.051	1.000	-0.007	0.253	381.345	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_cdb_buf_00792/Z	CLKBUFFX1_NLPLHS	SPEF	1	0.065	0.316	1.000	-0.026	0.305	381.650	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_ccl_a_buf_00110/Z	CLKBUFFX1_NLPLHS	SPEF	2	0.059	0.396	1.000	0.000	0.307	381.957	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_ccl_a_buf_00065/Z	CLKBUFFX1_NLPLHS	SPEF	1	0.004	0.359	1.000	0.000	0.083	382.040	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_cdb_buf_01316/Z	CLKBUFFX1_NLPLHS	SPEF	3	0.020	0.040	1.000	-0.008	0.119	382.159	R	1.980
xss_conv_control/U_fsw_clock_divider/rst_sync_out/falsegenblk_sync2ff/u_clk_buf/Z	CLKBUFFX1_NLPLHS	SPEF	2	0.006	0.123	1.000	0.000	0.082	382.241	R	1.980
xss_conv_control/U_fsw_clock_divider/rst_sync_out/falsegenblk_sync2ff/u_sync_reg/Q	DFFCNX2_NLPLHS	SPEF	1	0.004	0.043	1.000	0.000	0.160	382.401	R	1.980
xss_conv_control/U_fsw_clock_divider/rst_sync_out/CKI_FE_PHC1043_n_35/Z	BUFFX1_NLPLHS	SPEF	2	0.009	0.028	1.000	0.000	0.072	382.474	R	1.980
xss_conv_control/U_fsw_clock_divider/rst_sync_out/p214748365A/Z	CLKMUX2X1_NLPLHS	SPEF	3	0.015	0.060	1.000	0.000	0.126	382.600	R	1.980
xss_conv_control/U_fsw_clock_divider/CKI_FE_PHC5965_rst_clk_fsw_n/Z	BUFFX1_NLPLHS	SPEF	2	0.014	0.097	1.000	0.000	0.101	382.701	R	1.980
xss_conv_control/U_fsw_clock_divider/CKI_FE_PHC4716_rst_clk_fsw_n/Z	BUFFX1_NLPLHS	SPEF	4	0.024	0.090	1.000	0.000	0.141	382.842	R	1.980
xss_conv_control/U_fsw_clock_divider/CKI_FE_PHC6589_rst_clk_fsw_n/Z	BUFFX1_NLPLHS	SPEF	3	0.015	0.151	1.000	0.000	0.107	382.949	R	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/p214748365A/Z	CLKMUX2X1_NLPLHS	SPEF	1	0.003	0.095	1.000	-0.001	0.080	383.029	R	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/CKI_FE_PHC7470_CKN_FE_RN_1/Z	BUFFX1_NLPLHS	SPEF	1	0.003	0.031	1.000	0.000	0.048	383.077	R	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/CKI_FE_PHC7384_CKN_FE_RN_1/Z	BUFFX1_NLPLHS	SPEF	1	0.003	0.025	1.000	0.000	0.049	383.125	R	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/CKI_FE_PHC7234_CKN_FE_RN_1/Z	BUFFX1_NLPLHS	SPEF	1	0.003	0.028	1.000	0.000	0.046	383.171	R	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/CKI_FE_PHC7064_CKN_FE_RN_1/Z	BUFFX1_NLPLHS	SPEF	1	0.002	0.024	1.000	0.000	0.045	383.216	R	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/CKI_FE_PHC6658_o_rst_clk_fsw_div_n/Z	BUFFX1_NLPLHS	SPEF	8	0.048	0.023	1.000	0.000	0.225	383.441	R	1.980
xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLRN	SDFFCSNQX1_NLPLHS	SPEF	8	0.048	0.285	1.000	0.000	0.000	383.441	R	1.980

Figura 5.11: *Launch path* violacion errónea



Timing Point	Cell	Arc Annotation	Fanout	Load (pf)	Trans (ns)	User Derate	Incr Delay	Delay (ns)	Arrival (ns)	Edge	Voltage
REF_CLK1_INT	(arrival)	SPEF	2	0.021	0.133	-	0.000	0.000	380.094	R	1.980
CTI_ccl_a_buf_00156/Z	CLKBUFFX16_NLPLHS	SPEF	4	0.140	0.133	1.100	0.000	0.144	380.238	R	1.980
xana_control/mux_925_41_g1/Z	CLKMUX2X4_NLPLHS	SPEF	2	0.046	0.113	1.100	0.000	0.183	380.422	R	1.980
U_digital_core/U_clock_reset_top/U_clock_reset/i_ref_clk1_tree_clk/tree_clock_mux_scan_clk_int/p214748365A/Z	CLKMUX2X1_NLPLHS	SPEF	1	0.010	0.091	1.100	0.000	0.121	380.543	R	1.980
U_digital_core/U_clock_reset_top/CTI_cid_buf_00332/Z	CLKBUFFX8_NLPLHS	SPEF	3	0.041	0.067	1.100	0.001	0.081	380.624	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/U_gating_and0/Z	CLKAND2X1_NLPLHS	SPEF	1	0.010	0.043	1.100	0.000	0.114	380.737	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/CTI_cid_buf_00351/Z	CLKBUFFX8_NLPLHS	SPEF	1	0.034	0.067	1.100	0.000	0.076	380.813	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/Fp9999840A/ZN	CLKINVX16_NLPLHS	SPEF	1	0.071	0.037	1.100	0.000	0.039	380.852	F	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/p214748365A/ZN	CLKNAND2X4_NLPLHS	SPEF	1	0.019	0.032	1.100	0.000	0.055	380.907	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/CTI_cid_buf_00272/Z	CLKBUFFX16_NLPLHS	SPEF	1	0.049	0.047	1.100	0.000	0.071	380.978	R	1.980
U_digital_core/U_clock_reset_top/U_clock_selection1/U_clk_gmux/g24/Z	CLKMUX2X4_NLPLHS	SPEF	1	0.021	0.034	1.100	0.000	0.135	381.114	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_cid_buf_00277/Z	CLKBUFFX16_NLPLHS	SPEF	3	0.098	0.051	1.100	0.000	0.086	381.199	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_cdb_buf_00791/Z	CLKBUFFX1_NLPLHS	SPEF	1	0.052	0.051	1.100	0.013	0.299	381.498	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_cdb_buf_00792/Z	CLKBUFFX1_NLPLHS	SPEF	1	0.065	0.316	1.100	0.073	0.437	381.935	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_ccl_a_buf_00110/Z	CLKBUFFX1_NLPLHS	SPEF	2	0.059	0.396	1.100	0.009	0.346	382.282	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_ccl_a_buf_00065/Z	CLKBUFFX1_NLPLHS	SPEF	1	0.004	0.359	1.100	0.000	0.091	382.373	R	1.980
xss_conv_control/U_fsw_clock_divider/CTI_cdb_buf_01316/Z	CLKBUFFX1_NLPLHS	SPEF	3	0.020	0.040	1.100	0.020	0.160	382.533	R	1.980
xss_conv_control/U_fsw_clock_divider/rst_sync_out/falseqenblk_sync2ff/u_clk_buf/Z	CLKBUFFX1_NLPLHS	SPEF	2	0.006	0.123	1.100	0.000	0.090	382.623	R	1.980
xss_conv_control/U_fsw_clock_divider/rst_sync_out/falseqenblk_sync2ff/u_sync_reg/Q	DFFCNX2_NLPLHS	SPEF	1	0.004	0.043	1.100	0.000	0.160	382.783	F	1.980
xss_conv_control/U_fsw_clock_divider/rst_sync_out/CK1_FE_PHC1043_n_35/Z	BUFFX1_NLPLHS	SPEF	2	0.009	0.034	1.100	0.000	0.087	382.871	F	1.980
xss_conv_control/U_fsw_clock_divider/rst_sync_out/p214748365A/Z	CLKMUX2X1_NLPLHS	SPEF	3	0.015	0.034	1.100	0.000	0.192	383.062	F	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/p214748365A/Z	CLKMUX2X1_NLPLHS	SPEF	10	0.063	0.099	1.100	0.000	0.418	383.480	F	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_div_clk/clk_out_i_reg/Q	SDFFCNQX1_NLPLHS	SPEF	1	0.007	0.600	1.100	0.000	0.328	383.808	F	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_div_clk/CTI_cid_buf_00436/Z	CLKBUFFX4_NLPLHS	SPEF	2	0.025	0.051	1.100	0.000	0.091	383.899	F	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_div_clk/clk_out_mux/CTI_ccl_a_buf_00137/Z	CLKBUFFX16_NLPLHS	SPEF	1	0.088	0.035	1.100	0.000	0.090	383.989	F	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_div_clk/clk_out_mux/p214748365A/Z	CLKMUX2X4_NLPLHS	SPEF	1	0.024	0.042	1.100	0.000	0.213	384.202	F	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_clk_out/CTI_cid_buf_00397/Z	CLKBUFFX16_NLPLHS	SPEF	1	0.075	0.064	1.100	0.000	0.097	384.300	F	1.980
xss_conv_control/U_fsw_clock_divider/U_clock_reset/i_clk_fsw_div_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_clk_out/p214748365A/Z	CLKMUX2X4_NLPLHS	SPEF	2	0.028	0.040	1.100	0.000	0.218	384.518	F	1.980
xss_conv_control/g185/Z	CLKMUX2X4_NLPLHS	SPEF	4	0.038	0.068	1.100	0.000	0.239	384.757	F	1.980
xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/CTI_ccd_buf_00095/Z	CLKBUFFX16_NLPLHS	SPEF	4	0.052	0.080	1.100	0.000	0.090	384.847	F	1.980
xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/Fp9999837A/ZN	CLKINVX16_NLPLHS	SPEF	1	0.073	0.024	1.100	0.000	0.050	384.897	R	1.980
xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/g2/Z	CLKMUX2X4_NLPLHS	SPEF	1	0.021	0.045	1.100	0.000	0.138	385.035	R	1.980
xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/CTI_cid_buf_00427/Z	CLKBUFFX16_NLPLHS	SPEF	1	0.083	0.052	1.100	0.000	0.091	385.126	R	1.980
xss_conv_control/U_pr_fault_detect_wrapper/U_pr_fault_detect/ocp_measure_fault_ph2_U_ocp_ph2_measure_fault/ocp_pls1_reg/CLK	SDFFCSNQX1_NLPLHS	SPEF	1	0.083	0.058	1.050	0.000	0.000	385.126	R	1.980

Figura 5.12: *Capture path* violacion errónea

Cuando se encuentran informes temporales de estas características, es oportuno contactar con el soporte técnico de la herramienta para que colaboren en la identificación del problema y su solución en la medida de lo posible, pues en ciertos casos no comparten respuestas muy concluyentes.

La solución para este problema pasaría por añadir una de las excepciones temporales estudiadas en el apartado 4.3, como por ejemplo, un *false_path* o un *disable_timing*.

5.2.5. Violación errónea 2

La ultima violación que se va a analizar en esta sección se trata de un supuesto error de configuración (*setup/late check*), concretamente el que puede verse en la figura 5.13.

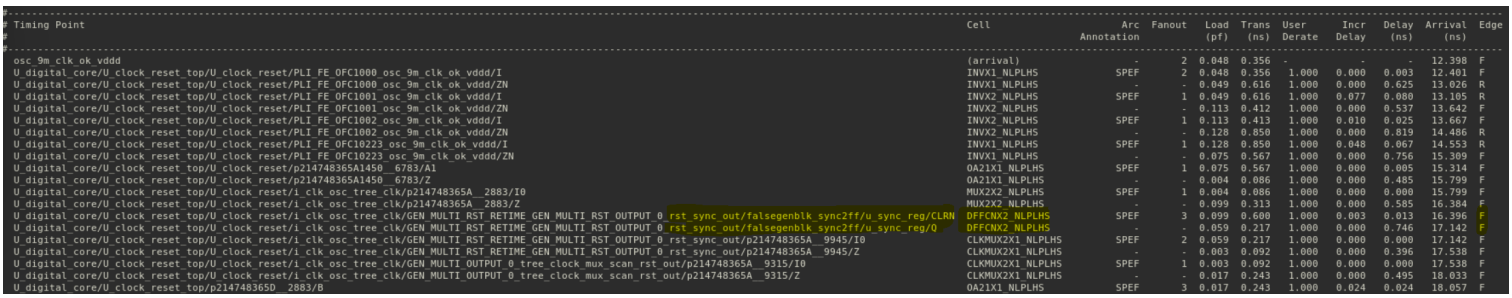
```
Path 1: VIOLATED (-14.584 ns) Late Output Delay Assertion
              View: func_ss_wcv_125_cworst
             Group: OSC_CLK
        Startpoint: (F) osc_9m_clk_ok_vddd
             Clock: (R) OSC_CLK
          Endpoint: (F) ibb1_fccm_vddd
             Clock: (R) OSC_CLK

                      Capture       Launch
       Clock Edge:+   105.000        0.000
       Drv Adjust:+     0.000        0.398
      Src Latency:+     0.000        0.000
      Net Latency:+     0.000 (P)    0.000 (I)
          Arrival:=   105.000        0.398

     Output Delay:-    83.000
      Uncertainty:-     0.300
      Cppr Adjust:+     0.000
    Required Time:=    21.700
     Launch Clock:=     0.398
      Input Delay:+    12.000
        Data Path:+    23.886
            Slack:=   -14.584
```

Figura 5.13: Cabecera violacion falsa

Aunque a priori esta violación vuelve a parecer real, se trata de un caso que no posee una relevancia real para el diseño, pues si se analiza el camino temporal de la figura 5.14, puede apreciarse que se está produciendo la activación del *reset*. Debido a este detalle, no es relevante si en los siguientes registros no se va a cumplir el *setup, hold, removal*... ya que todas las celdas secuenciales del diseño van a ir a un estado conocido y controlado.



Timing Point	Cell	Arc Annotation	Fanout	Load (pf)	Trans (ns)	User Derate	Incr Delay	Delay (ns)	Arrival (ns)	Edge
osc_9m_clk_ok_vddd	(arrival)	-	2	0.048	0.356	-	-	-	12.398	F
U_digital_core/U_clock_reset_top/U_clock_reset/PLI_FE_OFC1000_osc_9m_clk_ok_vddd/I	INVX1_NLPLHS	SPEF	2	0.048	0.356	1.000	0.000	0.003	12.401	F
U_digital_core/U_clock_reset_top/U_clock_reset/PLI_FE_OFC1000_osc_9m_clk_ok_vddd/ZN	INVX1_NLPLHS	-	-	0.049	0.616	1.000	0.000	0.625	13.026	R
U_digital_core/U_clock_reset_top/U_clock_reset/PLI_FE_OFC1001_osc_9m_clk_ok_vddd/I	INVX2_NLPLHS	SPEF	1	0.049	0.616	1.000	0.077	0.080	13.105	R
U_digital_core/U_clock_reset_top/U_clock_reset/PLI_FE_OFC1001_osc_9m_clk_ok_vddd/ZN	INVX2_NLPLHS	-	-	0.113	0.412	1.000	0.000	0.537	13.642	F
U_digital_core/U_clock_reset_top/U_clock_reset/PLI_FE_OFC1002_osc_9m_clk_ok_vddd/I	INVX2_NLPLHS	SPEF	1	0.113	0.413	1.000	0.010	0.025	13.667	F
U_digital_core/U_clock_reset_top/U_clock_reset/PLI_FE_OFC1002_osc_9m_clk_ok_vddd/ZN	INVX2_NLPLHS	-	-	0.128	0.850	1.000	0.000	0.819	14.486	R
U_digital_core/U_clock_reset_top/U_clock_reset/PLI_FE_OFC10223_osc_9m_clk_ok_vddd/I	INVX1_NLPLHS	SPEF	1	0.128	0.850	1.000	0.048	0.067	14.553	R
U_digital_core/U_clock_reset_top/U_clock_reset/PLI_FE_OFC10223_osc_9m_clk_ok_vddd/ZN	INVX1_NLPLHS	-	-	0.075	0.567	1.000	0.000	0.756	15.309	F
U_digital_core/U_clock_reset_top/U_clock_reset/p214748365A1450__6783/A1	OA21X1_NLPLHS	SPEF	1	0.075	0.567	1.000	0.000	0.005	15.314	F
U_digital_core/U_clock_reset_top/U_clock_reset/p214748365A1450__6783/Z	OA21X1_NLPLHS	-	-	0.004	0.086	1.000	0.000	0.485	15.799	F
U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/p214748365A__2883/I0	MUX2X2_NLPLHS	SPEF	1	0.004	0.086	1.000	0.000	0.000	15.799	F
U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/p214748365A__2883/Z	MUX2X2_NLPLHS	-	-	0.099	0.313	1.000	0.000	0.585	16.384	F
U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/GEN_MULTI_RST_RETIME_GEN_MULTI_RST_OUTPUT_0_rst_sync_out/falsegenblk_sync2ff/u_sync_reg/CLRN	DFFCNX2_NLPLHS	SPEF	3	0.099	0.600	1.000	0.003	0.013	16.396	F
U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/GEN_MULTI_RST_RETIME_GEN_MULTI_RST_OUTPUT_0_rst_sync_out/falsegenblk_sync2ff/u_sync_reg/Q	DFFCNX2_NLPLHS	-	-	0.059	0.217	1.000	0.000	0.746	17.142	F
U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/GEN_MULTI_RST_RETIME_GEN_MULTI_RST_OUTPUT_0_rst_sync_out/p214748365A__9945/I0	CLKMUX2X1_NLPLHS	SPEF	2	0.059	0.217	1.000	0.000	0.000	17.142	F
U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/GEN_MULTI_RST_RETIME_GEN_MULTI_RST_OUTPUT_0_rst_sync_out/p214748365A__9945/Z	CLKMUX2X1_NLPLHS	-	-	0.003	0.092	1.000	0.000	0.396	17.538	F
U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/p214748365A__9315/I0	CLKMUX2X1_NLPLHS	SPEF	1	0.003	0.092	1.000	0.000	0.000	17.538	F
U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk/GEN_MULTI_OUTPUT_0_tree_clock_mux_scan_rst_out/p214748365A__9315/Z	CLKMUX2X1_NLPLHS	-	-	0.017	0.243	1.000	0.000	0.495	18.033	F
U_digital_core/U_clock_reset_top/p214748365D__2883/B	OA21X1_NLPLHS	SPEF	3	0.017	0.243	1.000	0.024	0.024	18.057	F

Figura 5.14: Camino temporal violación falsa

Para evitar que se reporte esta violación, es posible emplear un *set_false_path* que desactive tan solo los caminos que se produzcan después de la desactivación de dicho *reset*, tal y como puede verse en la sección de código 5.4.

```
1
2 set_false_path -fall_through
      U_digital_core/U_clock_reset_top/U_clock_reset/i_clk_osc_tree_clk
3     /GEN_MULTI_RST_RETIME_GEN_MULTI_RST_OUTPUT_0_rst_sync_out
4     /falsegenblk_sync2ff/u_sync_reg/Q
```

Listado de código 5.4: Restricción para solucionar la falsa violación

Capítulo 6

Optimización del flujo de diseño

El estudio de STA es una de las muchas disciplinas en las que el diseño de un ASIC puede ser dividido, entrando en colaboración con otras de ellas como pueden ser: el diseño del RTL, el estudio de CDC, RDC y LINT, el diseño físico, la parte analógica del chip, etc. Es por ello que entran en juego muchos procesos que no dependen directamente del diseñador encargado de realizar el estudio de STA, y aunque la comunicación entre departamentos es importante, en otros casos resulta más eficiente la creación de procesos automatizados.

Todos los conceptos estudiados en este tabajo de fin de máster han sido plasmados en uno de los proyectos llevados a cabo por la empresa Analog Devices, especializada en el desarrollo de circuitos integrados. Además de la realización de los análisis temporales descritos en anteriores secciones, se han desarrollado ciertos programas que permitan a futuros diseñadores encargados de esta disciplina, un ahorro de tiempo y esfuerzo a la hora de plasmar las reglas de diseño de los puertos de entrada y salida.

Por lo general, los diseños digitales pueden tener cientos o incluso miles de entradas y salidas, concretamente el proyecto realizado contaba con alrededor de 800 pines de este tipo. Es por ello, que realizar el proceso de modelado externo a través de los parámetros estudiados en la sección 4.2 puede resultar un proceso tedioso y lleno de errores como no se realice de forma ordenada y controlada.

Entre el cliente y la empresa de diseño suele existir algún documento en el que se especifiquen las características generales del ASIC, incluyendo los pines de entrada y salida del diseño digital. Este documento puede ser creado por una de las partes de forma unilateral, o bien en conjunto durante las fases iniciales del proyecto. En el caso específico de Analog Devices, el diseño contaba con un archivo excel en el que se indican todos los puertos existentes, por lo que hacer uso de este fichero es una de las mejores opciones para ahorrar tiempo y prevenir errores.

Sin embargo, a lo largo del proceso de diseño los puertos cambian y se adaptan a las nuevas necesidades, por lo que es importante estar al tanto de todos esos cambios. El primero de los programas realizados tiene justamente esta finalidad: encontrar las diferencias existentes entre el fichero compartido con el cliente y aquel interno en el que se han ido desarrollando todas las restricciones, de forma que si se añaden o eliminan puertos, la actualización del archivo se lleve a cabo sin errores.

```
1
2  import os
3  import re
4  import pandas as pd
5
6  # Rutas a los archivos Excel
7  excel_file1 = 'Connectivity_JA03_digital.xls'
8  excel_file2 = 'internal_connectivity.xlsx'
9
10 # Ruta al archivo de resultados
11 txt_file_path = 'results.txt'
12
13 # Lista de nombres de las hojas a procesar en el primer archivo
       Excel
14 sheet_names1 = ['JA03Q', 'WLP', 'TOP', 'DIG_TOP', 'DIG', 'VPOS1',
       'ANA', 'SBIA_TOP', 'SPMI', 'IO', 'OSC_4M_1', 'OSC_4M_2']
15
16 # Nombre de la hoja en el segundo archivo Excel
17 sheet_name2 = 'Hoja1'
18
19 # Inicializar listas para almacenar los valores de los nombres de
       los puertos de cada archivo Excel
20 port_names1 = []
21 port_names2 = []
22
23 # Procesar cada hoja en el primer archivo Excel
24 for sheet_name in sheet_names1:
25     df = pd.read_excel(excel_file1, sheet_name=sheet_name,
           header=1)
26     yes_values = df[df['D cell port'] == 'yes']
27     port_names1.extend(yes_values['Net name'].str.strip())  #
           Eliminar espacios en blanco al final de cada valor
28
29 # Procesar la única hoja en el segundo archivo Excel
30 df2 = pd.read_excel(excel_file2, sheet_name=sheet_name2)
31 port_names2 = df2['Port Name'].str.replace(r'\s*\[.*\]', '',
       regex=True).tolist()
32
33 # Función para comparar los nombres permitiendo algunas
       diferencias específicas
34 def compare_names(name1, name2):
35     # Verificar si los nombres son cadenas
36     if not isinstance(name1, str) or not isinstance(name2, str):
37         return False
38
39     # Lista de expresiones regulares para casos permitidos
40     special_cases = [
41         r'#',   # '#' como comodín para cualquier número
42         r'[rgb]',  # 'R', 'G', o 'B' como letras individuales
43         r'p[12]',  # 'P1' o 'P2'
44         r'\d+'
45         # Agrega más expresiones regulares según sea necesario
46     ]
```

```
47
48      # Función auxiliar para limpiar los nombres de los puertos
49      def clean_name(name):
50          for case in special_cases:
51              name = re.sub(case, '', name)
52          return name
53
54      # Limpiar los nombres de los puertos antes de comparar
55      clean_name1 = clean_name(name1)
56      clean_name2 = clean_name(name2)
57
58      # Realizar la comparación
59      return clean_name1 == clean_name2
60
61
62  # Encontrar valores presentes en una lista pero no en la otra, con
        la comparación personalizada
63  values_in_file1_not_in_file2 = [str(name1) for name1 in
        port_names1 if not any(compare_names(name1, name2) for name2 in
        port_names2)]
64  values_in_file2_not_in_file1 = [str(name2) for name2 in
        port_names2 if not any(compare_names(name2, name1) for name1 in
        port_names1)]
65
66  # Write the results to the TXT file
67  with open(txt_file_path, 'w') as txt_file:
68      if values_in_file1_not_in_file2:
69          txt_file.write("################ Values in Oficial but not
                in mine:\n")
70          for value in values_in_file1_not_in_file2:
71              txt_file.write(f"{value}\n")
72      if values_in_file2_not_in_file1:
73          txt_file.write("################ Values in mine but not
                in Oficial:\n")
74          for value in values_in_file2_not_in_file1:
75              txt_file.write(f"{value}\n")
```

Listado de código 6.1: Automatización comparación ficheros

En la sección de código 6.1 puede verse el código del programa realizado para el control de cambios en el diseño. Entre las líneas 1 y 5 se incluyen las librerías necesarias para analizar la información presente en archivos excel. A continuación, hasta la línea 12, se definen tanto los dos archivos excel a comparar como un fichero de texto donde se almacenarán los resultados de la comparación.

Para poder procesar información en un archivo excel, es necesario conocer el nombre de las hojas internas de las que se quiere recoger información, esto puede verse en las líneas 14 y 17. A la hora de realizar la comparación, primero se van a almacenar todos los puertos de entrada y salida en variables, y posteriormente se realizará la comparación entre estas cadenas.

A continuación, es necesario saber qué datos son relevantes dentro de cada archivo, por lo que hay que buscar el patrón adecuado. En este proyecto, se incluye siempre una columna llamada *D cell port* que en caso de contener la palabra *yes*, se tratará de un puerto presente en el diseño digital,

y será agregado a la variable de almacenamiento. A la hora de realizar estas búsquedas, puede ser también necesario agregar o eliminar ciertos patrones, como espacios en blanco o determinados sufijos innecesarios.

Entre la línea 34 y 60 se define una función empleada para comparar si un puerto de una cadena se encuentra en la otra, para lo cual se han tenido en cuentra ciertos casos especiales, ya que en el documento oficial se incluían puertos repetidos para diferentes bloques, y sufijos numéricos o algunas letras características eran empleadas. La comparación se realizará entendiendo que estos casos especiales pueden ser tratados como un solo puerto.

En las líneas 63 y 64 se hace uso de dicha función de comparación, y a partir de la línea 66 se reportan y guardan los resultados en los ficheros de salida, con ciertos comentarios aclaratorios.

El siguiente proceso que necesita ser automatizado es la creación de las propias reglas de diseño de todos los puertos de entrada y salida que se encuentran en el fichero actualizado. Para ello se ha utilizado la información de cada puerto presente en la hoja excel, tal y como se refleja en el figura 6.1.

bst1_trim_iss_vddd[*]	output	85	-2	OSC_CLK	VPOS1
bst2_trim_iss_vddd[*]	output	85	-2	OSC_CLK	VPOS2
bst1_flt_ovp_xtive_n_vddd	input	12	10	OSC_CLK	VPOS1
bst2_flt_ovp_xtive_n_vddd	input	12	10	OSC_CLK	VPOS2
bst1_flt_ocp_xtive_n_vddd	input	12	10	OSC_CLK	VPOS1
bst2_flt_ocp_xtive_n_vddd	input	12	10	OSC_CLK	VPOS2

Figura 6.1: Detalle de la hoja excel automatizada

En columnas adyacentes se han autogenerado las restricciones haciendo uso de las fórmulas propias de excel, tal y como puede verse en la sección de código 6.2.

```
1
2 ="set_" & B144 & "_delay -max " & C144 & " -clock " & E144 & "
      [get_ports " & A144 & "]"
3
4 ="set_" & B64 & "_delay -max " & C64 & " -clock " & E64 & "
      [get_ports " & A64 & "]"
```

Listado de código 6.2: Automatización de restricciones

El resultado de dichas fórmulas puede apreciarse en la figura 6.2.

set_output_delay -min -2 -clock OSC_CLK [get_ports bst1_trim_iss_vddd[*]] -add_delay	set_output_delay -max 85 -clock OSC_CLK [get_ports bst1_trim_iss_vddd[*]]
set_output_delay -min -2 -clock OSC_CLK [get_ports bst2_trim_iss_vddd[*]] -add_delay	set_output_delay -max 85 -clock OSC_CLK [get_ports bst2_trim_iss_vddd[*]]
set_input_delay -min 10 -clock OSC_CLK [get_ports bst1_flt_ovp_xtive_n_vddd] -add_delay	set_input_delay -max 12 -clock OSC_CLK [get_ports bst1_flt_ovp_xtive_n_vddd]
set_input_delay -min 10 -clock OSC_CLK [get_ports bst2_flt_ovp_xtive_n_vddd] -add_delay	set_input_delay -max 12 -clock OSC_CLK [get_ports bst2_flt_ovp_xtive_n_vddd]
set_input_delay -min 10 -clock OSC_CLK [get_ports bst1_flt_ocp_xtive_n_vddd] -add_delay	set_input_delay -max 12 -clock OSC_CLK [get_ports bst1_flt_ocp_xtive_n_vddd]
set_input_delay -min 10 -clock OSC_CLK [get_ports bst2_flt_ocp_xtive_n_vddd] -add_delay	set_input_delay -max 12 -clock OSC_CLK [get_ports bst2_flt_ocp_xtive_n_vddd]

Figura 6.2: Resultado de la hoja excel automatizada

Una vez las reglas de diseño han sido generadas, es necesario parsearlas en el excel y traspasarlas a un archivo con extensión .sdc (Synopsis Design Constraints). Para ello se ha creado el siguiente programa, que recopila y ordena las reglas de diseño de forma que se inserten en el fichero de una forma comprensible y que facilite la detección de errores.

```
1
2  import pandas as pd
3  # Excel file path
4  excel_file = "internal_connectivity.xlsx"
5
6  # Names of the columns you want to parse
7  column1 = "max_delay"
8  column2 = "min_delay"
9  column3 = "max_delay_dqa"
10 column4 = "min_delay_dqa"
11 column_dir = "Dir"
12
13 # Custom header text
14 custom_header1 = "# This is an auto-generated file"
15 custom_header2 = "# Inputs and output constraints"
16
17 # Custom comment text
18 custom_comment = "# End of Inputs and output constraints"
19
20 # Path to save the text file
21 text_file = "utils/digital_top_tcons.sdc"
22
23 # Open the text file in read mode to read its current content
24 with open(text_file, "r") as file:
25     lines = file.readlines()
26
27 # Find the lines between the header and the comment
28 header_start = lines.index(custom_header2 + "\n") + 1
29 comment_end = lines.index(custom_comment + "\n")
30
31 # Delete existing lines between the header and comment
32 del lines[header_start:comment_end]
33
34 # Open the text file in write mode to overwrite it with new data
35 with open(text_file, "w") as file:
36     # Write the custom header
37     file.write(custom_header1)
38     file.write("\n\n")
39     file.write(custom_header2)
40     file.write("\n\n")  # Add two blank lines
41
42     # Read the Excel file
43     df = pd.read_excel(excel_file)
44
45     # Separate data into inputs and outputs for column1 and column2
46     data_column1_input = []
47     data_column1_output = []
48     data_column2_input = []
49     data_column2_output = []
50
51     data_column3_input = []
52     data_column3_output = []
53     data_column4_input = []
```

```
    data_column4_output = []

    for index, row in df.iterrows():
        if row[column_dir] == "input":
            if not pd.isna(row[column1]):
                data_column1_input.append(row[column1])
            if not pd.isna(row[column2]):
                data_column2_input.append(row[column2])
            if not pd.isna(row[column3]):
                data_column3_input.append(row[column3])
            if not pd.isna(row[column4]):
                data_column4_input.append(row[column4])
        elif row[column_dir] == "output":
            if not pd.isna(row[column1]):
                data_column1_output.append(row[column1])
            if not pd.isna(row[column2]):
                data_column2_output.append(row[column2])
            if not pd.isna(row[column3]):
                data_column3_output.append(row[column3])
            if not pd.isna(row[column4]):
                data_column4_output.append(row[column4])

    # for index, row in df.iterrows():
    # if row[column_dir] == "input":
    #     if not pd.isna(row[column3]):
    #         data_column3_input.append(row[column3])
    #     if not pd.isna(row[column4]):
    #         data_column4_input.append(row[column4])
    # elif row[column_dir] == "output":
    #     if not pd.isna(row[column3]):
    #         data_column3_output.append(row[column3])
    #     if not pd.isna(row[column4]):
    #         data_column4_output.append(row[column4])

    #Constraints only for DQA
    file.write("if { $env(TASK) == \"dqa\" } {\n")

    # Write input data for column3
    file.write("#Inputs max delay\n")
    file.write('\n'.join(map(str, data_column3_input)))
    file.write("\n\n")  # Add two blank lines

    # Write input data for column4
    file.write("#Inputs min delay\n")
    file.write('\n'.join(map(str, data_column4_input)))
    file.write("\n\n")  # Add two blank lines

    # Write output data for column3
    file.write("#Outputs max delay\n")
    file.write('\n'.join(map(str, data_column3_output)))
    file.write("\n\n")  # Add two blank lines

    # Write output data for column4
```

```
107     file.write("#Outputs min delay\n")
108     file.write('\n'.join(map(str, data_column4_output)))
109     file.write("\n\n")  # Add two blank lines
110 
111     file.write("} else {\n\n")
112 
113     #Constraints only the rest of tools
114     # Write input data for column1
115     file.write("#Inputs max delay\n")
116     file.write('\n'.join(map(str, data_column1_input)))
117     file.write("\n\n")  # Add two blank lines
118 
119     # Write input data for column2
120     file.write("#Inputs min delay\n")
121     file.write('\n'.join(map(str, data_column2_input)))
122     file.write("\n\n")  # Add two blank lines
123 
124     # Write output data for column1
125     file.write("#Outputs max delay\n")
126     file.write('\n'.join(map(str, data_column1_output)))
127     file.write("\n\n")  # Add two blank lines
128 
129     # Write output data for column2
130     file.write("#Outputs min delay\n")
131     file.write('\n'.join(map(str, data_column2_output)))
132     file.write("\n\n")  # Add two blank lines
133 
134     file.write("}\n")
135 
136     # Write the custom comment
137     file.write(custom_comment)
138     file.write("\n")  # Add one blank line
139 
140 print(f"Data from {column1} and {column2} has been saved in
      {text_file}")
```

Listado de código 6.3: Inserción de constraints

El diseñador encargado de incluir las reglas de diseño de las entradas y salidas, tendrá que mantener y actualizar el excel donde se encuentran todas las restricciones, lo que ayudará a evitar errores, y en caso de tener que realizar nuevas versiones, tan solo será necesario editar el excel y volcar los cambios sobre el archivo .sdc.

Capítulo 7

Conclusiones

A través de las secciones estudiadas en este trabajo fin de máster se ha podido realizar un recorrido exhaustivo a través del proceso de análisis temporal estático, comprendiendo desde las bases del diseño digital, las reglas de diseño introducidas por el diseñador o el comportamiento de la herramienta de análisis, hasta la resolución de violaciones que permitan desarrollar un diseño libre de errores.

Tal y como se ha estudiado a lo largo de este documento, la fabricación de un ASIC se trata de un proceso costoso, por lo que minimizar los errores y futuros cambios de diseño es una parte crucial para las empresas del sector, y los procesos de verificación juegan un papel crucial a la hora de reducir los costes, anticipando problemas y mejorando la calidad del diseño desde el primer momento.

Por lo tanto, considero que este trabajo es de gran utilidad para ingenieros y diseñadores de circuitos digitales, ya que proporciona una guía detallada y práctica sobre cómo llevar a cabo el análisis temporal estático. Al seguir las metodologías y técnicas descritas, los profesionales pueden mejorar significativamente la fiabilidad y el rendimiento de sus diseños de ASIC. Además, el trabajo aporta un conocimiento profundo sobre los chequeos temporales y las reglas de diseño, lo que permite a los diseñadores anticipar y resolver problemas potenciales antes de la fabricación del chip.

Además del conocimiento relativo al STA, se ha propuesto y realizado una mejora del flujo de trabajo. Sin embargo, no es la automatización de reglas de diseño lo más significativo de dicha sección, si no la capacidad de analizar el proceso de diseño e identificar partes que pueden ser mejoradas. De esta forma, los diseñadores deben estar constantemente tratando de mejorar partes del flujo de diseño que han entorpecido de alguna forma el trabajo durante un proyecto, permitiendo así que en futuros diseños, el tiempo y el esfuerzo se centre en lo verdaderamente importante.

En resumen, este trabajo no solo ofrece una comprensión detallada del análisis temporal estático, sino que también proporciona herramientas prácticas y conocimientos aplicables que pueden ser utilizados para mejorar el diseño, la verificación de ASICs y la automatización de las tareas necesarias para su aplicación, mejorando el flujo de diseño y la prevención de errores humanos, ayudando significativamente a los diseñadores durante todo el proyecto. La implementación de las técnicas y metodologías descritas puede llevar a la creación de circuitos más eficientes y fiables, contribuyendo así al avance de la tecnología en el campo de la electrónica digital.

Bibliografía

[1] *MMMC*. Accedido el 2 de septiembre de 2024. URL: https://vlsi.kr/MMMC/.

[2] *Tempus Cadence*. Accedido el 13 de junio de 2024. URL: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/silicon-signoff/tempus-timing-signoff-solution.html.

[3] *Xcelium Cadence*. Accedido el 13 de junio de 2024. URL: https://www.cadence.com/en_US/home/tools/system-design-and-verification/simulation-and-testbench-verification/xcelium-simulator.html.

[4] *Jasper Formal Verification Cadence*. Accedido el 13 de junio de 2024. URL: https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification.html.

[5] *Tempus Cadence*. Accedido el 13 de junio de 2024. URL: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/power-analysis/joules-rtl-power-solution.html.

[6] *Logic Equivalence Check Cadence*. Accedido el 13 de junio de 2024. URL: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/logic-equivalence-checking.html.

[7] Clifford E Cummings. "Clock domain crossing (CDC) design & verification techniques using SystemVerilog". En: *SNUG-2008, Boston* (2008).