

# TRABAJO FINAL



FACULTAD DE INGENIERÍA

Autora:  
Victoria Otegui Alexenicer

Enero 2023

# **Historial de Versiones**

<b>Fecha</b>	<b>Versión</b>	<b>Descripción</b>
17/01/2023	0.1	Inicio
15/02/2023	0.2	Continuación
12/04/2023	0.3	Modificaciones y continuación
9/02/2025	0.3	Continuación del laboratorio
3/07/2025	0.3	Redacción del laboratorio

## **Contenidos**

<b>Historial de Versiones</b>	<b>1</b>
<b>1. Introducción</b>	<b>3</b>
1.1. Motivaciones . . . . .	3
1.2. Alcance . . . . .	3
1.3. Personal Involucrado . . . . .	3
1.4. Definiciones, acrónimos y abreviaturas . . . . .	5
<b>2. Diseño Físico</b>	<b>6</b>
2.1. Introducción al flujo de fabricación de semiconductores . . . . .	6
2.1.1. Introducción . . . . .	6
2.1.2. Proceso de diseño de un CI . . . . .	6
2.1.3. Herramientas EDA . . . . .	8
2.1.4. Tecnología CMOS . . . . .	9
2.1.5. Tecnología FinFet . . . . .	10
2.2. Flujo de trabajo . . . . .	11
2.2.1. Inputs de Place and Route . . . . .	12
2.2.2. <i>Constraints</i> de diseño . . . . .	13
2.2.3. Librerías . . . . .	15
2.2.4. Standard Cells . . . . .	18
2.2.5. Netlist y RTLs . . . . .	22
2.2.6. Condiciones de operación, <i>corners</i> y escenarios . . . . .	23
2.2.7. Síntesis . . . . .	25
2.3. Place and Route . . . . .	26
2.3.1. Floorplanning . . . . .	26
2.3.2. Placement . . . . .	32
2.3.3. Clock Tree Synthesis (Síntesis del Árbol de Reloj) . . . . .	33
2.3.4. Enrutamiento . . . . .	36
2.4. Signoff . . . . .	39
2.4.1. Static Timing Analysis . . . . .	39
<b>3. Laboratorio: Colocacion y enrutamiento de una ALU</b>	<b>42</b>
3.1. Descripción funcional del diseño a implementar . . . . .	42
3.2. Configuración del flujo de diseño . . . . .	44
3.3. Floorplanning . . . . .	45
3.3.1. Integración de las fases <i>design_planning.tcl</i> e <i>init_design.tcl</i> . . . . .	45
3.3.2. Parámetros esenciales del proyecto . . . . .	48
3.3.3. Resultados de la implementación . . . . .	49
3.4. Colocación . . . . .	52
3.5. Síntesis de árbol de reloj . . . . .	57
3.6. Enrutamiento . . . . .	61
3.7. Trabajo Práctico de laboratorio: Cuestionario . . . . .	71

<b>4. Cuestionario de integración final</b>	<b>71</b>
4.1. Enunciado del cuestionario . . . . .	71
4.2. Guía de soluciones y criterios de corrección . . . . .	74
<b>5. Bibliografía</b>	<b>78</b>
5.1. Videos útiles . . . . .	78
5.2. Recursos útiles recomendados (online y de acceso gratuito) . . . . .	78

# 1. Introducción

El propósito de este documento es delimitar, abarcar y explicar el proceso de Diseño Físico de la implementación de una ALU (Unidad Aritmético-Lógica, por sus siglas en inglés: Arithmetic Logic Unit) con el flujo de diseño para un circuito integrado diseñado para una aplicación específica (ASIC, por sus cifras en inglés : Application-Specific Integrated Circuit).

Se tiene por fin obtener el título de grado de fin de la carrera Ingeniería Electrónica de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata.

Destinado al Departamento de Electrónica y a todo estudiante, profesor y directivo involucrado en el desarrollo.

## 1.1. Motivaciones

- Popularizar la metodología de trabajo utilizada en una gran empresa, con tecnología de estado del arte.
- Dar conocimiento en materia de microelectrónica para la industria, para aquellos estudiantes interesados.
- Demostrar experiencia en el campo de la industria y finalizar la carrera de grado, obteniendo el título profesional de Ingeniera Electrónica.

## 1.2. Alcance

En el contexto del presente Proyecto Final, presentando la experiencia en el campo de trabajo de los últimos tres años, el objetivo del proyecto consiste en generar bibliografía que detalle y explique las tareas realizadas en tal área. Esto es, las tareas pertenecientes al Diseño Físico de un diseño ASIC.

Para tal objetivo se realizará una conexión remota al servidor de la Fundación Fulgor, a una máquina con 8 procesadores, en pos de utilizar un versión estudiantil de la herramienta EDA De Synopsys, Fusion Compiler.

El nodo de tecnología para la realización de la Colocación y Enrutamiento es uno genérico de 14 nanómetros, perteneciente a un Kit de Diseño Educacional (EDK, por sus siglas en inglés: Educational Design Kit) llamado SAED\_EDK14\_FINFET.

Para la implementación se utilizará el flujo de Diseño Físico instalado por los estudiantes de la Universidad Nacional de Córdoba Pablo Sosa y David Sosa.

La conexión se realizará a través de X2GO, una solución de escritorio remoto basada en NX que permite acceder a entornos gráficos de Linux de manera eficiente a través de conexiones de red. Utiliza el protocolo SSH para la autenticación y transmisión de datos.

Del proceso de implementación de Diseño Físico se realizará todas las etapas de COlocación y Enrutamiento (PnR, por sus siglas en inglés: Place and Route). Tales etapas corresponden a: *Floorplanning, Placement, Clock Tree Synthesis y Routing*.

Se verificará la correcta implementación a través del análisis de reportes que arroja la herramienta, siendo estos en términos de área, potencia, *timing* y que cumplan las reglas físicas y restricciones (análisis de DRCs, DRVs).

Se dará por exitoso el proyecto si se logra un diseño implementable.

## 1.3. Personal Involucrado

### Integrante

<b>Apellido y Nombre</b>	Otegui Alexenicer, Victoria
<b>Categoría profesional</b>	Estudiante, Ingeniera de Diseño Físico
<b>Responsable de</b>	Desarrollo
<b>Contacto</b>	victoriaotegua@gmail.com

### Director

<b>Apellido y Nombre</b>	Castiñeira Moreira, Jorge
<b>Categoría profesional</b>	Dr. Ingeniero, Categoría I CONICET, Director LAC, Profesor Titular Exclusiva
<b>Responsable de</b>	Dirección
<b>Contacto</b>	casti@fi.mdp.edu.ar

Co-Director

<b>Apellido y Nombre</b>	Uriz, Alejandro José
<b>Categoría profesional</b>	Dr. Ingeniero, Profesor Asociado Exclusiva, Investigador Adjunto CONICET
<b>Responsable de</b>	Co-Dirección
<b>Contacto</b>	ajuriz@fimdp.edu.ar

## 1.4. Definiciones, acrónimos y abreviaturas

Nombre	Descripción
VLSI	Very Large Scale Integration
RTL	Register Transfer Level
SoC	System-on-a-chip.
ASIC	Application specific integrated circuit
EDA	Electronic Design Automation
PnR	Place and Route
IC o CI	Integrated Circuit
Foundries	Industrias que llevan a cabo la fabricación
PD	Physical Design
STA	Static Timing Analysis
FE	Front-End
BE	Back-End
DSP	Digital Signal Processor
CMOS	Complementary metal-oxide semiconductor
SCE	Short channel effect.
SDC	Synopsis Design Constraints
DRC	Design Rule Checking
Layout	arreglo físico de transistores, cables y otros componentes
RAM	Random Access Memory
ROM	Read-Only Memory
PLL	Phase-Locked Loop
ADC	Analog-to-digital converter
DAC	Digital-to-analog converter
VR	Voltage regulator
IP	Intellectual Property (Propiedad Intelectual)
Wafer	Placa de silicio dentro de la cual se fabrican los IC
Die	cada chip dentro del wafer antes del encapsulado
Drive Strength (DS)	Fuerza de conducción
Optical Process Correction	OPC
LEF	Library Exchange Format
NDM	Non-Default Model
LIB	archivo de librería Liberty
PDK	process design kit
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
top level	refiere al nivel más alto de abstracción en la jerarquía de diseño
HDL	Hardware Description Language
PG	PowerGrid ; red de distribución de energía eléctrica en el chip
DSP	(Digital Signals Processor) Procesamiento de Señales Digitales
GUI	Graphic User Interface
RDL	Redistribution Layer (Capa de Redistribución)
PG	Power Grid
CTS	Clock Tree Synthesis (Síntesis del árbol de reloj)
GRC	Celdas de Enrutamiento Global (Global Routing Cells)
SSH	Secure Shell
MCMM	Multi-Corner, Multi-Mode. Todos los modos y corners.
.ndm	Archivo binario. Modelo nativo de diseño (lógica, geometría y restricciones) para FC.
NDR	Non-default rule: reglas custom para el ruteo
ECO	Orden de cambio de Ingeniería ( <i>Engineering Change Order</i> )
SI	Integridad de señal ( <i>signal integrity</i> )
TNS	Total Negative Slack (retardo negativo total acumulado)
WNS	Worst Negative Slack (peor retardo negativo)
VT	Voltage Treshold (Umbral de voltaje)
DFT	Design for Testing

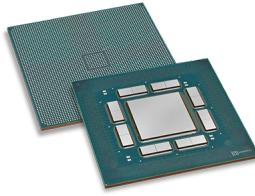


Figura 1: Chip ASIC

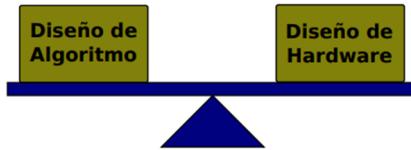


Figura 2: Balance para los requisitos de desempeño

## 2. Diseño Físico

### 2.1. Introducción al flujo de fabricación de semiconductores

#### 2.1.1. Introducción

Para 1965, la tercera generación de dispositivos electrónicos comenzó con la invención del Circuito Integrado (IC). El IC comenzó a reemplazar circuitos con transistores discretos. Esto resultó en una reducción sustancial del tamaño físico y costo de los sistemas. Esta generación impulsó la rápida integración de las formas de diseño de circuitos (pequeña, mediana y grande) a dispositivos muy grandes que contenían millones de transistores.

Estos logros fueron posibles por el desarrollo de equipamiento de procesamiento de IC, herramientas de diseño y software. En los últimos veinte años, el mundo fue testigo de una rápida reducción del tamaño en las características de los transistores (de centenas a unidades de nanómetros), tanto como de la espectacular innovación de sofisticadas herramientas de automatización de *diseño físico*.

La complejidad del diseño físico de ASICs hoy en día requiere un mix de *backgrounds* en ingeniería electrónica, ciencia de la computación y procesos de IC. Tal conocimiento diversificado ha creado una nueva disciplina en la ingeniería - el ingeniero de diseño físico, de quien se espera esté familiarizado con todos los aspectos de las etapas de implementación de diseños ASIC que incluyen: los procesos de dispositivos, *development* de librerías, algoritmos de place-and-route y verificación.

#### 2.1.2. Proceso de diseño de un CI

Un ASIC es un circuito integrado creado específicamente para resolver una aplicación de cálculo precisa. Está construido para un cómputo específico. Posee un ciclo de planificación complejo y largo, con costos iniciales altos y posteriormente costos por unidad bajos. Ocupan poco espacio y disipan poca potencia, con frecuencia de funcionamiento altas. En contraposición a un diseño en FPGA, no se puede modificar y el desempeño es el enfoque principal.

El diseño de los algoritmos para estas aplicaciones y su implementación en VLSI siguen un proceso de evolución paralela cuyos caminos se ven influenciados por:

- Los diseñadores de algoritmos: enfatizan la mejora del rendimiento.
- Los diseñadores de *hardware*: buscan flexibilidad en la implementación de acuerdo a la tecnología empleada.

Estos objetivos complementarios resultan en un largo proceso iterativo entre ambos equipos de diseño para poder converger a una arquitectura implementable y que cumpla con los requisitos de desempeño. Los objetivos críticos se delimitan por

1. Área ocupada por compuertas lógicas (*logic gates*) e interconexiones.
2. El retardo del camino crítico (*critical path*) de la ruta más larga a través de la lógica.



Figura 3: Relación entre impacto del diseño y complejidad

3. Disipación de potencia de las compuertas lógicas.
4. El grado de capacidad de prueba (*testability*) del circuito, medida en términos del porcentaje de las fallas cubiertas por un conjunto específico de vectores de pruebas para un modelo de falla apropiado.

En general, el diseño de un IC incorpora dos etapas principales que se definen como Diseño **Front-End** y Diseño **Back-End**.

El Diseño Front-End (FE) incluye el estudio del mercado, de tecnología, la revisión de requerimientos, el diseño de arquitectura y el diseño digital mediante un lenguaje de descripción de hardware (por ejemplo, Verilog, System Verilog o VHDL). Además, esta etapa abarca la verificación del diseño mediante simulación y otras técnicas de verificación. A continuación, se detallan los ítems:

1. Especificación del problema (Sistemas): Se trata de una interpretación de alto nivel de un sistema. Se abordan los parámetros clave, como las técnicas de diseño, la funcionalidad, el rendimiento, la tecnología de fabricación y las dimensiones físicas. Los requerimientos de los clientes y del mercado (máxima tasa de datos en el transmisor, la tasa de error de bit, ancho de banda del canal, frecuencia de portadora e intermedias, límite de potencia, interfaces). Además, las especificaciones finales incluyen la potencia, la funcionalidad, la velocidad y el tamaño del sistema VLSI.
2. Definición de la arquitectura (Sistemas): Se buscan algoritmos que implementen la funcionalidad deseada, los desarrolladores toman estos requisitos y exploran las técnicas de comunicación digitales que pueden cumplir con las especificaciones indicadas. Se llevan a pruebas en simuladores elementales de Matlab. Además, se modela el canal de comunicaciones (fibra óptica) y los componentes electro-ópticos externos a nuestro chip (amplificadores, modulares, etc.). Se utiliza el criterio de operaciones en punto flotante.
3. Diseño funcional (RTL/DSP/FEC): Reconoce las unidades funcionales vitales de un sistema. Se llevan los algoritmos que definen la arquitectura a un diseño en hdl básico, buscando las mejores opciones para potencia, área, temporización. Se itera sobre este diseño implementando junto con los diseñadores de arquitectura.
4. Optimización del diseño funcional: Se enfoca en el Register Transfer Level (RTL), en la cuantización de los algoritmos, optimizaciones de bajo nivel, asignación de registros y la verificación. Se entrega a los diseñadores de sistema un simulador de punto fijo que representa el RTL.

El diseño Back-End (BE) consiste en la caracterización y el diseño de la biblioteca CMOS. Además, incluye la simulación de fallos y el diseño físico.

1. Diseño del circuito y síntesis: Este paso lleva a cabo la realización del circuito en forma de netlist (conexiones a nivel de compuertas). Utiliza la simulación para comprobar el resultado.
2. **Diseño Físico:** En este paso, se crea el diseño convirtiendo el netlist en una representación geométrica. Otros pasos que corresponden al backend del IC pero que se realizan **posterior** al Diseño Físico:

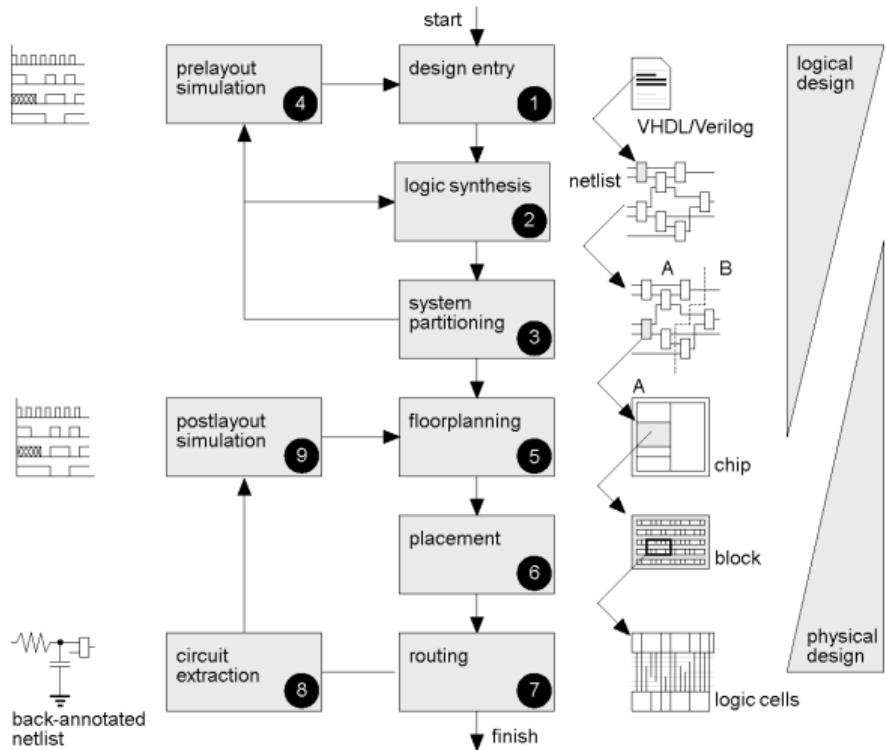


Figura 4: Diseño de flujo VLSI

3. Procesamiento de obleas: En este paso se utiliza silicio puro fundido en una olla a 1400°C. A continuación, se inyecta en el silicio licuado una pequeña semilla que comprende la orientación del cristal requerida y se extrae gradualmente, 1mm por minuto. Se fabrica el cristal de silicio como un lingote cilíndrico y se corta en discos u obleas antes de pulirlo y orientarlo.
4. Litografía: Este proceso (fotolitografía) incluye el enmascaramiento con fotograbado y una máscara fotográfica. A continuación, se aplica una película fotorresistente sobre la oblea. A continuación, un alineador fotográfico alinea la oblea con una máscara. Por último, se expone la oblea a la luz ultravioleta, lo que permite resaltar las huellas a través de la máscara.
5. Grabado: aquí eliminamos selectivamente material de la superficie de la oblea para producir patrones. Con una máscara de grabado para proteger las partes esenciales del material, se utiliza plasma o productos químicos para eliminar los restos de fotorresistencia.
6. Implementación de iones: Aquí se utiliza un método para conseguir una característica eléctrica deseada en el semiconductor, es decir, un proceso de adición de dopantes. El proceso utiliza un haz de iones dopantes de alta energía para dirigirlo a zonas precisas de la oblea. El nivel de energía del haz determina la profundidad de la penetración en la oblea.
7. Metalización: En este paso se aplica una fina capa de aluminio sobre toda la oblea.
8. Montaje y embalaje: Cada una de las obleas contiene cientos de chips. Por ello, se utiliza una sierra de diamante para cortar las obleas en chips individuales. Despues, se someten a pruebas eléctricas y se descartan los chips fallidos. Los que pasan la prueba son sometidos a una inspección visual exhaustiva con un microscopio. Por último, se empaquetan los chips que superan la inspección visual y se vuelven a comprobar.

### 2.1.3. Herramientas EDA

EDA (Electronic Design Atomation) es un segmento el mercado que consiste de software y servicios con el objetivo colectivo de asistir en la definición, planificación, diseño, implementación, verificación y la subsequente manufacturación de dispositivos semiconductores o chips. En cuanto a la manufacturación de estos

dispositivos, los principales proveedores de este servicio son las *foundries* de semiconductores o fábricas. Estas instalaciones, sumamente complejas y costosas, son propiedad de grandes empresas de semiconductores integradas verticalmente.

Periodo de Tiempo	Avances en el Diseño de Circuitos y Diseño Físico
1950 - 1965	Diseño manual únicamente.
1965 - 1975	Editores de diseño, por ejemplo, herramientas de colocación y ruteo, desarrolladas por primera vez para placas de circuitos impresos.
1975 - 1985	Herramientas más avanzadas para CI y PCB, con algoritmos más sofisticados.
1985 - 1990	Primeras herramientas basadas en rendimiento y algoritmos de optimización paralela para el diseño; mejor comprensión de la teoría subyacente (teoría de grafos, complejidad de la solución, etc.).
1990 - 2000	Primer enrutamiento sobre celdas, primeras técnicas de colocación y enrutamiento en 3D y multicapa. La síntesis automática de circuitos y el diseño orientado a la rutaabilidad se vuelven dominantes. Inicio de la paralelización de cargas de trabajo. Surgimiento de la síntesis física.
2000 - presente	Diseño para la Fabricabilidad (DFM), corrección de proximidad óptica (OPC) y otras técnicas emergen en la interfaz de diseño-fabricación. Mayor reutilización de bloques, incluidos bloques de propiedad intelectual (IP).

Cuadro 1: Evolución de los Avances en el Diseño de Circuitos y Diseño Físico

La industria EDA empezó hace más de 40 años y desde entonces se encarga de proveer todas las herramientas de software que permiten la creación y simulación de las diferentes etapas de un circuito integrado, así también como la optimización de los mismos para obtener la mejor relación costo/producto.

Otro segmento de mercado que está estrechamente relacionado con la EDA es la propiedad intelectual de los semiconductores o *IP* (Propiedad Intelectual, por sus siglas en inglés: *Intellectual Property*). Este segmento de mercado proporciona circuitos prediseñados de complejidad variable que pueden utilizarse tal cual o adaptarse para una aplicación particular. Los *IP* permite que los chips altamente complejos se diseñen en mucho menos tiempo ya que se pueden reutilizar muchos trabajos de diseño existentes. Debido a la fuerte dependencia del uso y reuso de las *IP* en las herramientas de la EDA, estos mercados suelen considerarse como uno.

#### 2.1.4. Tecnología CMOS

En la evolución de IC, si bien el MOS (*Metal-oxide-silicon*) fue inventada antes que el transistor bipolar, inicialmente fue difícil de fabricar. La tecnología nMOS (n-channel MOS) fue desarrollada en 1970, requería menos pasos de enmascaramiento, era más denso y consumía menos potencia que los ICs bipolares equivalentes, por lo tanto un IC MOS era más barato que un IC bipolar y llevó a una inversión y crecimiento en el mercado IC MOS.

Los *gates* de aluminio fueron reemplazados por polisilicio a principio de 1980. El CMOS (*Complementary MOS*) consiste de un transistor MOS n-channel y uno MOS p-channel, que deriva en menor consumo de potencia y un proceso de fabricación más simple.

A continuación, en la Figura 5 y Figura 6, se muestra la representación de los transistores CMOS.

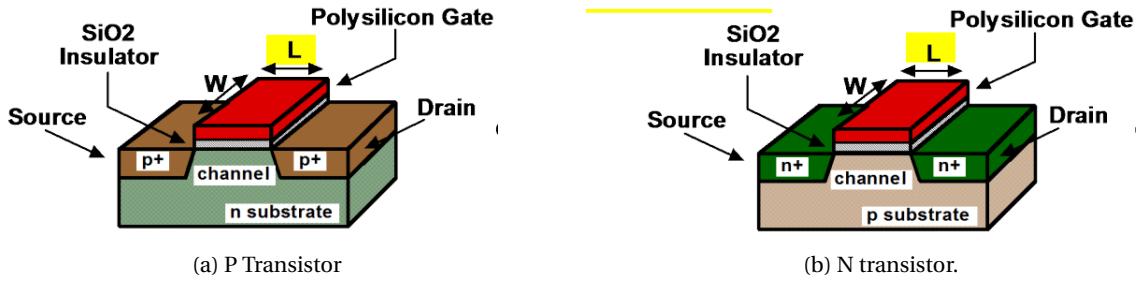


Figura 5: Transistores CMOS.

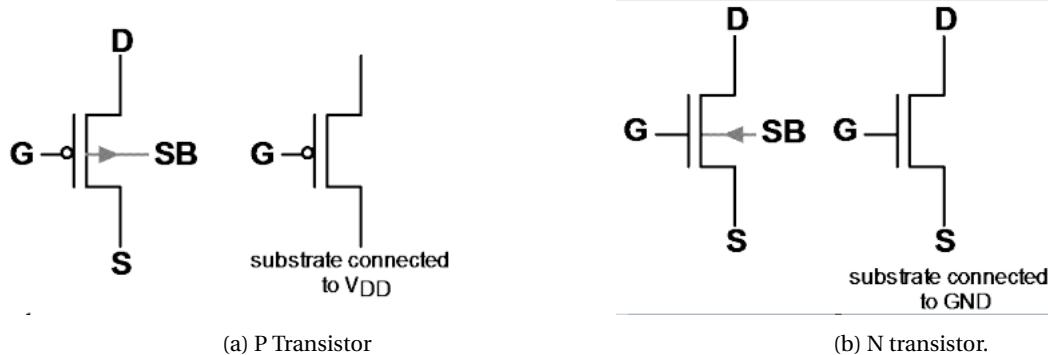


Figura 6: Esquemático de transistores CMOS.

La característica clave es la **Longitud L**. El escalamiento de los transistores MOS se da para incrementar la densidad de embalaje del dispositivo, para mejorar la frecuencia de respuesta (tiempo transitorio)  $\alpha \frac{1}{L}$  y para mejorar la transconductancia. La disminución de la longitud del canal y el espesor del óxido de puerta aumenta  $g_m$ , es decir, la corriente de conducción del transistor. Gran parte de la escala es por lo tanto impulsado por la disminución de L y  $t_{ox}$ . Sin embargo, si solo estos dos parámetros se escalan muchos problemas se encuentran, por ejemplo, aumento del campo eléctrico.

Los transistores están construidos sobre un sustrato de silicio, que es un material del Grupo IV y forma una red cristalina con enlaces a cuatro vecinos.

Si bien el silicio es un semiconductor, el puro no tiene portadores libre y conduce pobremente. El agregado de dopantes incrementa la conductividad, generando semiconductores *p-type* (electrón extra) y *n-type* (electrón faltante).

En los procesos modernos, el voltaje de alimentación  $V_{DD}$  ha decrecido, dado que uno alto dañaría los diminutos transistores modernos. Además, un bajo  $V_{DD}$  reduce el consumo de potencia, que es el enfoque actual de diseño y fabricación.

El diseño de MOS tiene como objetivo convertir una especificación en máscaras para procesar silicio para cumplir con la especificación. Los circuitos MOS se forman en cuatro capas básicas:

- *N-diffusion*
- *P-diffusion*
- Poly Si
- Metal

que se aíslan entre sí por capas aislantes de dióxido de silicio gruesas o delgadas (*thinox*). La región de la máscara de óxido delgado incluye las *n-diffusion*, *p-diffusion* y los canales de transistor. Las regiones de polisilicio y *thinox* interactúan para que se forme un transistor donde se crucen entre sí.

### 2.1.5. Tecnología FinFet

A medida que la tecnología de semiconductores avanza, el tamaño de los transistores se reduce constantemente. En los MOSFET planos tradicionales, como el que se muestra en la Figura 7(a), el canal es prácticamente “horizontal” y el *gate* controla la corriente sólo desde la parte superior. Cuando se intenta escalar estos

dispositivos por debajo de los 20 nm, ese control del *gate* sobre el canal se debilita: aparecen los llamados efectos de canal corto (*short channel effects*, SCE), aumenta la fuga de corriente y el comportamiento del transistor se vuelve menos predecible.

Para mejorar este control electrostático, la industria migró hacia dispositivos *multi-gate* o de múltiples compuertas, entre ellos el FinFET. En la Figura 7(b) se observa que el canal ya no es un plano, sino una aleta vertical (*FIN*) alrededor de la cual se envuelve el *gate*. De esta forma, el *gate* “abraza” mejor al canal y puede controlar la carga desde más de un lado, lo que reduce las corrientes parásitas y hace al dispositivo más robusto frente a variaciones de dopaje y de proceso.

En los FinFET, el ancho efectivo del canal ya no se define por la distancia lateral entre *source* y *drain*, sino por la geometría del *FIN*. Por eso se habla de *altura del FIN* ( $H_{FIN}$ ) y del número de aletas en paralelo para cuantificar el ancho de canal. Un FinFET puede aumentar la corriente (y por lo tanto la capacidad de conducción) incrementando el número de aletas o la altura del FIN, como se sugiere en la Figura 7(b). Al mismo tiempo, una altura de FIN adecuada ayuda a mejorar la estabilidad mecánica y eléctrica del dispositivo: un FIN demasiado bajo reduce el ancho efectivo del canal, y uno excesivamente alto puede resultar más difícil de fabricar y menos robusto.

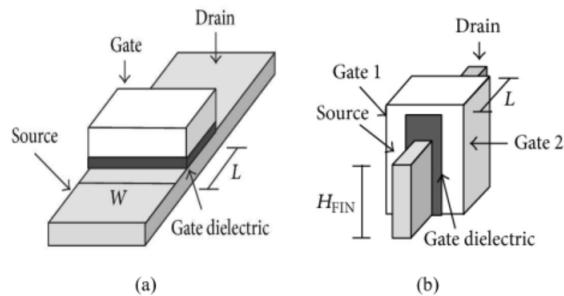


Figura 7: Comparación entre (a) MOSFET plano y (b) FinFET

Los FinFETs pueden ser implementados en un wafer (oblea) en donde todas las aletas comparten un sustrato común.

La transición de una tecnología plana a una FinFET causa varios retos en el proceso de manufacturación:

- Forma de la aleta (*fin*): por ejemplo, la inclinación en la altura de la aleta causa la falta de inmunidad para SCE, además de serios problemas a nivel de escalamiento de longitud de *gate*.
- Capacitancias parásitas.

El **nodo de tecnología actual** para el diseño de ASIC es de alrededor de 14 nanómetros (nm) o menos.

En los últimos años, la industria de semiconductores ha avanzado rápidamente, y los diseños ASIC de vanguardia ahora utilizan nodos de tecnología tan pequeños como 5nm. Estos nodos avanzados ofrecen mejoras significativas en rendimiento, consumo de energía y densidad en comparación con nodos más antiguos. Sin embargo, el diseño de ASIC utilizando nodos de tecnología avanzada puede ser un desafío debido a la complejidad y variabilidad en la fabricación.

En resumen, en las primeras generaciones de tecnología CMOS, el parámetro clave del transistor era la longitud de canal  $L$ , y el “nodo de tecnología” (por ejemplo, 180 nm, 90 nm) se asociaba de forma bastante directa con esa dimensión mínima: al reducir  $L$  y el espesor del óxido de puerta se lograban transistores más rápidos, con mayor transconductancia y una mayor densidad de integración. Sin embargo, al seguir escalando únicamente  $L$  comenzaron a aparecer problemas importantes, como campos eléctricos muy elevados y fuertes efectos de canal corto, lo que obligó a introducir nuevas arquitecturas de transistor, como los FinFET de compuerta envolvente, para recuperar el control electrostático del canal. En los nodos avanzados (7 nm, 5 nm, etc.), el número en nanómetros ya no corresponde de manera directa a la longitud física del canal, sino que funciona como un indicador de “generación tecnológica” ligado a varios parámetros de escala (por ejemplo, el *gate pitch* y el *metal pitch*), reflejando que se trata de una familia de transistores FinFET más densos, más eficientes y con mejores prestaciones que la generación anterior.

## 2.2. Flujo de trabajo

Dentro de la etapa de diseño físico podemos hacer una división entre varias subetapas o tareas que deben realizarse como parte del flujo de PD: Place and Route y Signoff.

Este escrito se enfocará en la etapa de implementación por se, el *Place and Route*.

Se utiliza el flujo de diseño instalado por estudiantes de la UNC, en Linux CentOS 7, con la versión de la EDA *Fusion Compiler* para un nodo de tecnología de 14 nm.

### 2.2.1. Inputs de Place and Route

#### Inputs de Place and Route

Dentro de los archivos que son necesarios como entradas para el diseño físico, también podemos categorizar algunos subgrupos:

- Archivos proporcionados por Front-End (FE): contemplan el RTL y los *constraints* (por ejemplo, archivos SDC de temporización e *IO delays*).
- Archivos de tecnología (.tf): contemplan archivos con reglas de diseño físico a cumplir, archivos de caracterización de parásitos, etc.
- Librerías de *standard cells*: provistas por la fundidora, son caracterizaciones físicas y de *timing* de compuertas (AND, OR, NOR, NAND, flops D, etc.) que la herramienta podrá usar para cumplir con los requerimientos del diseño. Contienen información física. En la Figura 8 se muestra un ejemplo de la descripción de una celda estándar con funcionalidad de inversor de la librería provista por el EDK, extraído de la bibliografía *SAED 14nm FinFET Educational Design Kit*.
- Macros e IPs físicas (memorias SRAM/ROM, PLLs, bloques analógicos, etc.): suelen venir como vistas abstractas LEF más modelos de *timing/potencia* (LIB/NDM) que se integran al mismo flujo de *Place & Route*.
- Modelos de interconexión (archivos TLU+/ITF): describen la resistencia y capacitancia de las capas de metal y se utilizan para la extracción de parásitos y el análisis de temporización post-routing.
- Librería de IO y *pad cells*: celdas de entrada/salida, pads de alimentación y ESD, junto con *corner pads*, necesarias típicamente en el *top level* de un chip para definir el anillo de pads y la interfaz con el exterior. **En este trabajo, al centrarnos en el diseño físico de un bloque interno (la ALU de 32 bits) y no en el chip completo, estas celdas de IO no se incluyen en el flujo de Place & Route.**

## General Information

Cell Name	TT SPICE model, VDD=0.8V, Temperature=25				Area	
	Cload	Prop Delay (Avg)	Power			
			ps	nW		
CLKSPLT_1	1 x Csl	34	1.02	0.56	1.1544	
CLKSPLT_8	8 x Csl	39	2.19	0.63	1.6428	

**INV\_\***

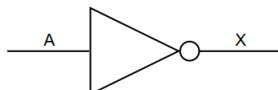


Figure 2.20. Logic Symbol of INV\_\*

Table 2.30. INV\_\*Truth Table

A	X
0	1
1	0

## General Information

Cell Name	TT SPICE model, VDD=0.8V, Temperature=25				Area	
	Cload	Prop Delay (Avg)	Power			
			ps	nW		
INV_OP5	0.5 x Csl	33	0.05	0.04	0.1776	
INV_OP75	0.75 x Csl	33	0.08	0.05	0.1776	
INV_1	1 x Csl	32	0.1	0.07	0.1776	
INV_10	10 x Csl	32	1.01	0.73	0.5772	
INV_12	12 x Csl	32	1.21	0.88	0.666	

Figura 8: Ejemplo de una celda con funcionalidad de inversor para el EDK utilizado

### 2.2.2. Constraints de diseño

Los *design constraints* (restricciones de diseño) son especificaciones que se aplican durante la síntesis lógica y física para guiar a las herramientas EDA hacia una implementación que cumpla los objetivos de rendimiento y de fabricación. En términos generales, pueden agruparse en dos grandes familias:

- **Timing constraints:** restricciones relacionadas con la temporización, sincronización de relojes y retardos máximos/mínimos aceptables en las rutas.
- **Design rule constraints:** restricciones vinculadas a reglas físicas y de carga, por ejemplo, máximas transiciones (*slew*), capacitancias de carga, fanout, etc.

Las restricciones de temporización son provistas por el usuario (o por el equipo de Front-End) y definen el entorno de *timing* en el que se debe analizar el diseño: frecuencia de operación, retardos en las interfaces, rutas que no deben analizarse (*false paths*), rutas de múltiples ciclos, entre otras. Estas *timing constraints* están directamente relacionadas con el rendimiento que se busca alcanzar.

El lenguaje SDC (*Synopsys Design Constraints*) es un formato basado en Tcl utilizado para describir estas restricciones de diseño en herramientas de síntesis, implementación y análisis estático de tiempos (STA). Se emplea ampliamente en herramientas como Synopsys Design Compiler, Fusion Compiler, Xilinx Vivado e Intel Quartus. Un archivo .sdc (o un conjunto de scripts Tcl equivalentes) suele incluir comandos para:

- Definir relojes del sistema y sus retardos asociados.
- Describir rutas de múltiples ciclos (*multicycle paths*).
- Especificar retardos de entrada y salida (*input/output delays*).

- Imponer retardos máximos y mínimos en determinadas rutas.
- Definir capacitancias de carga y límites de transición en las señales.
- Declarar rutas falsas (*false paths*) que no deben verificarse en STA.

En este trabajo, las restricciones de tiempo se implementan mediante varios scripts Tcl. A modo de ejemplo, el archivo `timing.tcl` se utiliza para declarar caminos falsos asociados a señales estáticas o de configuración (*mode pins*) y al *reset* asincrónico:

```

1 # Definicion de restricciones de tiempo - timing.tcl
2 # Autora: Victoria Otegui Alexenicer
3 # Version: 0.1
4 # Fecha 9/12/2024
5
6 puts "INFO: Configurando false paths para senales estaticas o reset asincrono"
7 set_false_path -through [get_ports i_reset]
8 set_false_path -through [get_ports i_operation]
9 set_false_path -through [get_ports i_sel_clock]
10 set_false_path -through [get_ports i_enable]
```

La definición de los relojes del bloque se realiza en el archivo `clocks.tcl`. En él se definen el reloj principal, relojes virtuales para la caracterización de *input/output delays*, y los relojes generados (divisores de frecuencia y salida del multiplexor de reloj), además de agruparlos lógicamente para evitar el análisis de combinaciones no físicas:

```

1 # Definicion de clock(s)
2 # Autora: Victoria Otegui Alexenicer
3 # Version: 0.4
4 # Fecha 17/12/2024
5 # INFO: aplicando mapeado de celdas de fusion para definir los puertos de los clocks generados
6
7 set CLK_NAME      "CLOCK_MAIN"
8 set CLK_PORT      "i_clock"
9 set CLK_PERIOD    "10"; # ns
10
11 # Defino los clocks segun SDC (synopsys design constraints)
12 puts "INFO: Definiendo clocks"
13 create_clock [get_ports $CLK_PORT] -name $CLK_NAME -period $CLK_PERIOD \
14   -waveform "0 [expr 0.5*$CLK_PERIOD]"
15
16 puts "INFO: Definiendo clocks virtuales para la definicion de input y output delays"
17 create_clock -name vir_${CLK_NAME} -period $CLK_PERIOD
18
19 puts "INFO: Definiendo clocks generados"
20 # Clock dividido por 2 (q_div_2)
21 create_generated_clock -name gen_clk_div2 -divide_by 2 \
22   -source [get_pins {q_div_2*/clocked_on}] -master_clock CLOCK_MAIN \
23   [get_pins {q_div_2*/Q}] -add
24 # Clock dividido por 4 (q_div_4)
25 create_generated_clock -name gen_clk_div4 -divide_by 2 \
26   -source [get_pins {q_div_4*/clocked_on}] -master_clock gen_clk_div2 \
27   [get_pins {q_div_4*/Q}] -add
28 # Clock dividido por 8 (q_div_8)
29 create_generated_clock -name gen_clk_div8 -divide_by 2 \
30   -source [get_pins {q_div_8*/clocked_on}] -master_clock gen_clk_div4 \
31   [get_pins {q_div_8*/Q}] -add
32
33 # Clock seleccionado en la salida del multiplexor de reloj (o_clock_w)
34 create_generated_clock -name o_clock_w -source [get_ports i_clock] \
35   [get_pins u_clock_mux/clk_out]
36
37 # Clocks generados para el MUX
38 create_generated_clock -name gen_clk_CLOCK_MAIN_MUX -divide_by 1 \
39   -source [get_pins {u_clock_mux/I0}] -master_clock CLOCK_MAIN \
40   [get_pins {u_clock_mux/Z}] -add
41 create_generated_clock -name gen_clk_DIV2_CLK_MUX -divide_by 1 \
42   -source [get_pins {u_clock_mux/I1}] -master_clock gen_clk_div2
```

```

43 [get_pins {u_clock_mux/Z}] -add
44 create_generated_clock -name gen_clk_DIV4_CLK_MUX -divide_by 1 \
45   -source [get_pins {u_clock_mux/I2}] -master_clock gen_clk_div4 \
46   [get_pins {u_clock_mux/Z}] -add
47 create_generated_clock -name gen_clk_DIV8_CLK_MUX -divide_by 1 \
48   -source [get_pins {u_clock_mux/I3}] -master_clock gen_clk_div8 \
49   [get_pins {u_clock_mux/Z}] -add
50
51 # Configuracion de los margenes (uncertainties)
52 set_clock_uncertainty -setup 0.2 [get_clocks i_clock]
53 set_clock_uncertainty -hold 0.1 [get_clocks i_clock]
54
55 # Grupos logicamente y fisicamente exclusivos
56 set_clock_groups -group {CLOCK_MAIN} -group {gen_clk_div2} \
57   -group {gen_clk_div4} -group {gen_clk_div8} \
58   -logically_exclusive
59 set_clock_groups -group {gen_clk_CLOCK_MAIN_MUX} -group {gen_clk_DIV2_CLK_MUX} \
60   -group {gen_clk_DIV4_CLK_MUX} -group {gen_clk_DIV8_CLK_MUX} \
61   -physically_exclusive

```

Por otra parte, se definen las restricciones de *input delay* y *output delay*, que presupuestan el tiempo consumido por la lógica externa al bloque. El *input delay* reserva el margen comprendido entre el instante en que la señal abandona el bloque precedente (o la PCB) y el primer flanco útil de reloj en el bloque actual; de este modo, la herramienta “ve” menos tiempo disponible y garantiza temporización aun considerando el retardo externo. El *output delay* reserva el margen necesario para que la señal, una vez sale del bloque, se propague a través de trazas, buffers o dispositivos hasta ser capturada por el siguiente sistema. Así, los retardos de entorno quedan presupuestados y cada IP o sub-bloque puede diseñarse y verificarse de forma independiente, asegurando que la ruta extremo a extremo cumplirá al integrarse.

En este proyecto, dichos retardos se describen en el archivo `iodelay.tcl`:

```

1 # Definicion de restricciones de tiempo - iodelay.tcl
2 # Autora: Victoria Otegui Alexenicer
3 # Version: 0.1
4 # Fecha 9/12/2024
5
6 # -- Input delays --
7 set_input_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] i_data_a
8 set_input_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] i_data_b
9 set_input_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] i_operati
10 set_input_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] i_sel_clo
11 set_input_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] i_enable
12 set_input_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] i_valid
13 set_input_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] i_reset
14
15 # -- Output delays --
16 set_output_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] o_data
17 set_output_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] o_clock
18 set_output_delay -add_delay 2.5 [get_clocks vir_CLOCK_MAIN] o_valid

```

### 2.2.3. Librerías

El diseño físico de un ASIC necesita el uso de numerosos *sets* de datos o librerías. Las librerías son una de las partes más críticas del flujo: la precisión de sus modelos y vistas (tanto lógicas como físicas y de temporización/potencia) tiene un impacto directo en la calidad del diseño implementado y, en última instancia, en el éxito del ASIC fabricado.

En términos generales, una librería agrupa colecciones de celdas predefinidas que se utilizan como bloques de construcción del circuito. Estas celdas incluyen puertas lógicas, *flip-flops*, multiplexores, buffers, inversores, así como celdas *physical only* (*taps*, *endcaps*, *fillers*, *decaps*, etc.). También es habitual disponer de:

- librerías de *standard cells* para la lógica general;
- librerías de pads de entrada/salida (IO pads) y celdas de ESD, usadas a nivel *top*;

- macros e IPs físicas (SRAM/ROM, PLLs, bloques analógicos, etc.) con sus vistas correspondientes.

Estas librerías proporcionan bloques de construcción ya optimizados y verificados que el diseñador puede reutilizar para ensamblar el circuito de forma eficiente. Esto no sólo acelera el proceso de diseño, sino que además garantiza que los componentes utilizados cumplen los requisitos de rendimiento, consumo de potencia y reglas de diseño de la tecnología. En la práctica, parte de la estrategia de optimización de potencia a nivel de chip se apoya precisamente en las librerías (mezcla de VTs, distintas *drive strengths*, celdas de baja fuga, etc.).

En este trabajo, al centrarse en el diseño físico de un **bloque interno** (la ALU de 32 bits) y no en el chip completo, se utilizan principalmente librerías de *standard cells* y celdas *physical only*. La librería de IO y de pads de borde (*IO pad ring*) no se incluye en el flujo, ya que su integración corresponde al *top level* del circuito.

Cada celda de la librería ofrece múltiples vistas que permiten su utilización en distintas etapas del flujo de diseño. Los formatos más habituales son:

- **.lib (Liberty)**: describe el comportamiento lógico y de temporización de las celdas, incluyendo retardos de propagación, tiempos de transición, requisitos de *setup/hold*, así como modelos de consumo dinámico y estático. Es fundamental para la síntesis lógica y el análisis de temporización estática (STA). El archivo **.lib** es una representación ASCII de estos parámetros para un nodo tecnológico y un *corner* determinado.
- **.db**: versión binarizada del **.lib**, optimizada para herramientas Synopsys, que acelera la carga y el análisis de librerías.
- **.lef (Library Exchange Format)**: provee una descripción física abstracta de las celdas: dimensiones, posición de pines, obstrucciones (*obstructions*) y reglas de colocación. Se utiliza en las etapas de *placement* y *routing*.
- **.gds (GDSII)**: contiene el *layout* completo y detallado de las celdas, a nivel de capas de proceso. Se emplea en las etapas finales del flujo, en verificación de reglas de diseño (DRC/LVS) y para la generación del *tape-out*.
- **.v**: describe el modelo funcional en Verilog de cada celda. Es utilizado en simulación funcional y en verificación lógica.
- nombre de la librería y de la tecnología;
- unidades de tiempo, potencia, voltaje, corriente, resistencia y capacitancia;
- condiciones de operación (proceso, voltaje y temperatura) para los distintos *corners* (Máximo, Mínimo y Típico).

En la Figura 9 se muestra la cabecera típica de un archivo **.lib** del nodo SAED14, donde se declaran la tecnología, las unidades y las condiciones de operación (corners PVT) de la librería.

---

```

library (SAEDRVT14_LIB)
  technology (cmos);

  delay_model          : table_lookup;
  date                : "2020/01/01";
  revision            : 1.0;
  time_unit           : "1ns";
  voltage_unit        : "1V";
  current_unit        : "1uA";
  leakage_power_unit  : "1nW";
  pulling_resistance_unit : "1kohm";
  capacitive_load_unit(1.0, ff);

  nom_voltage         : 0.80;
  nom_temperature     : 25.0;
  nom_process          : 1.0;

  operating_conditions("TYPICAL") {
    process             : 1.0;
    voltage              : 0.80;
    temperature          : 25.0;
  }

  operating_conditions("SS_0p72V_125C") {
    process             : 1.0;
    voltage              : 0.72;
    temperature          : 125.0;
  }

  operating_conditions("FF_0p88V_m40C") {
    process             : 1.0;
    voltage              : 0.88;
    temperature          : -40.0;
  }

```

---

Figura 9: Ejemplo simplificado de cabecera de un archivo .lib SAED14, con definición de tecnología, unidades y condiciones de operación (corners PVT).

La segunda parte del .lib contiene la descripción detallada de cada celda: sus pines, funciones lógicas y tablas de temporización/potencia en función de *input slew* y carga de salida. Esta información es la base para que las herramientas de síntesis, STA y optimización física puedan estimar con precisión el comportamiento del diseño implementado sobre esa librería.

```

cell ( "SAEDRVT14_INV_0p5" ) {
    area           : 0.170000;
    cell_leakage_power : 0.00045;

    pin ( A ) {
        direction      : input;
        capacitance   : 0.00150;
        rise_capacitance : 0.00150;
        fall_capacitance : 0.00150;
    }

    pin ( ZN ) {
        direction      : output;
        function       : "!A";
        max_capacitance : 0.02000;
        max_transition : 0.10000;

        timing () {
            related_pin   : "A";
            timing_sense  : negative_unate;

            index_1 ( "0.01, 0.05, 0.10" ); /* input slew (ns) */
            index_2 ( "0.002, 0.010, 0.020" );/* load cap (pF) */

            cell_rise ( index_1, index_2 ) {
                values ( \
                    "0.03, 0.04, 0.05", \
                    "0.05, 0.06, 0.07", \
                    "0.07, 0.08, 0.09" );
            }

            cell_fall ( index_1, index_2 ) {
                values ( \
                    "0.02, 0.03, 0.04", \
                    "0.04, 0.05, 0.06", \
                    "0.06, 0.07, 0.08" );
            }

            rise_transition ( index_1, index_2 ) {
                values ( \
                    "0.01, 0.02, 0.03", \
                    "0.02, 0.03, 0.04", \
                    "0.03, 0.04, 0.05" );
            }

            fall_transition ( index_1, index_2 ) {
                values ( \
                    "0.01, 0.02, 0.03", \
                    "0.02, 0.03, 0.04", \
                    "0.03, 0.04, 0.05" );
            }
        }
    }
}

```

Figura 10: Ejemplo de segunda parte de un archivo .lib estilo SAED14: definición de una celda inversora, sus pines y tablas de temporización.

#### 2.2.4. Standard Cells

Una celda estándar (*standard cell*) es un diseño específico para cada compuerta en la biblioteca. Debe tomarse especial cuidado en diseño físcio de tales librerías para obtener un tamaño de dado (*die*) del ASIC y buena performance. El "dado" se refiere a la sección o área de silicio en la que se fabrica el circuito integrado. El tamaño de éste es una consideración crítica en el diseño de semiconductores, ya que afecta la cantidad de chips que se pueden fabricar en una sola placa de silicio (*wafer*) y, por lo tanto, puede tener un impacto significativo en el costo y la eficiencia de la fabricación. En la Figura 11 se puede observar una placa de silicio.

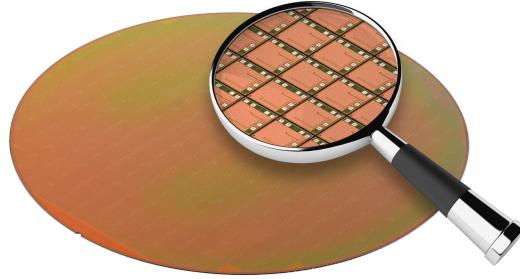


Figura 11: *Wafer* o placa de silicio conteniendo los *dies* (dados o matrices)

Con los avances en el proceso de fabricación y la creciente complejidad de los diseños lógicos, el área total de los diseños de ASIC está cada vez más dominada por el área de ruteo en lugar del área total de transistores utilizados. Por lo tanto, es necesario que el área de ruteo sea minimizada más que minimizar el área utilizada por las celdas estándar. Dado que la mayoría del enrutamiento ASIC se realiza automáticamente, es importante diseñar tamaños de celda estándar para que sean adecuados para las herramientas de PnR que se utilicen.

El paso básico en el diseño físico de celdas estándar comienza con la determinación de las pistas de cableado (*wire tracks*) horizontales y verticales. Estas son usadas para guiar las herramientas de place-and-route para realizar la interconexión entre celdas estándar.

Las directrices de fabricación, como el ancho y el espaciado de las dos primeras capas conductoras (por ejemplo, metal uno y metal dos), se utilizan para establecer el espaciado adecuado de las pistas de cableado. En la práctica, la manera de computar el espaciado de las pistas de cableado más óptima ha sido *Via-to-line*. Esta cumple con todas las reglas de espaciado de capas conductoras y presenta el enrutamiento general más compacto.

La mayoría de las herramientas de PnR requieren que la altura y el ancho de una celda estándar sean múltiplos enteros de la pista de cableado (*wire track*) vertical y horizontal. La altura de la celda estándar es la misma en toda la librería, pero sus anchos varían según sus funciones lógicas y *drive strength* (DS, fuerza de conducción).

En una celda estándar, la "fuerza de conducción" se refiere a la capacidad del buffer de salida de la celda para manejar la capacidad de carga de la red conectada a ella. La fuerza de conducción se mide típicamente en términos de "fanout", que es el número máximo de entradas de compuerta equivalentes que pueden ser manejadas por el buffer de salida de la celda sin causar una degradación en la calidad de la señal. La fuerza de conducción de una celda estándar es importante porque determina la capacidad de la celda para manejar las interconexiones y las cargas en el diseño.

Una celda estándar típica para el proceso de Complementary Metal Oxide Semiconductor (CMOS) está compuesta por una fila de transistores NMOS (tipo N) con ancho de canal  $W_n$ , y una fila de transistores PMOS con ancho de canal  $W_p$  separado por la distancia del área de difusión (o área activa) P y N. El espaciado del área de difusión P y N, el ancho del canal de los transistores PMOS y NMOS, y el ancho de los buses de alimentación (VDD) y tierra (VSS) son los parámetros clave para determinar la altura de las celdas estándar. Se ilustra en la Figura 12.

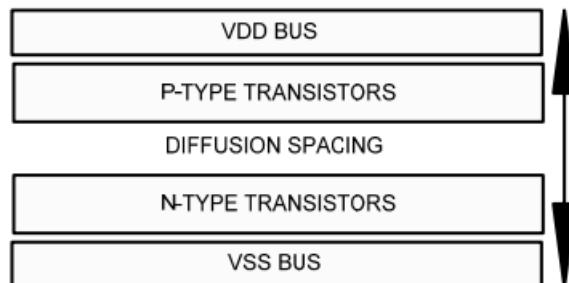


Figura 12: Altura generalizada de una celda estándar

Una vez determinadas las pistas de cableado vertical y horizontal, así como la altura de la celda estándar, esta

información se utiliza para crear un modelo de pista de cableado para su uso durante el diseño de la celda estándar.

Superponer el modelo de pista de cableado en un diseño de celda estándar durante el diseño físico como una guía de diseño asegura que la implementación de las celdas estándar y sus ubicaciones de pines intrínsecos cumplan con los requisitos de enrutamiento de la herramienta de PnR.

La Figura 13 muestra una malla (*mesh*) de pista de cableado marcada por la pista de cableado Horizontal y Vertical. En esta figura también se observa que es deseable usar la primera capa de metal para los puertos o pines de las celdas y ubicarlas donde los *tracks* de cable horizontal y vertical se cruzan.

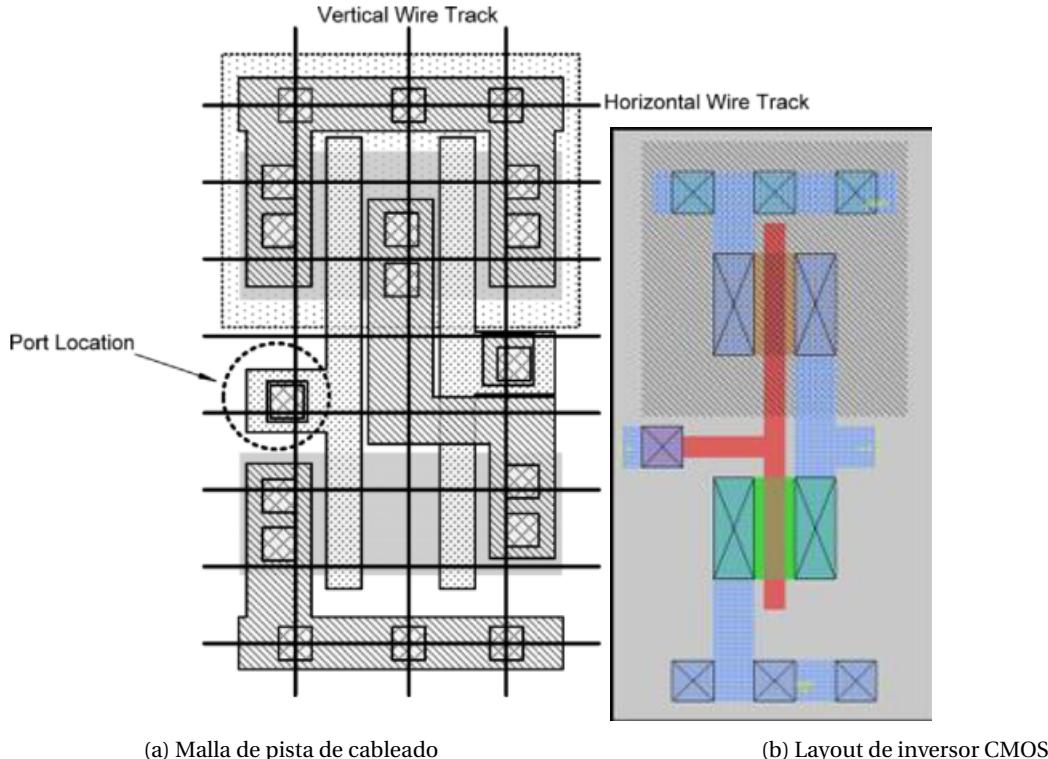


Figura 13: Std Cell Layout

El ancho de los buses de alimentación y tierra que atraviesan la parte superior e inferior de las celdas estándar es uno de los factores cruciales para definir la altura de la celda estándar. Las alturas de las celdas estándar están limitadas si la capa inicial de ruteo horizontal y las capas de alimentación (VDD) y tierra (VSS) son las mismas. Esto se debe a que el ancho de las líneas de VDD y VSS debe ser lo suficientemente amplio para proporcionar la capacidad adecuada de flujo de corriente, y este aumento en el ancho de las líneas de alimentación y tierra afectará la altura de la celda estándar.

Para reducir la resistencia y aumentar la inmunidad de las celdas al fenómeno de *latch-up*, es conveniente utilizar múltiples cortes de contacto o lo que se conoce como "*fully strapped*". Como los anchos de alimentación y tierra mínimos están relacionados al tamaño de contacto del primer metal, los anchos de ambos buses deben ser suficientemente anchos para evitar violaciones de reglas de diseño físicas.

Es preferible usar la primera capa de conducción para conexiones y en los metales más bajos, dentro de la celda.

La regularidad geométrica a lo largo de todo el diseño de las celdas del mismo tipo tiene dos ventajas permite el uso de software de compactación para reducir el área de las celdas estándar al mismo tiempo que permite la migración de una librería de celdas de unas reglas de diseño a otro. Además, es ventajoso eléctricamente ubicar los transistores NMOS lo más cerca posible del bus VSS y los PMOS al bus VDD.

El proceso CMOS submicrónico actual es muy complejo en su naturaleza. Sin embargo, en el diseño ASIC, al diseñar *layouts* de celda estándar para el proceso CMOS, un conjunto mínimo de reglas de diseño a modo de ejemplo es mostradas en el Cuadro 2 es adecuado para una *standard cell* que usa hasta metal dos en su diseño.

Anchura mínima de un nwell
Espacio mínimo entre dos nwell del mismo potencial
Superficie mínima de nwell
Ancho mínimo de difusión para definir el ancho NMOS/PMOS
Ancho mínimo de difusión para la interconexión
Espacio mínimo entre regiones de difusión
Superposición mínima de nwell sobre la región P+ dentro de nwell
Espacio libre mínimo de nwell a P+ fuera de nwell
Área mínima de difusión
Ancho mínimo de un poly para la longitud del canal del transistor MOS
Anchuro mínimo de un poly para interconexión
Espacio libre mínimo de difusión a poly sobre óxido de campo
Espacio mínimo entre dos poly en el área de óxido de campo
Mínimo poly extendido en óxido de campo
Mínimo área de poly
Tamaño de contacto
Espaciado de contactos
Espacio mínimo de contacto en la difusión a poly
Espacio mínimo de contacto en poly a difusión
Ancho máximo del metal 1
Extensión mínima de metal1 sobre contacto
Espacio metal1 mínimo
Superficie mínima de metal1

Cuadro 2: Set de reglas de diseño

En La Figura 14 , se muestra los pasos para el desarrollo de una Librería.

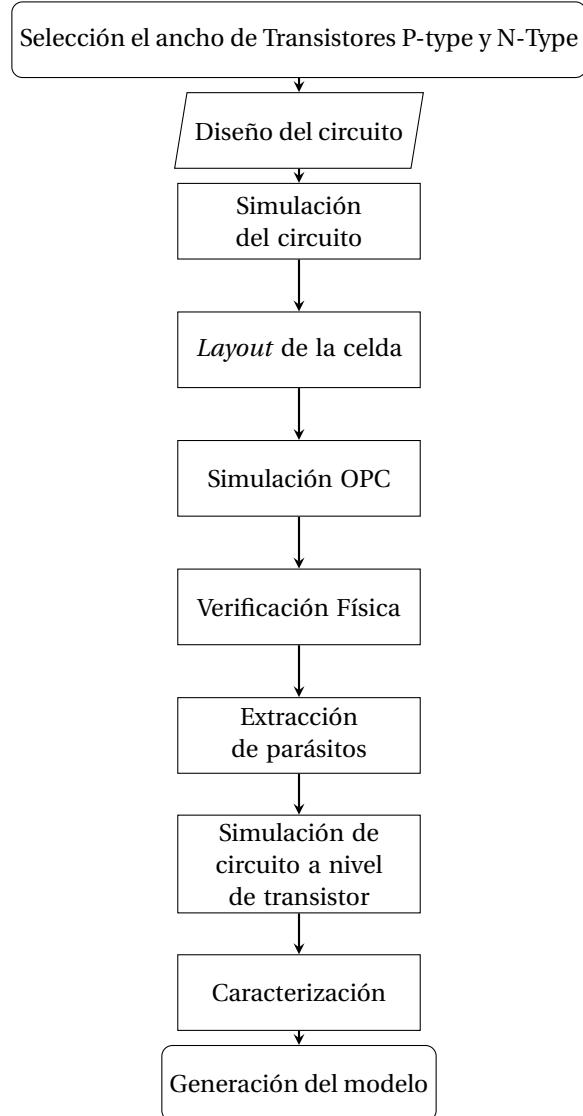


Figura 14: Pasos para el desarrollo de una librería

### 2.2.5. Netlist y RTLs

En la literatura, el término *netlist* puede usarse de forma amplia para referirse a cualquier descripción de conectividad de un circuito. Sin embargo, en el contexto de este escrito se adopta la siguiente convención:

- Llamaremos **RTL** a la descripción de alto nivel en un HDL (Verilog, SystemVerilog o VHDL) basada en registros y lógica combinacional, donde se especifica el comportamiento del circuito mediante operaciones aritméticas, comparaciones, estructuras *if/case*, etc., pero *sin instanciar explícitamente celdas estándar* de la librería.
- Llamaremos **netlist** al resultado de la *síntesis lógica*: una descripción estructural, de bajo nivel, donde el diseño está implementado mediante instancias de celdas estándar (puertas lógicas, *flip-flops*, buffers, multiplexores, etc.) y sus interconexiones.

Un **netlist** es, por tanto, una representación de bajo nivel del diseño que enumera los componentes lógicos y cómo se conectan entre sí. Normalmente es generado por una herramienta de síntesis lógica a partir de una descripción RTL en Verilog o VHDL. El netlist se utiliza como entrada para el flujo de diseño físico, es decir, para las etapas de *Place & Route* del circuito sobre el chip.

Por otro lado, la **descripción RTL** es una representación de nivel más alto que describe la funcionalidad del circuito en términos de registros y lógica combinacional: indica cómo se transfieren los datos entre registros en cada flanco de reloj y qué operaciones se realizan sobre esos datos. Esta descripción se usa principalmente

para simulación y verificación funcional en etapas tempranas, y es el punto de partida para la síntesis que generará el netlist.

En resumen, el RTL describe *qué* hace el circuito (a nivel de registros y operaciones), mientras que el netlist describe *con qué* se implementa físicamente (celdas estándar y sus conexiones). En la Figura 15 se muestra un ejemplo básico de un sumador de 4 bits en Verilog: en la subfigura 15a, el código RTL utiliza la operación  $a + b$  para describir la suma; en la subfigura 15b, el mismo sumador se implementa como un netlist estructural instanciando celdas estándar (sumadores completos de 1 bit y buffers de salida) que podrían pertenecer a la librería de celdas utilizada en el diseño físico.

```

module adder (
    input [3:0] a, b,
    output [3:0] sum
);
    assign sum = a + b;
endmodule

```

```

module adder (
    input [3:0] a, b,
    output [3:0] sum
);
    wire c1, c2, c3;

    // c0 = 0
    FA_X1 U0 (
        .A (a[0]),
        .B (b[0]),
        .CI (1'b0),
        .S (sum[0]),
        .CO (c1)
    );

    FA_X1 U1 (
        .A (a[1]),
        .B (b[1]),
        .CI (c1),
        .S (sum[1]),
        .CO (c2)
    );

    FA_X1 U2 (
        .A (a[2]),
        .B (b[2]),
        .CI (c2),
        .S (sum[2]),
        .CO (c3)
    );

    FA_X1 U3 (
        .A (a[3]),
        .B (b[3]),
        .CI (c3),
        .S (sum[3]),
        .CO () // carry final no utilizado
    );
endmodule

```

(a) Descripción RTL

(b) Netlist estructural

Figura 15: Ejemplo de RTL y netlist estructural para un sumador de 4 bits. En el RTL se describe el comportamiento con la operación  $a + b$ , mientras que el netlist instancia celdas estándar FA\_X1 (sumadores completos de 1 bit) provenientes de la librería de celdas.

## 2.2.6. Condiciones de operación, *corners* y escenarios

Se llama *PVT corners* a las combinaciones de **Proceso** (*Process*), **Voltaje** (*Voltage*) y **Temperatura** (*Temperature*) en las que se simula y caracteriza el comportamiento del diseño. Las propiedades intrínsecas de los materiales de fabricación modifican el comportamiento de las compuertas básicas de las librerías de *standard cells* en función de la temperatura, el voltaje de alimentación y el proceso de fabricación. Estas variaciones se traducen en cambios en las dimensiones efectivas de los transistores, en la resistencia de las interconexiones y, en definitiva, en el retardo y la potencia del circuito.

Dado que los circuitos integrados pueden ser utilizados en entornos muy distintos, es necesario garantizar su funcionamiento dentro de un rango de temperatura adecuado para la aplicación final. En la Figura 16 se muestran ejemplos de rangos de temperatura típicos para distintas clases de producto (militar, extendido, industrial y comercial).

CLASS	Lower Limit	Upper Limit
Military	-55C	+125C
Extended	-40C	+125C
Industrial	-40C	+85C
Commercial	0C	+70C

Figura 16: Rangos de temperaturas de operación según clase de producto.

Las *foundries* se encargan de proveer, para cada *corner*, modelos de *timing* de *standard cells* (archivos .lib) que simulan el comportamiento del circuito en los límites de estos rangos de PVT. Cada conjunto de modelos (uno por *corner*) se entrega en archivos separados y contiene información tanto del retardo interno de las celdas como del retardo asociado a las interconexiones. Estos datos son utilizados por las herramientas de síntesis, *Place & Route* y STA para calcular el tiempo de propagación de los datos a lo largo de un *path* de temporización.

Las variaciones de proceso pueden modelarse de manera simplificada como una distribución normal en torno a un valor *típico*. El punto central se denomina *typical (TT)*, mientras que los extremos alejados aproximadamente  $\pm 3\sigma$  representan los casos *lentos (slow o worst case)* y *rápidos (fast o best case)*. Sobre estos puntos se definen los distintos *PVT corners* (por ejemplo, *ss, tt, ff*) que se utilizan posteriormente en el análisis de temporización y potencia.

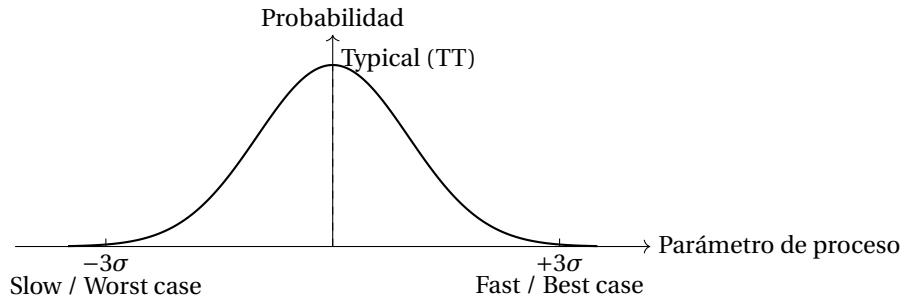


Figura 17: Distribución normal simplificada de variaciones de proceso y definición de *PVT corners*.

En la Figura 18 se ve la variación del *delay* de las celdas standard en función de los corners PVT.

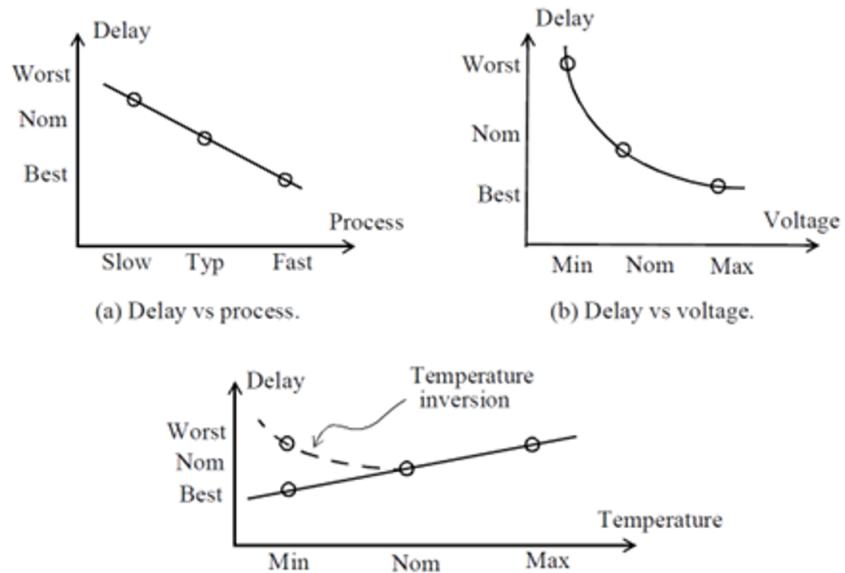


Figura 18: Variación del delay en función del corner PVT

Los escenarios se constituyen de un corner más de un modo de operación. Los modos de operación hacen referencia a los estímulos externos que se usarán para el desarrollo del diseño, entradas, salidas y clocks habilitados. Generalmente, en *PnR*, existen dos grandes modos de operación: el modo funcional y el modo de SCAN.

El modo funcional estimula el diseño con los vectores de entrada para los cuales se diseñó, tratando de validar la funcionalidad del mismo, mientras que el modo "scan" se utiliza para comprobar fallas inherentes a la construcción del IC. En este escrito se hace énfasis en la implementación funcional del diseño, la implementación del diseño de escaneo se llama *Design For Testing* (DFT) y no es parte del laboratorio del proyecto.

Como conclusión, se debe tener en cuenta las peores y mejores condiciones para realizar las simulaciones correspondientes, si se espera que la simulación sirva para diseñar y lograr que la mayoría de los chips de un lote fabricado cumpla con las especificaciones de diseño.

### 2.2.7. Síntesis

La síntesis de un diseño consiste en obtener una implementación optimizada en timing a nivel de compuertas de un *netlist*/RTL. Para llegar a esto, las herramientas ejecutan diferentes procesos que varían según el proveedor, pero generalmente contemplan los siguientes pasos:

1. Lectura de HDL: Se indica cuáles son los *paths* en donde buscar el *top level* o jerarquía mayor del diseño y los HDL de los que depende.
2. Elaboración: La herramienta infiere estructuras lógicas genéricas para los bloques de código de cada RTL. Sin optimizar, eliminar, agregar o modificar módulos.
3. Lectura de *constraints* de timing: se leen los archivos *.sdc/.tcl* con las definiciones de *clocks*, incertidumbres, *false paths*, input/output delay, atributos como "*don't touches*", etc.
4. Síntesis genérica: Primera optimización donde comienzan a inferirse estructuras lógicas genéricas más sencillas. Se remueven los registros sin carga o constantes y por ende grandes partes del diseño post-elaboración, se comienza a tener en cuenta optimizaciones de *timing* y fanout de nets.
5. Mapeo: las estructuras lógicas son reemplazadas por combinaciones de compuertas, registros, compuertas de clock u otros componentes discretos disponibles en la librería. Se continúa optimizando, tratando de encontrar las compuertas que cumplen los requerimientos indicados al inicio de la síntesis (*timing*, potencia, área, *leakage*).

6. Síntesis de optimización: Se enfoca directamente en los settings indicados o por defecto que posea la herramienta, *timing*, potencia, área. Realiza iteraciones y comienza a agregar nuevas compuertas que ayuden a cumplir los objetivos. Optimización de retardo basado en la lógica utilizada.

Si bien la síntesis consta de los pasos nombrados anteriormente, se pueden distinguir dos tipos dependiendo de la información con la que cuenten. Si la síntesis cuenta con información del layout (tamaño disponible, ubicación de puertos, macros e IPs) se llamará síntesis física, mientras que si no cuenta con esta información, la llamaremos síntesis lógica.

Usualmente la síntesis física tendrá ventaja sobre la lógica debido a la cantidad de información que posee, en este punto, podrá decidir el tamaño de las compuertas a utilizar, como ubicar cada una de ellas dentro del *layout* y estimará con mas precisión los parásitos de las nets.

### 2.3. Place and Route

El objetivo de este flujo es traducir el lenguaje descriptivo de RTL en lenguaje de hardware usando EDAS.

El Place and Route (PnR) es un paso crítico en el proceso de automatización del diseño electrónico (EDA) que implica la colocación de componentes electrónicos y el enrutamiento de conexiones entre ellos en una placa de circuito impreso (PCB) o un chip de circuito integrado (IC). Se realiza cumpliendo todas las reglas de diseño de la tecnología actual (brindada por la *foundry*).

El proceso de Place and Route típicamente involucra varios pasos, incluyendo la planificación inicial de la disposición, la colocación de celdas lógicas, la optimización de la disposición para el tiempo y la potencia, y finalmente el enrutamiento de conexiones entre las celdas colocadas, en este punto el *netlist* obtiene una representación física (o gráfica) llamada *layout*. Este proceso es esencial para diseñar circuitos digitales complejos y asegurar que cumplan con los requisitos funcionales y de rendimiento.

Se identifican dentro de este flujo las etapas mostradas en la Figura 19.

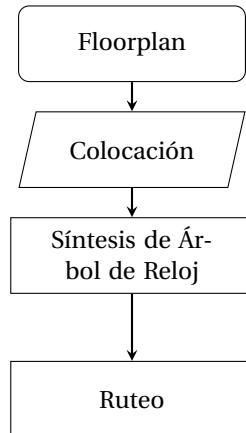


Figura 19: Pasos para el PnR

Hay varias herramientas de software disponibles para automatizar el proceso de Place and Route, como Cadence Encounter, Synopsys Design Compiler y Mentor Graphics Olympus-SoC. Estas herramientas utilizan algoritmos sofisticados para optimizar la colocación y el enrutamiento para el tiempo, la potencia y otras restricciones.

#### 2.3.1. Floorplanning

En la etapa de plano general, o *floorplan* por sus siglas en inglés, se define el tamaño del bloque, la ubicación de los puertos I/O y las macros, la creación de la *PowerGrid* (PG) y las costumizaciones como bloques de *placement*, de ruteo o guías de *placement*.

En el diseño ASIC, un "macro" se refiere a un bloque o módulo funcional pre-diseñado y pre-verificado que puede ser reutilizado en diferentes diseños. Los macros se utilizan comúnmente para ahorrar tiempo y reducir el riesgo de errores al diseñar circuitos integrados complejos. Los macros pueden ser simples, como un flip-flop o un contador, o pueden ser complejos, como un núcleo de microprocesador o un bloque de procesamiento de señales digitales (DSP). Están diseñados para ser fácilmente integrables en diseños más grandes y tienen interfaces bien definidas para comunicarse con otros módulos en el sistema.

Los **bloques** están diseñados para ser autocontenido, lo que significa que pueden operar de manera independiente al resto del sistema. Esto permite una prueba, verificación e integración más sencilla de todo el chip ASIC. Los bloques pueden variar en tamaño y complejidad dependiendo de su función y los requisitos del sistema. Algunos ejemplos comunes de bloques ASIC incluyen bloques de memoria, bloques de procesamiento de señales digitales y bloques de interfaz de comunicación.

En este punto se determinará la implementación del diseño correcta o no, y que dependen exclusivamente del diseñador físico a cargo de tal. Un mal *floorplanning* podría convertir un diseño sencillo en uno complicado, y viceversa. Es fundamental tener en claro cuáles son los requerimientos del diseño en cuanto a cantidad de puertos, macros o IPS, cómo se involucran en el flujo de datos y cuál sería la ubicación óptima de estos en el *layout*. Cuando se trata de diseños complejos, es importante tener una comunicación constante con quien haya diseñado el RTL ya que podrá brindar información precisa de las complicaciones que se podrían ver en el futuro.

Habrá un *trade-off* entre el área, la forma disponible para dicho diseño y el *floorplanning* que mejores métricas brinde (menos área, mejor uso de recursos, facilidad para optimizar *timing paths*, reducción de potencia, etc). Se tendrán más o menos grados de libertad dependiendo de la forma del top level y la partición de netlist final de tal.

Para realizar el *floorplanning* se requieren los siguientes inputs:

- RTL/netlist sintetizado
- Constraints de timing del diseño (i.e. archivos sdc)
- Librerías físicas/timing de la tecnología donde se implementará (.lib, .lef o .ndm)
  - o Librerías de timing caracterizadas para los corners que se utilizaran
- Archivo de tecnología (.tf, .tlef)

Se realiza una variedad de comprobaciones automáticamente para determinar si la estructura de datos interna de la síntesis física o de ubicación y enrutamiento está lista para continuar con el resto del flujo de implementación del diseño. A menudo, las verificaciones que detectan problemas con la base de datos física están relacionadas con la netlist, como puertos no conectados, puertos no coincidentes, errores en las celdas estándar o errores en los archivos de la biblioteca y la tecnología. Debido a estas verificaciones, se generará un archivo de registro que contiene todos los errores y advertencias. Es importante revisar el archivo de registro y asegurarse de que se resuelvan todos los errores y advertencias informados antes de pasar a la siguiente fase.

Después de crear la base de datos física utilizando la netlist importada y los archivos de biblioteca y tecnología correspondientes, el primer paso es determinar el **ancho y la altura** del dispositivo ASIC y del núcleo. Además, se crean filas (*rows*) de celdas estándar y sitios (*sites*) de pad de entrada/salida (I/O). Las filas y los sitios de pad de entrada/salida son para la colocación de celdas estándar y pads de entrada/salida. La Figura 20 muestra un floorplan inicial de un diseño ASIC.

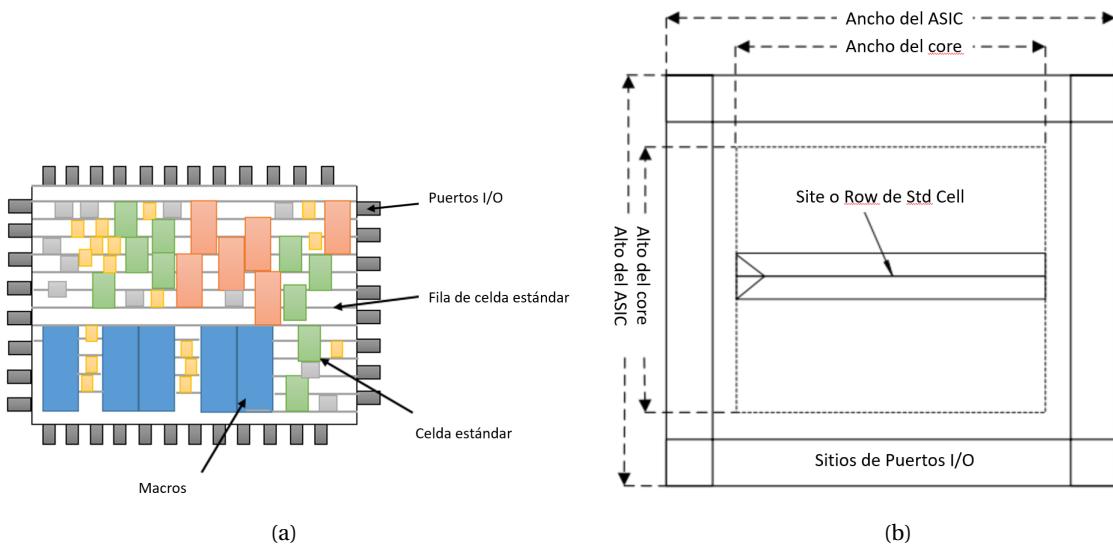


Figura 20: Boceto de un Floorplan

La altura de una fila es igual a la altura de las celdas estándar en la biblioteca. Si hay múltiples celdas estándar de diferentes alturas en la biblioteca, ocuparán múltiples filas. La mayoría de las veces, las filas estándar se crean por "*abutment*"(acoplamiento en español), lo que implica que las filas estándar están orientadas en rotación alternante de 180 grados o se voltean a lo largo del eje X para que las celdas estándar puedan compartir buses de alimentación y tierra. Si el núcleo ASIC tiene congestión de enrutamiento debido al número limitado de capas de enrutamiento, una solución es crear canales de enrutamiento entre filas. Todos estos pueden separarse individualmente o en pares.

La **posición de los puertos** es uno de los últimos parámetros a definir al momento de crear un floorplan. Los puertos son las entradas y salidas de un bloque y pueden desde cientos a miles, dependiendo de la naturaleza del bloque. Una mala planificación de puertos puede implicar problemas de congestión a futuro. La posición depende del flujo de datos, además del top level y su ubicación respecto al bloque vecino.

En general, los puertos se colocan en grupos y a su vez los grupos se dividen en subgrupos. El orden de agrupación depende de: la dirección (entrada o salida), bus (grupo de bits) y el propósito (Datos, submódulos, DFT, clock, etc). Los puertos de datos son los más importantes porque tienen que cumplir requisitos de timing más importantes. Por otro lado, los puertos de clock pueden tener pines de distinto grosor y espaciamiento para disminuir la resistividad y el ruido, respectivamente.

Todos los puertos sincrónicos deben ser registrados. Esto significa que los datos que entran en el chip deben pasar por un registro antes de ser utilizados en la lógica sincrónica. De esta manera, se garantiza que la señal de reloj esté estabilizada antes de que los datos sean procesados, lo que reduce el riesgo de errores de sincronización y aumenta la confiabilidad del circuito. Además, el registro permite el control de la temporización y el flujo de datos, lo que es esencial en el diseño de circuitos digitales. Por otro lado, las salidas no necesariamente tienen que ser registradas, pero es una práctica común registrarlas también para garantizar un sincronismo y una sincronización adecuados con los demás componentes del diseño. Al registrar las entradas y salidas, los problemas de tiempo como los tiempos de configuración y retención pueden ser manejados adecuadamente, y el diseño puede operar de manera confiable a la frecuencia de reloj requerida.

Uno de los últimos pasos que el diseñador tiene que realizar en el *floorplanning* es el **posicionamiento de las macros**, que pueden ser memorias u otros módulos. En general las memorias son los dispositivos más problemáticos al momento de cerrar *timing*, por lo que es crucial tener criterio al momento de diseñar el layout. Al igual que los puertos, es necesario conocer el flujo de datos para colocar las memorias de manera acorde. En algunos casos es conveniente colocar las memorias en un costado o esquina para que no afecten en el cableado de lógica combinacional, y en otros es conveniente colocarlos en el medio del flujo de datos porque se usan muy frecuentemente.

La ubicación de macros puede ser manual o automática. La ubicación manual de macros es más eficiente cuando hay pocos macros que colocar y se conoce su relación con el resto del diseño del ASIC. Es necesario asegurarse de que haya suficiente espacio entre los bloques para las interconexiones, proceso comúnmente conocido como "*definición de canal*"(*channel definition*), puede ser manual o realizado por las herramientas de floorplan. Para definir los canales de ruta entre bloques, se utiliza una estructura de datos jerárquica, llamada "*slicing tree*", que es utilizada por los algoritmos de floorplanning para dividir el área del chip en rectángulos o *slices* más pequeños, ayudando también a reducir la congestión y optimizar el uso de los recursos disponibles.

Dada una solución de colocación, las medidas físicas podrían ser la longitud de los cables, la dirección del flujo de datos (por ejemplo, la colocación relativa de los macros entre sí y con respecto a la colocación de celdas estándar), o la accesibilidad y el tiempo relacionados a los puertos y macros. La longitud total de los cables para una colocación dada es un buen indicador al comparar diferentes colocaciones del mismo diseño físico. Para disminuir la longitud total de los cables, es importante asegurarse de que el área del chip no esté segmentada por la colocación de los macros.

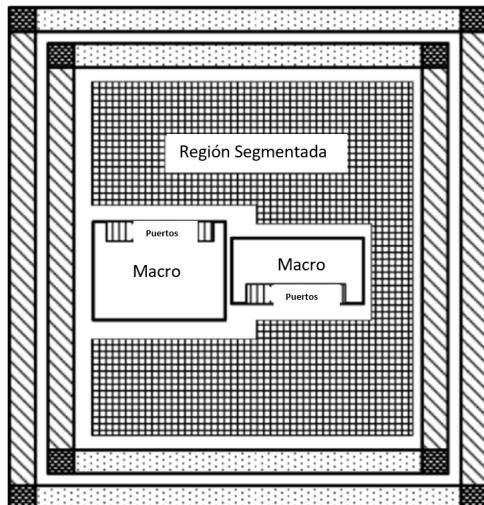
Para evitar la segmentación del área, los macros deben colocarse de manera que el área de las celdas estándar sea continua. Se recomienda un área con una relación de aspecto cercana a 1:1, ya que permite a los colocadores de celdas estándar utilizar el área de manera más eficiente y, por lo tanto, reducir la longitud total de los cables. Un floorplan segmentado conduce a un exceso de interconexiones de longitud de cable desde las celdas estándar ubicadas en la parte inferior del dato hasta aquellas en la parte superior del dato, como se muestra en la Figura 21a .

En cuanto a la reducción de la longitud de los cables, los macros deben orientarse de tal manera que sus puertos estén orientados hacia las celdas estándar, o el área central, y su orientación debe coincidir con las capas de enrutamiento disponibles. Por lo tanto, cualquier algoritmo de colocación de macros requiere el cálculo de la distancia de interconexión de la macro, incluyendo la orientación y la posición de los pines adecuados. En la Figura 21b se ve un floorplan con sus macros enfrentando las la región de celdas estándar, por lo tanto minimizando el aumento localizado en longitud de cableado.

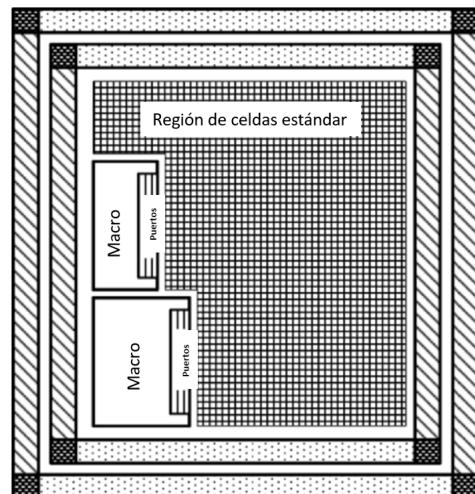
Otro aspecto de la medida física es el de las colocaciones de macros relativas entre sí, así como con respecto a la colocación de celdas estándar, y la accesibilidad de los puertos de la macro tiene un impacto directo en el enrutamiento final del chip. La colocación de macros, y por lo tanto la medida de calidad, puede ser determinada por el análisis de la congestión de enrutamiento producida por un enrutador global, que produce reportes estadísticos tanto de texto como gráficos con ayudas visuales (i.e. hot spots).

Los escenarios más comunes por las que puede presentarse congestión de ruteo son:

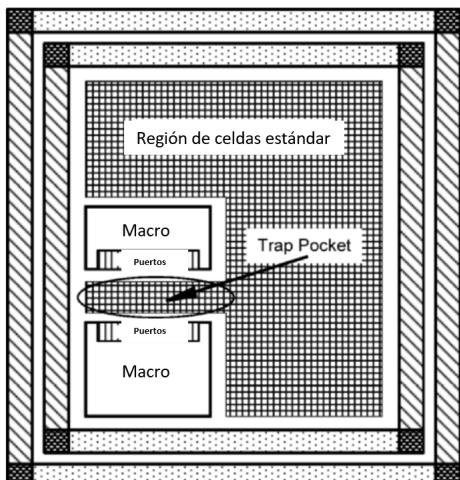
- No hay suficiente espacio entre las macros para proporcionar canales de ruteo, especialmente para los puertos I-O (por ejemplo si las macros están en la periferia del ASIC)
- el ruteo está prohibido
- "*trap pockets*" de las celdas estándar están en los bordes de las macros o entre los corners del floorplan. Estas son canales largos y delgados entre macros que deberían ser liberados sólo para repetidores o inserción de buffers. La Figura 21c muestra un floorplan con un Trap Pocket.



(a) Floorplan con región segmentada



(b) Macros enfrentando celdas estándar



(c) Floorplan con Trap Pocket

Figura 21: Ubicación de macros en el floorplan

Después de la ubicación de macros y antes de realizar el enrutamiento global, la mayoría de los diseños físicos requieren que se definan regiones de exclusión o de solo búfer dibujando una capa de bloqueo sobre un área que contiene macros para evitar que el colocador mueva cualquier celda estándar hacia esas regiones. Los cables que se utilizan en las regiones de exclusión tienden a ser largos pero al permitir la inserción de búfer en esas áreas mediante el uso de una región de solo búfer (o bloqueo), el colocador reducirá la longitud de estos cables largos y, por lo tanto, evitará los largos tiempos de transición asociados con ellos.

Las capas de bloqueo también se utilizan para aliviar la congestión de enrutamiento alrededor de las esquinas de los macros. Cuando un macro está bloqueado en muchas capas de enrutamiento, los cables tienden a desviarse alrededor de las esquinas y conectarse a celdas estándar cercanas, creando así congestión de enrutamiento en la esquina de los macros. Para reservar más recursos para el enrutador, se puede dibujar una capa de bloqueo en estas esquinas. Estas regiones de bloqueo pueden ser simples o graduales, como se ilustra en la Figura 22.

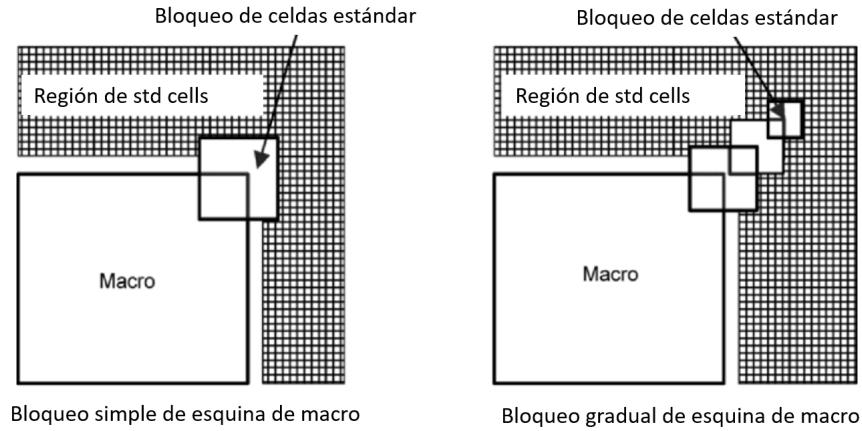


Figura 22: Bloqueos de celdas estándar en esquinas en una macro

Después de la refinación del floorplan y la colocación de macros, se colocan celdas estándar y se realiza un análisis de conectividad. El análisis de conectividad es el proceso de estudiar las conexiones entre macros, puertos de entrada/salida y las instancias de celda estándar relacionadas. El análisis de conectividad también se utiliza para identificar macros que tienen una conectividad directa sustancial y para refinar sus ubicaciones en consecuencia.

Este análisis se lleva a cabo utilizando lo que se conoce como *fly lines*. Cuando se activan las *fly lines* a través de herramientas de diseño físico o de Place & Route, la interfaz gráfica de usuario (GUI) muestra las líneas que marcan las conexiones entre las instancias seleccionadas actualmente (por ejemplo, celdas estándar, macros o pads de entrada/salida). Usando *fly lines*, se pueden analizar e identificar situaciones en las que mover o rotar macros producirá longitudes de cable más cortas que mejorarán la rutabilidad general del ASIC durante la etapa de floorplanning del ciclo de diseño físico. En la Figura 23 se muestra un ejemplo de *fly lines*.

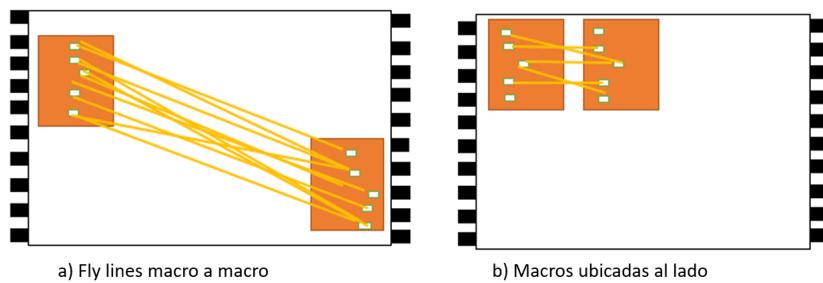


Figura 23: Fly lines

### POWER PLANNING

"Power planning" (planificación de alimentación o energía) en el diseño ASIC se refiere a la planificación y diseño de la distribución de energía y tierra en todo el circuito integrado. El objetivo de la planificación de energía es garantizar que cada componente del circuito reciba suficiente energía de la fuente de alimentación para operar de manera confiable, mientras que la planificación de la tierra se enfoca en proporcionar una ruta de retorno de corriente adecuada.

Se describe el *power planning* para un "flip chip", que refiere a un tipo de tecnología de empaquetado para circuitos integrados (IC). En un paquete de flip chip, el IC se da vuelta y se monta directamente sobre un sustrato, como una placa de circuito impreso (PCB), con soldadura de bolas o *bumpers* conectando el chip

al sustrato. Esto se contrasta con otras tecnologías de empaquetado, como el cableado, donde el IC se monta verticalmente y se conecta al sustrato con alambres.

El proceso comienza desde una base de silicio donde se forman los transistores, y luego se van agregando capas de metal. Cada capa puede tener un metal distinto, cada uno con distintos requisitos y características. El conjunto de metales se le llama “metalización”, y cada proyecto puede tener una metalización distinta.

Por encima de la última capa de metal se agrega una capa del tipo RDL (Redistribution Layer, capa de redistribución) donde se encuentran los *bumps* por los que se alimenta el chip. Este formato implica que la corriente que ingresa por los *bumps* debe circular a través de todas las capas de metal para llegar a las celdas. Por lo tanto, se requiere una infraestructura que sea lo suficientemente eficiente para que la caída de potencial sea mínima y suficientemente espaciada para que no interfiera con el cableado de señales. En la Figura 24 se puede observar la estructura de un flip chip.

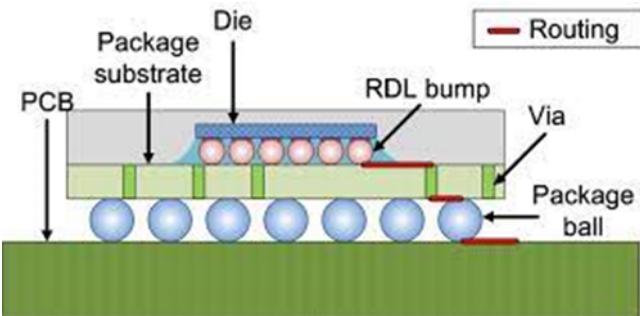


Figura 24: Estructura de un *flip chip*

La grilla de alimentación, red de energía, o Power Grid (PG) se utiliza para distribuir la energía eléctrica a través del chip. En general, los circuitos integrados requieren múltiples niveles de voltaje y corriente para alimentar los diferentes bloques funcionales del chip. La Power Grid se encarga de suministrar estos niveles de voltaje y corriente a los bloques de circuitos específicos del chip de manera uniforme y eficiente. Es un conjunto de rieles y pilares que se asegura que todas las celdas tengan la alimentación apropiada. Al igual que las celdas y macros, los netlists de los bloques también poseen pines de power (VDDC, VSSC VDDM).

La primera capa de metal (M0) siempre está formada por rieles, y conecta los pines de alimentación de todas las celdas. El formato de todas las capas superiores se decide por los diseñadores, pero se suelen seguir las recomendaciones de la fundidora. La fundidora siempre provee una “PG recomendada” para la tecnología. En la Figura 25 se observan pilares desde una capa M2 a una capa M6 de la metalización.

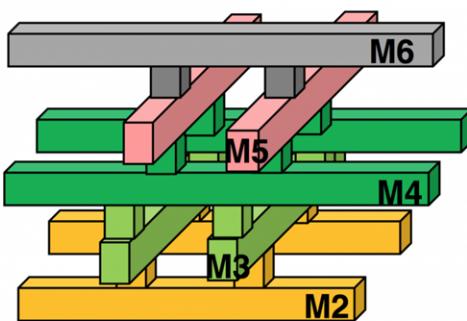


Figura 25: Pilar desde metal M2 a metal M6

En cuanto a los pines de alimentación, las celdas los poseen en los lados superior e inferior, y tienen el mismo ancho que la celda. De ser necesario, la celda será girada 180 grados por la herramienta para hacer coincidir los pines con los rieles. Se muestra en la Figura 26 la distribución de los pines en las filas de celdas estándar.

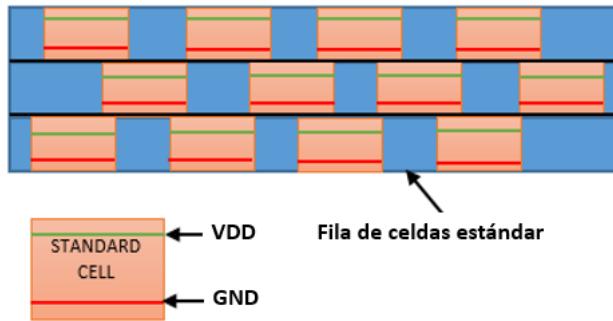


Figura 26: Disposición de pines de alimentación

A modo de ejemplos prácticos, se listan algunas implementaciones de redes de alimentación comúnmente utilizadas en diseños ASIC en tecnología de 14nm:

1. Red de alimentación múltiple VDD (MV): En esta implementación de red de alimentación, se utilizan rieles de suministro de energía separados para diferentes partes del circuito, lo que permite el uso de diferentes niveles de voltaje en diferentes áreas para optimizar el rendimiento, el consumo de energía y la utilización del área. Por ejemplo, un bloque de alto rendimiento puede usar un riel de suministro de voltaje más alto para lograr velocidades más altas, mientras que un bloque de bajo consumo de energía puede usar un riel de suministro de voltaje más bajo para minimizar el consumo de energía.
2. Red de alimentación híbrida VDD/GND (HV): En esta implementación de red de alimentación, se utiliza una combinación de rieles VDD y GND para proporcionar una distribución de energía más uniforme y minimizar las caídas de voltaje. Por ejemplo, los rieles VDD y GND pueden estar entrelazados para reducir la distancia entre ellos y minimizar la resistencia de la red de distribución de energía.

Las implementaciones de red de alimentación se pueden optimizar aún más a través de una combinación de simulación de circuitos, optimización de diseño y análisis de energía para lograr los objetivos deseados de rendimiento, consumo de energía y utilización del área para un diseño ASIC específico.

### 2.3.2. Placement

El objetivo de la ubicación de las celdas estándar es mapear los componentes del ASIC, o celdas, en posiciones del área central del ASIC, o región de ubicación de celdas estándar, que está definida por filas. Las celdas estándar deben colocarse en la región asignada (es decir, filas) de manera que el ASIC pueda ser enrutado eficientemente y se puedan satisfacer los requisitos generales de *timing*.

La ubicación de las celdas estándar puede considerarse como un proceso automático que requiere menos intervención por parte del diseñador físico. Sin embargo, existen varios restricciones de diseño que se pueden aplicar durante la ubicación de las celdas estándar para lograr un diseño ASIC óptimo en términos de área, rendimiento y consumo de energía. Estas restricciones pueden ser en torno a la congestión, sincronización, consumo de energía o cualquier combinación de ellas, siempre guiando a la herramienta para una correcta optimización.

La fase de ubicación tendrá un impacto significativo en la viabilidad del diseño, ya que las celdas estándar posicionadas en esta etapa no podrán ser desplazadas considerablemente en fases posteriores. Subsecuentes etapas podrán mover algunas celdas para cumplir con las reglas de diseño, aunque la mayoría de las celdas que componen las jerarquías internas permanecerán en su posición original.

La mayoría de las herramientas de PnR siguen un proceso en dos etapas para posicionar las instancias de celdas estándar. Estas etapas se conocen como ubicación global y ubicación detallada. El objetivo del algoritmo de ubicación global es reducir al mínimo las longitudes de los cables de interconexión, mientras que el propósito del algoritmo de ubicación detallada es satisfacer las restricciones de diseño, como el tiempo y/o la congestión, y completar la disposición final de las celdas estándar. También, previo a correr la etapa de ubicación, se definen los *settings* de la herramienta para la tecnología indicada (*constraints de placement*). Se establecen cuáles serán los targets y el esfuerzo (power, congestión, área, leakage), umbrales de *timing*, grupos de caminos críticos, jerarquías complejas a optimizar, entre otros.

Dentro la primera etapa, **ubicación global**, se pueden definir regiones. Las regiones pueden ser suaves o rígidas. Una región suave es una restricción física donde un módulo lógico se asigna a una ubicación en el núcleo y en el límite de la región, y está sujeta a cambios durante la colocación de celdas estándar. Por otro lado, una región rígida es más rigurosa que la región suave y define una partición física para el diseño

modular. Tiene límites rígidos que impiden el cruce de celdas estándar durante la colocación. Al utilizar la opción de región rígida, se debe definir tanto la ubicación como la forma de la región. Esta opción se utiliza principalmente para problemas relacionados con el tiempo, como agrupar dominios de reloj, voltaje o umbral. Además, las regiones pueden ser exclusivas o no-exclusivas.

La herramienta coloca inicialmente todas las celdas estándar de manera aleatoria en el diseño y, a través de iteraciones basadas en la distancia entre secciones de lógica y la longitud de los cables, comienza a reorganizar las jerarquías. También considera las regiones o guías de ubicación y los bloqueos. Por lo general, en esta etapa, no se llevan a cabo análisis exhaustivos de congestión y sincronización durante las iteraciones. No obstante, en algunas herramientas, al final de esta subetapa, podría realizarse un análisis de congestión que sienta las bases para los comandos posteriores.

En la subetapa de **ubicación detallada**, como en la fase precedente, la herramienta comienza a considerar las configuraciones establecidas al comienzo del proceso. Además, se encarga de llevar a cabo simultáneamente análisis exhaustivos de congestión y sincronización. En lo que respecta al sincronismo, la herramienta empieza a corregir las rutas críticas para cada dominio de reloj. Introduce buffers e inversores para agilizar los caminos de *setup*, lo que tendrá un impacto considerable en cada una de las iteraciones. Junto con su enfoque en los términos de calidad de resultados mencionados previamente, la herramienta se ocupa de regularizar el diseño. Esto implica eliminar superposiciones entre dispositivos y colocar adecuadamente todas las celdas estándar. Generalmente, esta legalización se lleva a cabo después de cada iteración de ajuste de congestión y *timing*.

Según el diseño, en esta etapa podrá reifnarse la colocación de celdas según algoritmos basados en *timing*, congestión, e incluso añadir más capas de metal para el enrutamiento.

### 2.3.3. Clock Tree Synthesis (Síntesis del Árbol de Reloj)

La idea detrás de la síntesis del árbol de reloj (CTS, por sus siglas en inglés) consiste en la inserción automatizada de buffers o inversores en las rutas de reloj del diseño del ASIC, por la cual se conectan todos los componentes secuenciales a su respectivo dominio de reloj. Esto se hace para equilibrar el retardo del reloj en todas las entradas de reloj. Las señales de reloj se consideran redes globales o ideales en este contexto.

A los componentes secuenciales se les llaman "*sinks*" o "*leafs*" (destinos y terminales, en español). En la Figura 27a se ve un soporte gráfico de tales terminales, pudiendo estas ser celdas estándar, memorias, IPs, entre otros.

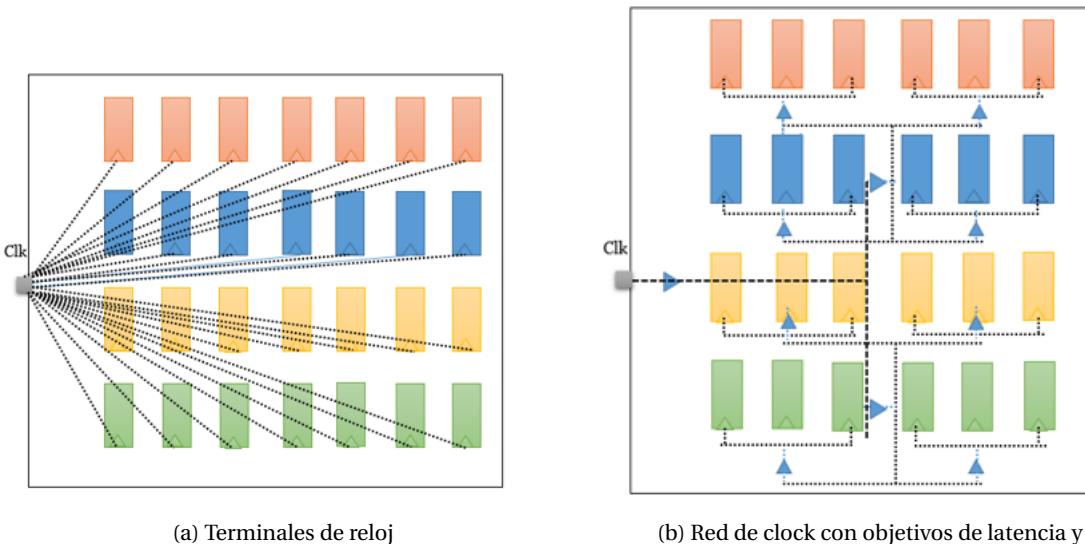


Figura 27: Red de reloj y terminales

Se recomienda que los buffers utilizados para ajustar las rutas de reloj tengan tiempos de retardo de subida y caída iguales. Esto se hace para preservar el ciclo de trabajo original y evitar solapamientos de señales de reloj debido a diferencias en los retardos de propagación.

Cuando se trabaja con relojes de altísima velocidad, si no se eligen adecuadamente los buffers o inversores de reloj, pueden degradar el ancho del pulso del reloj a medida que se propaga a través de ellos antes de llegar a su destino final.

Frecuentemente es difícil lograr un equilibrio perfecto entre los tiempos de subida y caída de la señal en el contexto de un diseño de ASIC durante la síntesis del árbol de reloj. Por lo tanto, un remedio para superar el estrechamiento del pulso de reloj es utilizar inversores en lugar de buffers. En la Figura 28 se observa el efecto de degradación de ancho del pulso de reloj.

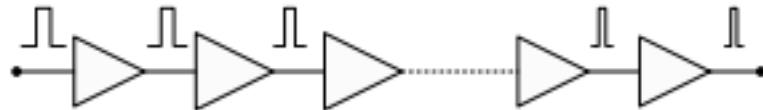


Figura 28: Degradación del ancho del pulso de reloj

La síntesis de la red de reloj es uno de los pasos importantes, ya que consume al menos el 30% del presupuesto total de energía. Para bloques de alto rendimiento, el consumo de energía puede llegar a ser tan alto como el 50% del presupuesto total de energía. Una red de reloj de buena calidad no solo facilitará la convergencia de la temporización, reducirá el consumo de energía, sino que también disminuirá el consumo de recursos de enrutamiento. Esto juega un papel fundamental al elegir los algoritmos para realizar la síntesis de las redes de relojes, los más utilizados para herramientas de colocación y enrutamiento pueden variar según la preferencia, los requisitos específicos y el enfoque (i.e. bajo consumo de potencia). Los enfoques más comunes incluyen:

1. Distribución de reloj en forma de árbol (H-Tree):

- Minimiza el desfasaje (*skew*) y el consumo de energía al dividir el reloj en múltiples niveles de búferes e inversores.
- La mayoría de las herramientas EDA modernas, como Cadence Innovus, Synopsis ICC2 y Synopsis Fusion Compiler, admiten la distribución de reloj en forma de árbol H. En la Figura 29, se muestra un Floorplan de una implementación en forma de árbol de reloj de 3 niveles.

2. Malla de reloj (Clock Mesh):

- Utiliza una red de mallas para distribuir el reloj en todo el chip.
- La diferencia entre una red de reloj convencional y una red de malla radica en la utilización de enrutamiento de malla.
- Mejora en el Rendimiento: La distribución uniforme de la señal de reloj puede dar como resultado un mejor rendimiento del chip, ya que reduce la probabilidad de no cumplir con las restricciones de *timing* debido a un exceso de desfasaje del reloj.
- Bajo consumo: La malla de reloj, cuando el bloqueo del reloj (*clock gating*), puede favorecer el diseño con un consumo bajo de potencia. En la Figura , se observa un esquema de una malla de reloj.

3. Síntesis de árbol de reloj personalizado :

- En algunos casos, especialmente para diseños altamente personalizados, los diseñadores pueden optar por la síntesis de árbol de reloj manual o personalizada.
- Este enfoque les brinda un control total sobre la arquitectura del árbol de reloj, lo que puede ser esencial para cumplir con requisitos de diseño específicos.
- La síntesis de árbol de reloj personalizado a menudo implica la creación de secuencias de comandos y la optimización dentro de la herramienta de síntesis de árbol de reloj proporcionada por los proveedores de EDA.

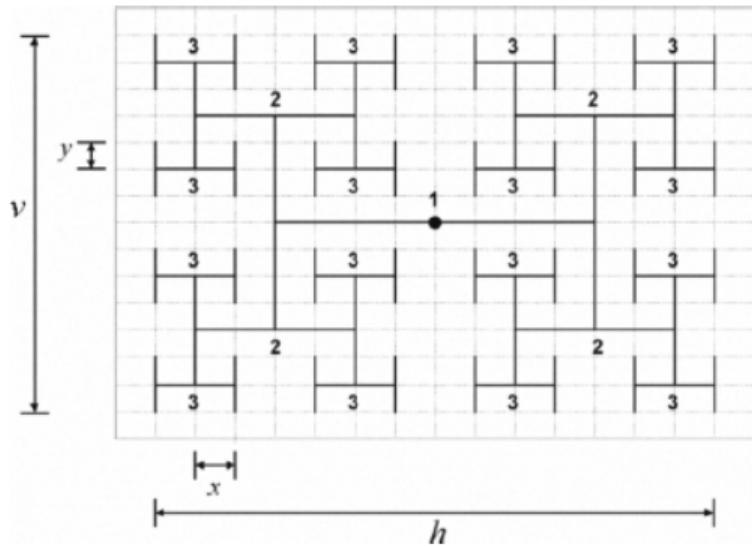


Figura 29: Distribución en forma de árbol de reloj

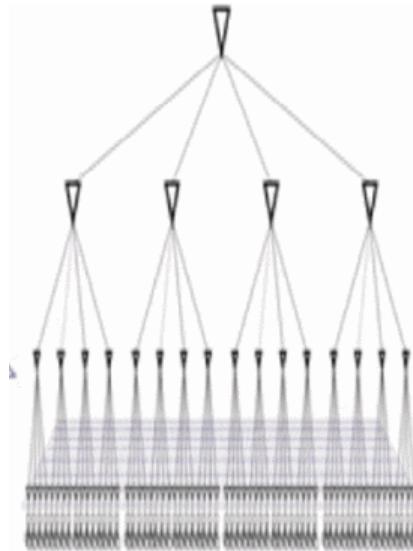


Figura 30: Distribución en forma de malla de reloj

El objetivo que determina la calidad del árbol de reloj, independientemente de la configuración utilizada para insertar buffers durante la síntesis del árbol de reloj, es lograr una baja dispersión (*skew*) mientras se mantiene una propagación aceptable de la señal de reloj. La desviación en el tiempo o la dispersión entre los registros finales y la fuente del reloj puede ser declarada como:

$$\delta = t_2 - t_1$$

Donde  $\delta$  es la dispersión (*skew*) del reloj entre dos registros finales (conectados en el último nivel del árbol de reloj) con una propagación del reloj de  $t_1$  y  $t_2$  a lo largo de dos rutas de reloj diferentes, como se muestra en la Figura 31.

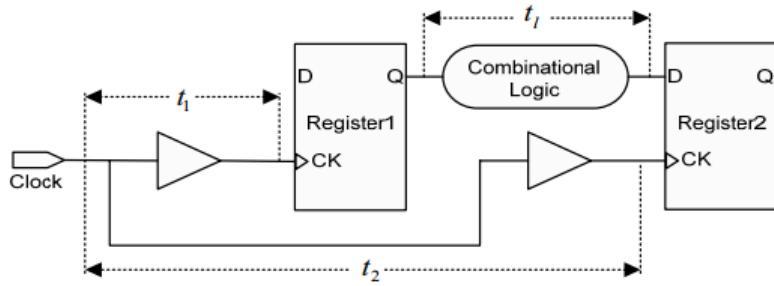


Figura 31: Dispersión (skew) entre dos registros

Tener una latencia pequeña y una dispersión cercana a 0 no es posible en todos los casos. La manera más sencilla de balancear todos los terminales del diseño es agregando retardo a los caminos más rápidos. Esto puede lograrse utilizando celdas estándar creadas para ese destino. Si bien esta manera disminuye la dispersión, aumenta la latencia y la cantidad de celdas totales del diseño, incrementando así también el consumo de potencia interna usada en la red de reloj.

Cualquier problema de temporización, como el tiempo de configuración de datos y el de retención en relación con el reloj de cualquier registro, debe corregirse una vez que se haya completado la síntesis del árbol de reloj. Los mejores resultados se obtienen cuando se utilizan datos de enrutamiento global y configuraciones de librería adecuadas (por ejemplo, una librería de caso lento para el tiempo de configuración y una librería de caso rápido para las violaciones de tiempo de retención).

Las transiciones lentas en la red de reloj pueden hacer que los caminos de reloj sean más vulnerables al **ruido**. Las transiciones lentas pueden hacer que la ventana de captura (el período de tiempo durante el cual un dato debe estar estable para que se capture correctamente) sea más larga. Esto significa que hay más oportunidades para que el ruido impacte la señal dentro de esa ventana más amplia, lo que aumenta las posibilidades de errores de sincronización. También, pueden requerir más energía para cargar y descargar capacitancias en las líneas de reloj. Esto puede aumentar el consumo de energía y generar más ruido en el sistema.

A su vez, si las transiciones son muy rápidas, podrían actuar como agresoras de otras señales de reloj o de datos.

#### 2.3.4. Enrutamiento

Después de completar la colocación de celdas estándar y el análisis de potencia, el siguiente fase es encaminar el diseño ASIC y realizar la extracción de enrutamiento (o ruteo) y parámetros parásitos para el análisis y la simulación de tiempos estáticos .

En esta etapa, se realiza el enrutamiento detallado de todas las nets del diseño implementado, a excepción de las nets del reloj, que para esta etapa llegarán ya ruteadas como se mencionó anteriormente.

El enrutamiento o ruteo se vuelve más sofisticado y difícil a medida que los diseños de ASIC se vuelven más grandes y más complejos. Es posible que el enrutamiento tome un tiempo inaceptablemente largo para ejecutarse o fallar por completo. Además de los algoritmos de ruteo, los factores que pueden influenciar la rutaabilidad de un ASIC dado son el *layout* de las celdas estandar, un floorplan bien preparado y la calidad del placement o colocamiento de las celdas.

Tiempo atrás, cuando las capas de metales estaban en orden de las unidades, se utilizaba el algoritmo de canal. Dado el número limitado de capas de enrutamiento, todas las conexiones se restringían al área entre celdas o alrededor de los bloqueos des macros (i.e. memorias). Esta técnica utiliza los espacios reservados entre las filas de las celdas estandar y las áreas *feed-through* (áreas dedicadas dentro del *layout* de la celda). En la Figura 32 se puede observar este método.

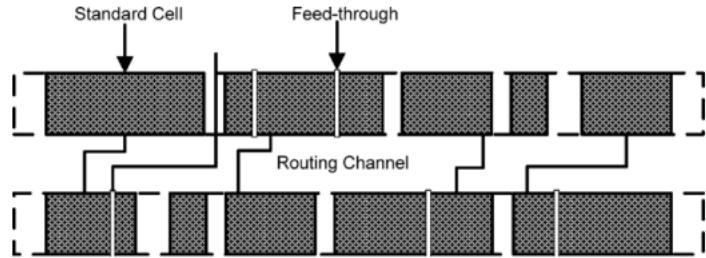


Figura 32: Método de ruteo de canal

Dada la complejidad de los diseños y el enorme número de interconexiones asociado, la bibliografía especializada en el tema define que el enruteamiento se realiza en varias etapas siendo las más esenciales ruta global y ruta detallada.

Primeramente, deben de tenerse en cuenta la configuración de las opciones para la herramienta que realizará el enruteamiento, por ejemplo: Si se realizarán arreglos de violaciones de antena, si se tendrá en cuenta el *timing* al momento de realizar el enruteamiento global cantidad de iteraciones máximas vs cantidad de DRC, entre otros.

El **enruteamiento global** es la descomposición de las interconexiones del diseño ASIC en segmentos de red y la asignación de estos segmentos de red a regiones sin especificar sus diseños reales. Por lo tanto, el primer paso del algoritmo de enruteamiento global es definir regiones o celdas de enruteamiento (es decir, un área rectangular con terminales en todos los lados) y calcular su densidad de enruteamiento correspondiente. Estas regiones de enruteamiento son comúnmente conocidas como Celdas de Enruteamiento Global (GRC por sus siglas en inglés *Global Routing Cells*). En la Figura 33, se muestran tales celdas.

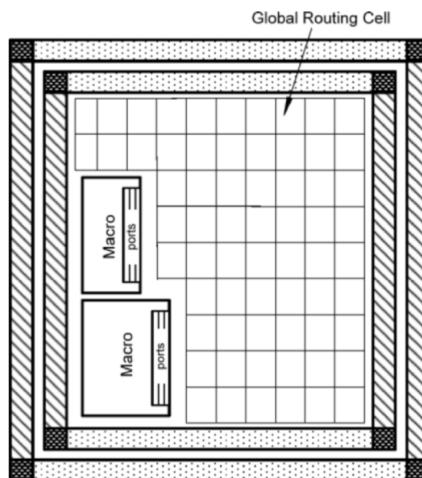


Figura 33: Celdas de Enruteamiento Global (GRC)

La densidad o capacidad de estas celdas se define como el número máximo de redes que cruzan las regiones de enruteamiento y es una función del número de capas de enruteamiento, la altura de la celda en dirección vertical u horizontal, el ancho mínimo y el espaciado del cable según lo definido en el archivo de tecnología, y se expresa como:

$$C_d = \frac{nh}{w+s}$$

Donde  $C_d$  es la densidad de enruteamiento global,  $n$  es el número de capas disponible para la dirección horizontal o vertical,  $h$  es la alutura de la longitud de la celda de enruteamiento global y  $w$  y  $s$  corresponden a la anchura mínima del cable y espaciado para direcciones verticales u horizontales.

El enruteamiento global utiliza un grafo para modelar las redes de interconexión. Los vértices del grafo representan los puertos de celdas estándar. Los bordes del grafo corresponden a conexiones entre dos puertos

dentro de las celdas de enrutamiento y entre las propias celdas de enrutamiento. El grafo se construye mediante la asignación de regiones (es decir, el orden en que se enrutará una región primero) y la asignación de cada red a un pin en el límite de la región de enrutamiento. En la Figura 34 se observa un grafo utilizado en el algoritmo del método *Complete Graph Wire Length Estimation*", uno de los tantos métodos.

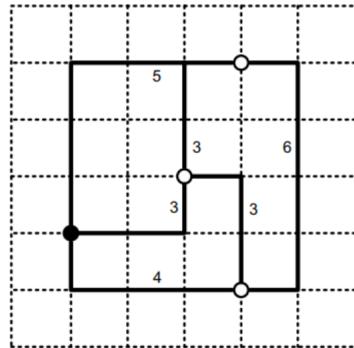


Figura 34: Grafo, sus vértices y bordes

Después de realizar el enrutamiento global, se determinarán las ubicaciones de los pines de manera que la conectividad entre todas las celdas estándar en el área central del ASIC sea mínima. Casi todos los enrutadores globales reportan la estadística de ruteo del diseño utilizando desbordamiento o subflujo para las Celdas de Enrutamiento Global (GRC), que es la relación entre la capacidad de las celdas de enrutamiento y el número de redes requeridas para enrutar una celda de enrutamiento dada en todas las capas de enrutamiento vertical y horizontal. La estadística GRC es una indicación muy buena de la congestión de cables y muestra el número de redes necesarias para enrutar una región versus el número disponible de capas de enrutamiento.

Existen varias maneras de estimar el largo de cableado para cada red del diseño. Los distintos algoritmos de ruteo global usan algún método e estimación para computarlo y extraer los valores parásitos.

El objetivo del **enrutamiento detallado** es seguir el enrutamiento global y realizar las interconexiones físicas reales del diseño ASIC. Por lo tanto, el enrutador detallado coloca los segmentos de cable reales dentro de la región definida por el enrutador global para completar las conexiones requeridas entre los puertos y realizar las conexiones punto a punto de todas las redes. Además, a partir de iteraciones, se encarga de obtener la menor cantidad de violaciones de reglas de diseño (aquellas que se cargaron junto al archivo de tecnología), asegurando así un diseño funcional y eficiente. Algunas de estas reglas de diseño pueden ser: dirección vertical u horizontal, cantidad de máscaras disponibles, distancia mínima entre dos cables del mismo metal, entre otras.

Los enrutadores detallados utilizan tanto cuadrículas de enrutamiento horizontal como vertical para el enrutamiento real. Las cuadrículas de enrutamiento horizontal y vertical están definidas en el archivo de tecnología para todas las capas que se están utilizando.

El ruteo detallado del tipo basado en la cuadrícula (*grid-based*) impone una grilla o cuadrícula de ruteo con tracks igualmente separados tanto vertical como horizontalmente a través del área de diseño que todos los segmentos de ruteo deben seguir. Se permite que el ruteador cambie dirección en la intersección de los tracks verticales con los horizontales, como se indica en la Figura 35.

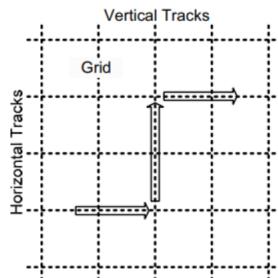


Figura 35: Ruteo basado en grilla

Existen diversos algoritmos de enrutamiento detallado. En ellos, a menudo, se utilizan la combinación de varias heurísticas y optimización para abordar restricciones de diseño específicas, minimizar la longitud de cable, reducir la congestión, etc. Cada uno tiene sus fortalezas y aplicabilidad a diferentes escenarios dentro del proceso de PnR.

El objetivo del enrutador detallado durante esta etapa del enrutamiento es completar toda el área de enrutamiento sin conexiones abiertas entre terminales, incluso si las conexiones causaran violaciones de las reglas de diseño o cortocircuitos. La siguiente fase del enrutamiento detallado se conoce como búsqueda y reparación. Durante esta fase, el enrutador detallado comienza a resolver todo tipo de violaciones de reglas de diseño físico, como el espaciado entre metales, las muescas de relleno y los cortocircuitos de cables. Por lo general, después de esta etapa, todo el diseño de ASIC se routea completamente sin violaciones de reglas de diseño físico.

Después de completar el enrutamiento y la búsqueda y reparación y dada la gran cantidad de , se recomienda la minimización de vías y la optimización del diseño, especialmente para diseños físicos en submicrones. La minimización de vías se refiere al proceso de eliminar tantas vías como sea posible al reducir el número de giros asociados con una conexión de cableado. Esta minimización de vías no sólo reduce la resistencia del cableado sino que también proporciona un mejor rendimiento en términos de procesamiento de vías. Así, además, se ofrece una mejor inmunidad contra electromigración.

Es posible que se corra una etapa de optimización posterior a las anteriores de enrutamiento. En esta etapa, se podrán realizar ruteos incrementales basándose en optimizaciones en pos de arreglar violaciones de *timing* de tiempos de *setup* agregando buffers o removiendo celdas, por ejemplo. También en pos de arreglar violaciones de *timing* de *hold*, agregando celdas de delay.

Entre otras optimizaciones también se pueden ver aquellas realizadas para arreglar cortos y nets abiertas, agregar NDRs (*non-default rules*) en metales de redes, arreglar violaciones de antena por la acumulación de cargas en las compuertas de los transistores en el proceso de metalización, entre otras.

## 2.4. Signoff

El diseño aún no está listo para ser enviado a fabricación luego de terminar el Place and Route. Antes del *tape-out*, debe pasar por las verificaciones de *sign-off* que aseguran su manufacturabilidad, equivalencia funcional y confiabilidad eléctrica. Entre ellas se incluyen:

- **DRC/LVS:** comprobación de reglas de diseño y correspondencia entre el layout y el netlist lógico.
- **Extracción parasítica (PEX):** obtención de resistencias y capacitancias reales de interconexiones.
- **Análisis estático de temporización (STA):** verificación final del cumplimiento de tiempos con parasitismos extraídos.
- **Análisis de IR drop y electromigración (EM):** evaluación de estabilidad de la red de alimentación y confiabilidad a largo plazo.
- **Verificación formal (FEC):** comprobación de equivalencia lógica entre el netlist funcional y el netlist físico.

Una vez superadas estas verificaciones, el diseño se considera listo para la generación de los archivos de salida (*GDSII/OASIS*) y su posterior envío a la *foundry* para fabricación.

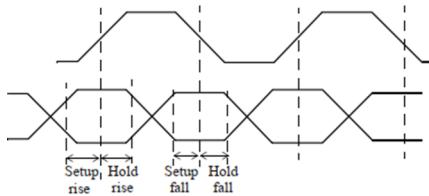
### 2.4.1. Static Timing Analysis

En pos de entender por qué el éxito en temporización de la implementación del diseño es importante, se explicarán dos conceptos claves del análisis de tiempo estático (STA).

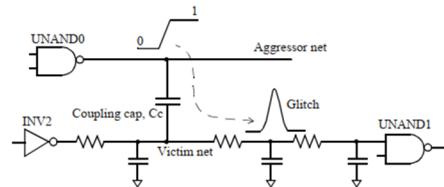
Es un método para validar el rendimiento de *timing* de un diseño mediante la comprobación de todas las rutas posibles para violaciones de tiempo en las peores condiciones. Considera la peor latencia posible a través de cada elemento lógico, pero no el funcionamiento lógico del circuito.

El análisis busca responder a la pregunta, "¿Estarán presentes los datos correctos en la entrada de datos de cada dispositivo sincrónico cuando llegue el pulso del reloj, en todas las condiciones posibles?"

Un **setup timing check** verifica la relación de tiempo entre el reloj y el pin de datos de un flip-flop para que se cumpla el requisito de *setup*. En otras palabras, el *setup check* garantiza que los datos estén disponibles en la entrada del flip-flop antes de que se registren en el flip-flop. Los datos deben ser estables durante un cierto período de tiempo, llamado el tiempo de *setup* del flip-flop, antes de que el flanco activo del reloj llegue al flip-flop. Este requisito garantiza que los datos se capturen de forma fiable en el flip-flop. Se ilustra el requerimiento



(a) Setup and Hold



(b) Glitch

Figura 36: Signoff

de *setup* en la Figura 37. En general, hay un flip-flop de lanzamiento, que es el flip-flop que lanza los datos, y un flip-flop de captura, el flip-flop que captura los datos cuyo tiempo de *setup* debe cumplirse. La comprobación de *setup* valida la ruta máxima desde el flip-flop de lanzamiento hasta el flip-flop de captura. Los relojes de estos dos flip-flops pueden ser iguales o diferentes.

La verificación de *setup* es desde el primer flanco activo del reloj en el flip-flop de lanzamiento hasta el siguiente flanco activo más cercano del flip-flop de captura. La verificación de *setup* garantiza que los datos lanzados desde el ciclo de reloj anterior estén listos para ser capturados después de un ciclo.

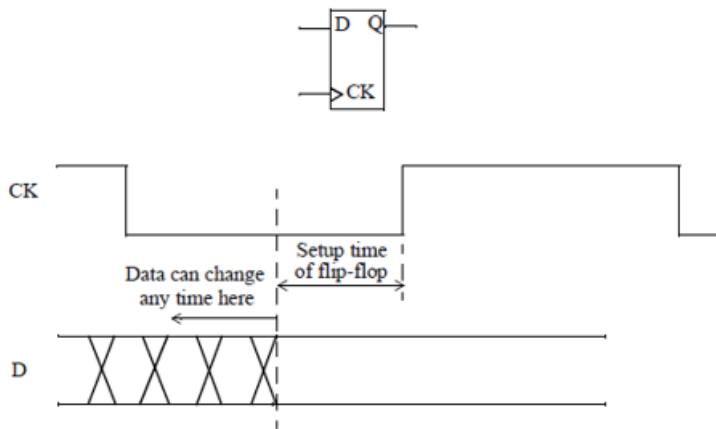


Figura 37: Requerimiento de setup para un flip-flop

La verificación de *setup* se puede expresar matemáticamente de la siguiente forma:

$$T_{\text{launch}} + T_{\text{clk } 2q} + T_{dp} < T_{\text{capture}} + T_{\text{cycle}} - T_{\text{setup}} \quad (1)$$

donde  $T_{\text{launch}}$  es el delay del árbol de reloj del flip-flop de lanzamiento UFF0,  $T_{dp}$  es el delay de la ruta de datos de lógica combinacional y  $T_{\text{cycle}}$  es el período del reloj.  $T_{\text{capture}}$  es el delay del árbol de reloj para el flip-flop de captura UFF1, del ejemplo de la Figura 38.

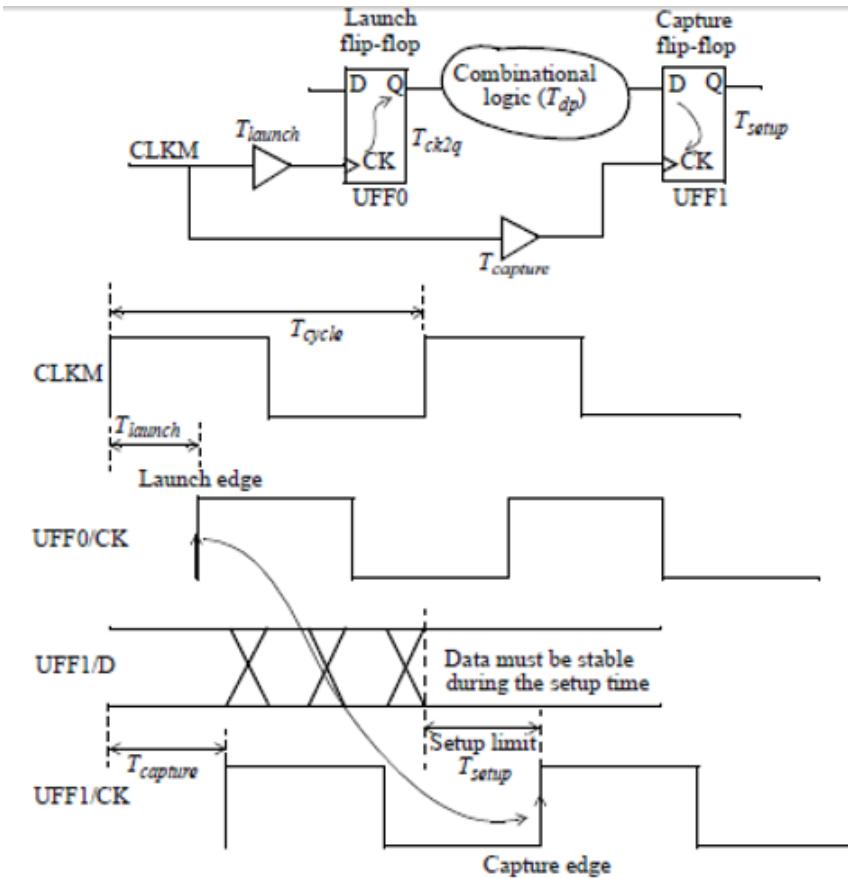


Figura 38: Ejemplo de señales de datos y reloj para setup

Un **hold timing check** garantiza que un valor de salida de un flip-flop que está cambiando no pase a través un flip-flop de captura y sobrescriba su salida antes de que el flip-flop haya tenido la oportunidad de capturar su valor original. Esta verificación se basa en el requisito de retención de un flip-flop. La especificación de *hold* de un flip-flop requiere que los datos que se enlazan se mantengan estables durante un período de tiempo específico después del flanco activo del reloj. La Figura 39 muestra el requisito de *hold* de un flip-flop típico.

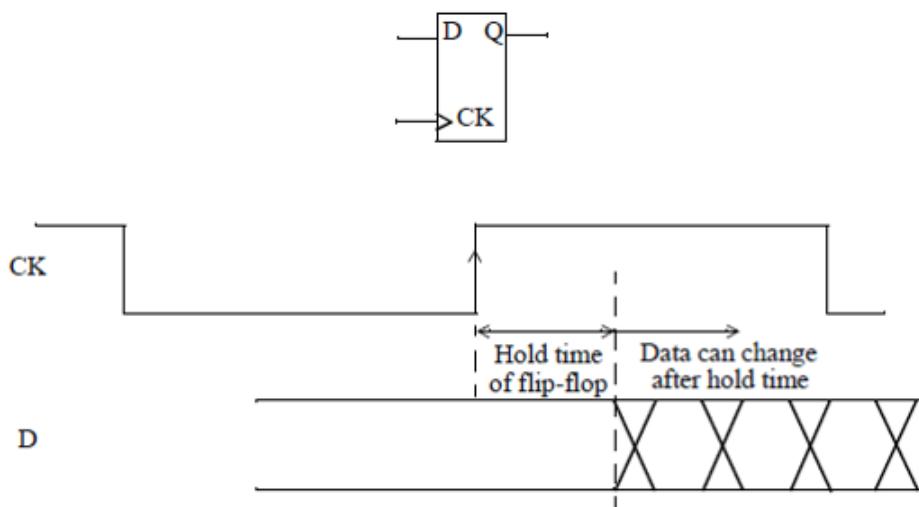


Figura 39: Requerimiento de hold para un flip-flop

El *hold check* puede ser matemáticamente expresado de la siguiente forma:

$$T_{\text{launch}} + T_{\text{ck2q}} + T_{\text{dp}} > T_{\text{capture}} + T_{\text{hold}} \quad (2)$$

donde  $T_{\text{launch}}$  es el delay del árbol de reloj del flip-flop de lanzamiento,  $T_{\text{dp}}$  es el delay en la ruta de datos de lógica combinacional y  $T_{\text{capture}}$  es el delay del árbol de reloj del flip-flop de captura. En otras palabras, el tiempo total requerido para que los datos lanzados por un flanco de reloj lleguen al pin D del flip-flop de captura debe ser mayor que el tiempo requerido para que el mismo flanco del reloj viaje al flip-flop de captura más el tiempo de hold. A modo de ejemplo, se adjunta el caso de la Figura 40.

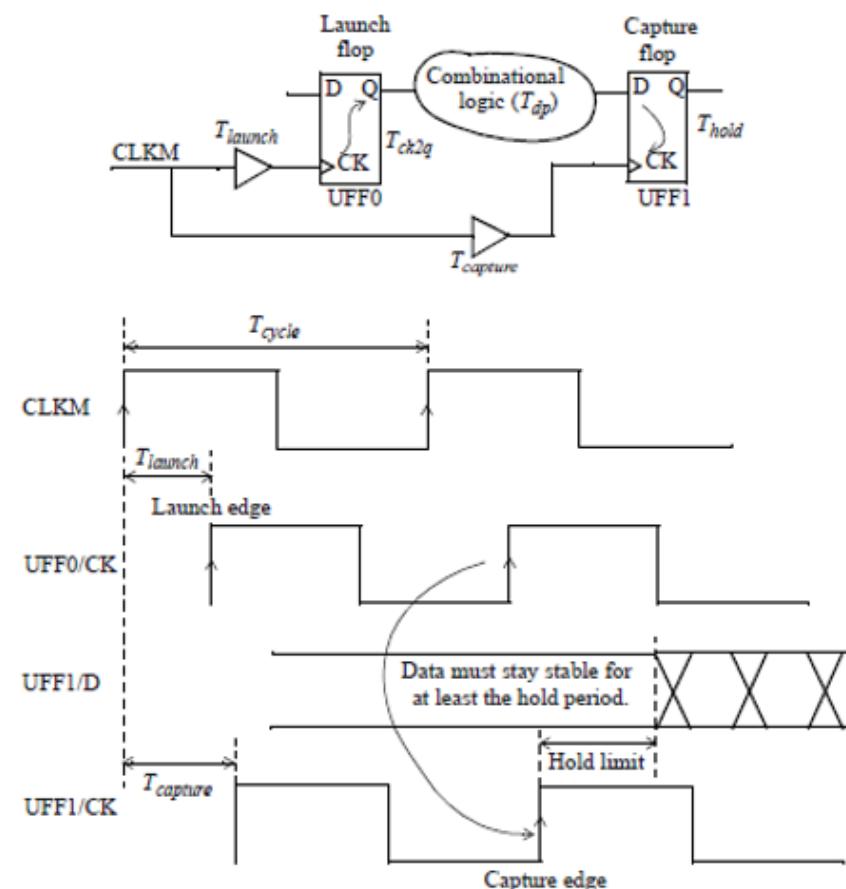


Figura 40: Ejemplo de señales de datos y reloj para un check de hold

### 3. Laboratorio: Colocacion y enrutamiento de una ALU

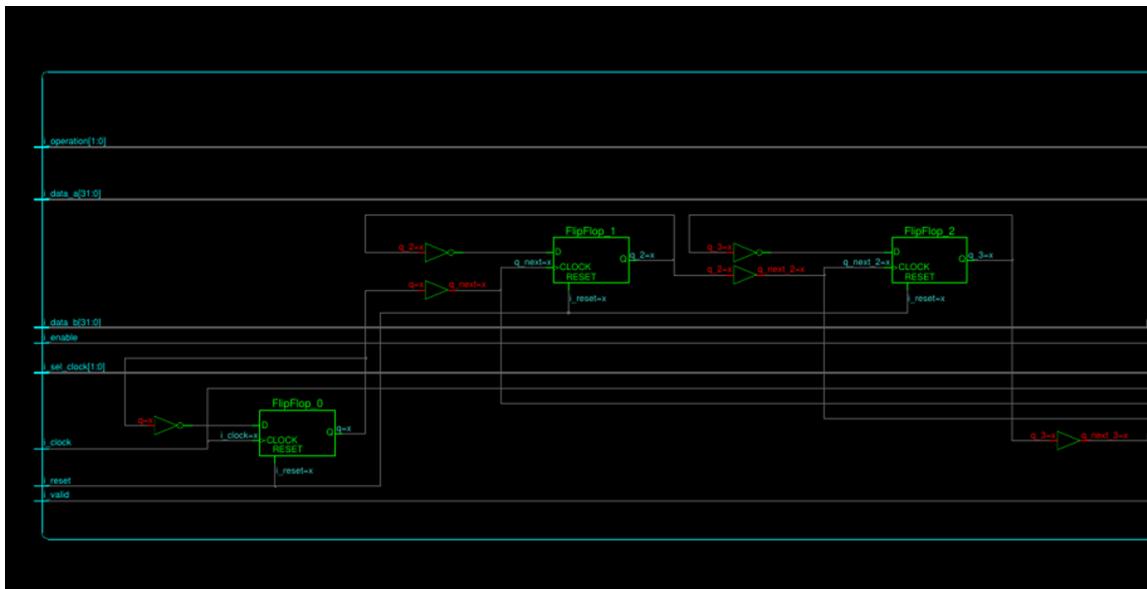
#### Repository del diseño

En este trabajo se utiliza Git, un sistema de control de versiones distribuido, para gestionar de forma eficiente la evolución del proyecto de diseño físico. Git permite llevar un registro detallado de los cambios realizados en los archivos fuente (scripts TCL, descripciones RTL, constraints, informes, figuras, etc.), facilitando tanto el potencial trabajo colaborativo como la trazabilidad individual del flujo de diseño. Todos los archivos fuente del diseño, *testbench*, restricciones y scripts de implementación están disponibles públicamente en:

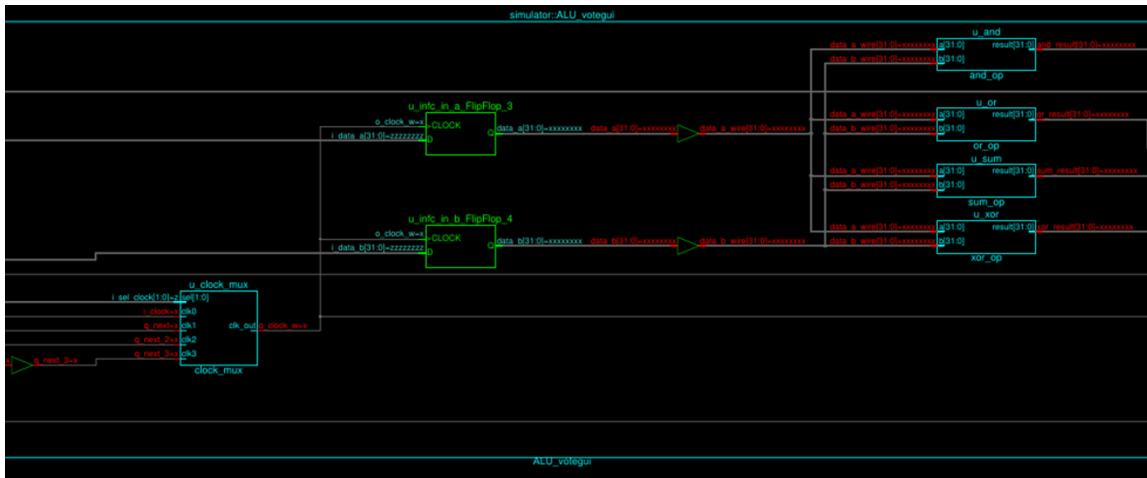
[https://github.com/xtegui/ALU\\_sin\\_inf.c](https://github.com/xtegui/ALU_sin_inf.c)

#### 3.1. Descripción funcional del diseño a implementar

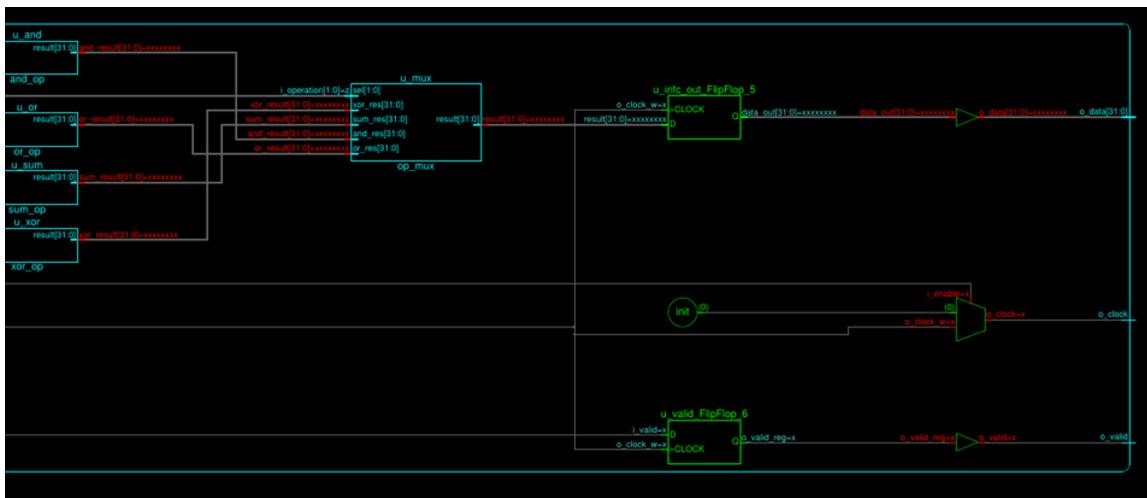
El bloque lógico implementado es una **Unidad Aritmético-Lógica (ALU)** paralela de 32 bit, definida como módulo superior `ALU_no_inf.c`. Esta ALU permite ejecutar operaciones lógicas y aritméticas elementales entre dos operandos, con capacidad de selección dinámica de operación y frecuencia de reloj.



(a) Parte A



(b) Parte B



(c) Parte C

Figura 41: Esquemático funcional del RTL. Muestra la jerarquía de módulos de la ALU, incluyendo operaciones básicas, multiplexores y lógica de división de reloj. (A) Parte 1. (B) Parte 2. (C) Parte 3.

## Entradas y salidas principales

Nombre	Tipo	Descripción
i_data_a, i_data_b	Entrada [31:0]	Operandos principales
i_operation	Entrada [1:0]	Selección de operación
i_sel_clock	Entrada [1:0]	Selección del divisor de reloj
i_clock, i_reset	Entrada	Señales de sincronismo y reinicio
i_valid, i_enable	Entrada	Control de flujo y habilitación de reloj
o_data	Salida [31:0]	Resultado de la operación
o_clock	Salida	Salida de reloj ajustada
o_valid	Salida	Indica validez del resultado

Cuadro 3: Puertos principales del módulo ALU\_no\_infc.

## Funcionamiento general

El diseño integra cuatro módulos funcionales (xor\_op, and\_op, or\_op, sum\_op), todos parametrizados con N\_BITS = 32. Sus salidas se conectan a un multiplexor de operaciones (op\_mux), controlado por la señal i\_operation, según la codificación de la Tabla 4.

i_operation[1:0]	Operación	Módulo activo	Descripción
00	XOR	xor_op	Operación lógica bit a bit
01	AND	and_op	Operación lógica bit a bit
10	OR	or_op	Operación lógica bit a bit
11	SUMA	sum_op	Suma binaria completa

Cuadro 4: Modos de operación definidos por i\_operation.

El reloj utilizado para registrar entradas y salidas es generado por el módulo clock\_mux, que selecciona entre la señal original i\_clock y divisiones sucesivas (i\_clock/2, /4, /8). Esto permite ajustar dinámicamente la frecuencia del pipeline según la configuración externa.

## Restricciones de temporización

Durante la implementación física, se aplica un **reloj virtual** de período 0,8 ns (1,25 GHz) definido sobre el puerto i\_clock. Las señales de entrada y salida están sujetas a retardos de 0,2 ns, y se establecen márgenes de:

- **Slew máximo:** 300 ps. El slew máximo (o maximum slew) es una restricción de diseño en circuitos digitales que define el tiempo máximo permitido para que una señal cambie de un nivel lógico a otro (por ejemplo, de 0V a VDD o de VDD a 0V).
- **Fan-out máximo:** 40
- **Clock uncertainty:** 60 ps. La incertidumbre de reloj refiere a una restricción de temporización que modela la variación esperada en la llegada del reloj a diferentes partes del circuito. Representa un margen de seguridad temporal que se resta del tiempo disponible para las señales de datos en análisis de *setup* y *hold*.

Estas restricciones aseguran que las rutas internas cumplan requisitos de temporización y robustez en los escenarios MCMM (*Multi-Corner, Multi-Mode*) definidos durante síntesis física.

## 3.2. Configuración del flujo de diseño

Para realizar la implementación del RTL de la Unidad Aritmética Lógica, se hará uso de un flujo de diseño instalado en un servidor con sistema operativo Linux Centos 7. Se utilizarán técnicas de programación en el lenguaje Tcl y comandos bash.

Al servidor se accede remotamente y con un escritorio gráfico (necesario para la GUI de la herramienta). Se realiza a través de la herramienta X2Go, con un protocolo SSH y un protocolo NX.

Los archivos de configuración, llamados de forma *config\_<descripcion>.tcl*. Su propósito principal es definir parámetros clave del diseño para que las herramientas los utilicen en las diferentes etapas (*initial\_design*, *compile*, *clock tree synthesis* y *route*).

Se configura la herramienta de manera de definir el nombre del diseño top (ALU\_votegui), que debe coincidir con el nombre del módulo RTL de mayor jerarquía.

Se especifica la ruta del directorio donde se encuentran los archivos *verilog\_list.lst* e *include\_dir\_list.lst*, necesarios para compilar el diseño.

Para la configuración del floorplan:

- Se define el ancho (50.024 µm) y el alto (49.8 µm) del floorplan, ajustado a los múltiplos requeridos por la tecnología de 14nm.
- Se establece un desplazamiento del core (core\_offset) respecto del área total del chip, especificado con precisión por la tecnología.
- Se carga un archivo externo *pin\_constraints.tcl*, donde se definen las ubicaciones de los pines del diseño.

La funcionalidad principal del archivo de configuración de herramienta es definir:

- Recursos computacionales: número de CPUs, memoria, y comandos para paralización de procesos y reducción de memoria.
- Estructura de directorios para resultados: especifica los directorios para almacenar los distintos de archivos generados durante el flujo. Estos archivos pueden ser reportes de las etapas, salidas de archivos procesados (netlists, DEF,GDS), logs de ejecución, y datos y comparaciones de calidad de resultados.
- Estrategias de optimización (*QoR - Quality of Results*): Define el modo de estrategia (*balanced*), la métrica prioritaria (*timing*), permitiendo que la herramienta enfoque sus esfuerzos en mejorar el rendimiento temporal del diseño.
- Parámetros de control de implementación: Permite habilitar o deshabilitar configuraciones avanzadas como optimizaciones de alto esfuerzo en temporización, uso limitado de celdas de umbral de bajo voltaje, entre otras.

El archivo de configuración de tecnología es crítico porque actúa como punto único de verdad que al invocarse inicializa a Synopsys Fusion Compiler con todos los parámetros del nodo de 14 nm, evitando discrepancias entre etapas y permitiendo reproducibilidad ; dentro del script se enumeran el techfile (.tf) que describe capas y reglas de diseño físicas, los abstractos LEF/DEF de celdas estándar e I/O, las bibliotecas de celdas en formato NDM/DB junto con sus Liberty (.lib) NLDM/CCS para análisis de temporización y potencia, las tablas TLU+ (\*.tluplus) e interfaz ITF que vinculan capas con modelos RC para extracción, los layer-map y reglas de coloración para multipatterning, además de ficheros auxiliares como decks de antena DRC, plantillas de red de potencia y listas de spare cells; al centralizar rutas y variables hacia todos estos archivos tecnológicos, el script sincroniza la configuración de la herramienta con el PDK genérico de 14 nm y garantiza que síntesis, floorplanning, CTS, ruteo y sign-off operen con exactamente los mismos supuestos eléctricos y geométricos.

### 3.3. Floorplanning

#### 3.3.1. Integración de las fases *design\_planning.tcl* e *init\_design.tcl*

El flujo RTL→GDSII se inicia con *design\_planning.tcl* (Fase 1) y continúa con *init\_design.tcl*<sup>1</sup> (Fase 2). Ambos scripts comparten las variables declaradas en *config\_tecnologia.tcl*, *config\_design.tcl* y *config\_herramienta.tcl*, lo que garantiza coherencia lógica-física sobre el PDK genérico de 14 nm.

##### Fase 1 — *design\_planning.tcl*

- *Floor-plan*: definición del *die* y del *core*.
- Inserción de filas a la grilla de 14 nm y reservas de *keep-out*.
- Generación de la malla de potencia (*straps* y rieles).
- Colocación preliminar de macros y asignación de pines.

<sup>1</sup>En el repositorio el archivo se llama *init\_design\_edit.tcl*, pero la funcionalidad es la misma.

- Verificación temprana de congestión y exportación del plano físico.

Listing 1: Creación del *floor-plan*

```
create_floorplan \
-die $die_width $die_height \
-core [expr {$core_util*$die_width}] \
[expr {$core_util*$die_height}] \
-offset $core_offset_x $core_offset_y
```

Listing 2: Generación de *straps* de VDD/VSS

```
create_power_straps \
-nets {VDD VSS} \
-layers $pg_layers \
-width $pg_width \
-pitch $pg_pitch
compile_pg_netlist
```

Listing 3: Exportación del *floor-plan*

```
write_floorplan outputs/floorplan.tcl
```

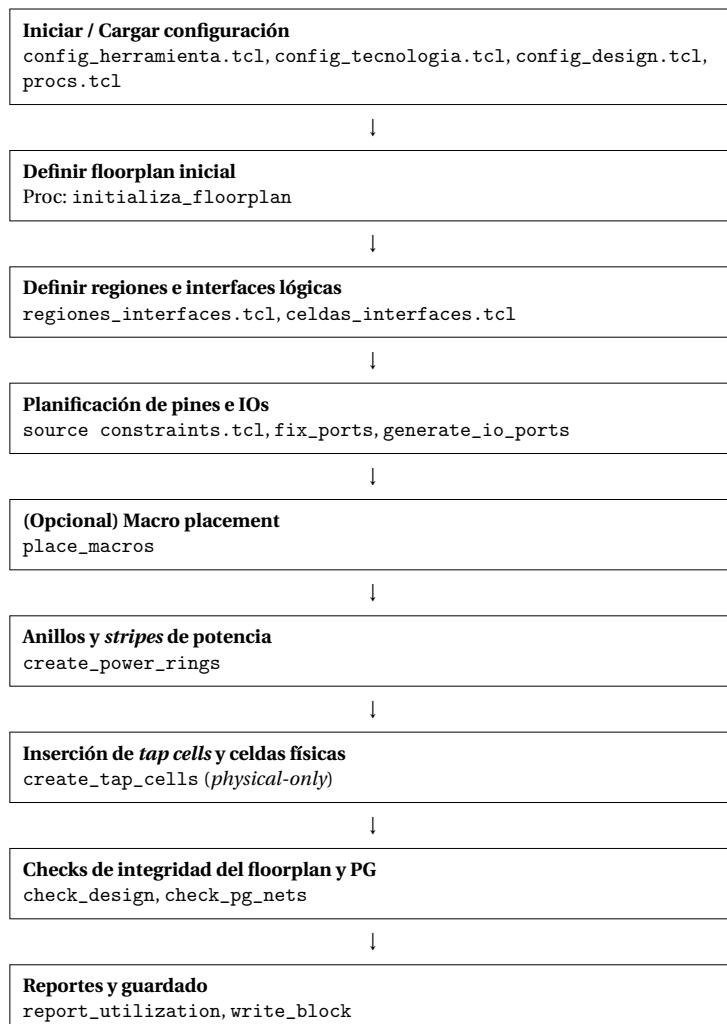


Figura 42: Flujo de *Floorplan* y preparación de potencia en el diseño físico del ALU

Cuadro 5: Etapas resumidas de Design-Planning y los comandos claves de Fusion Compiler.

#	Etapa	Comandos principales	Propósito / Objetivo
0	<b>Carga de configuración</b>	source config_tool.tcl source config_tech.tcl source config_design.tcl	Variables de entorno, bibliotecas y opciones de herramienta.
1	<b>Inicialización del floorplan</b>	initialize_floorplan	Tamaño de core, márgenes y filas de celdas estándar.
2	<b>Planificación de pines</b>	source pin_constraints.tcl fix_ports -auto, write_io_ports	Colocar pines I/O según reglas de encapsulado y guardar snappy.
3	<b>Colocación de macros</b>	place_macros -effort high	Insertar macros/IP respetando halos, bloqueos y orientación.
4	<b>Red PG inicial</b>	create_power_rings create_power_stripes	Anillos (M9–M8) y stripes de alimentación (M7–M6).
5	<b>Tracks y celdas tap</b>	create_tracks create_tap_cells -double_density	Reservar pistas de ruteo y distribuir tap / endcap.
6	<b>Comprobaciones tempranas</b>	check_design -floorplan check_pg_nets	Verificar DRC, utilización y continuidad de la red PG.
7	<b>Reportes y guardado</b>	report_utilization write_block -hier all	Documentar área/uso y salvar .ndm + .fp.

#### Fase 2 — init\_design.tcl

- Lectura de la RTL y de las bibliotecas .ndm/.lib.
- Creación de vistas PVT y modelos de parasitaje ( $C_{max}/C_{min}$ ).
- Aplicación de relojes y restricciones de temporización.
- Síntesis lógica optimizada con políticas multibit y *fan-out/slew* de config\_herramienta.tcl.
- Exportación del *netlist* limpio para las fases de CTS y ruteo.

Listing 4: Lectura y enlace del diseño lógico

```
set search_path "$rtl_dir_$lib_path_$pdk_path"
read_file -format verilog $rtl_files
link_design -top $design_name
```

Listing 5: Definición de análisis *setup/hold*

```
create_library_set -name setup_lib -timing $ndm_ss0p72v125c
create_rc_corner -name setup_rc -tluplus $tluplus_Cmax
create_analysis_view -name setup_ss0p72v125c \
    -library_set setup_lib -rc_corner setup_rc
```

Listing 6: Síntesis lógica en 14 nm

```
compile_ultra \
    -gate_clock \
    -use_multibit_cells $use_multibit \
    -no_autoungroup
```

Listing 7: Escritura del net-list optimizado

```
write_design -hier -design_top $design_name \
    outputs/${design_name}_postsyn.db
```

**Resumen.** design\_planning.tcl establece el entorno físico (geometría, malla de potencia y pines), mientras que init\_design.tcl genera un net-list lógico optimizado bajo los escenarios PVT de trabajo. Ambos scripts aprovechan las mismas variables globales, evitando divergencias entre las fases lógica y física del flujo RTL→GDSII.

### 3.3.2. Parámetros esenciales del proyecto

<b>Nombre lógico</b>	ALU_votegui
<b>Funcionalidad RTL</b>	ALU paralela de 32 bit con XOR, suma, AND, OR y divisor de reloj programable ( $f, f/2, f/4, f/8$ ).
<b>Nodo / PDK</b>	SAED 14 nm CMOS (s14, 1P9M, M1–M9).
<b>Techfile Milkyway</b>	saed14nm_1p9m_mw.tf
<b>Librerías NDM</b>	RVT, HVT, LVT, SLVT + versiones <i>low-V<sub>DD</sub></i> .
<b>PG nets</b>	VDD (power) · VSS (ground)
<b>Capas con DPT</b>	M1–M2 ( <i>double pattern</i> )
<b>Straps PG</b>	M9: 0,60 μm / 6 μm
<b>Rails locales</b>	M1: 0,094 μm
<b>Tap-cell</b>	SAEDRVT14_TAPPN, espaciamiento 80 μm
<b>Site de colocación</b>	unit (simetría {unit Y})
<b>Estrategia QoR</b>	balanced_timing, multibit activado
<b>Lím. transición</b>	≤ 300 ps
<b>Lím. fan-out</b>	≤ 40
<b># CPU para compilación</b>	8

### Parámetros geométricos

- **Dimensión del die:** 50,024 μm × 49,800 μm
- **Offset del core:** 1,184 μm (horizontal) · 1,200 μm (vertical)
- **Utilización inicial del core:** valor core\_util de config\_design.tcl

### Escenarios de temporización (MCMM)

Escenario	Proceso	$V_{DD}$	Temperatura	Parasitaje
<b>setup</b>	SS	0,72 V	125 °C	$C_{max}$
<b>hold</b>	FF	0,88 V	125 °C	$C_{min}$

Cuadro 6: Condiciones PVT definidas en mcmm\_setup.tcl.

### Extracto representativo del RTL

Listing 8: Cabecera del módulo ALU\_votegui

```
module ALU_votegui
  #(parameter N_BITS = 32)
(
  // Salidas
  output [(N_BITS-1):0] o_data,
  output                  o_clock,
  output                  o_valid,
  // Entradas
  input [1:0]              i_operation,
  input [1:0]              i_sel_clock,
  input                   i_enable,
  input                   i_clock,
```

```

    input  [N_BITS-1:0]      i_data_a,
    input  [N_BITS-1:0]      i_data_b,
    input                      i_valid,
    input                      i_reset
);
/* ... logica de divisores de frecuencia y operaciones ALU ... */
endmodule

```

### Relación con los scripts de flujo

1. `design_planning.tcl` — Construye el *floor-plan*: define die/core, inserta filas, genera la malla de potencia (M9 + M1) y distribuye pines conforme a `pin_constraints.tcl`. Valida congestión inicial.
2. `init_design.tcl` (alias `init_design_edit.tcl`) — Carga este RTL, enlaza las bibliotecas SAED-14 nm, aplica los dos escenarios MCMM y genera un net-list lógico optimizado que respeta los límites de *slew/fan-out* y las políticas multibit de `config_herramienta.tcl`.

#### 3.3.3. Resultados de la implementación

##### 1. Resumen del *floor-plan*

Métrica	Valor	Fuente del log
Utilización inicial del core	1.90 %	<code>report_floorplan</code>
Nº total de pines	106	<code>place_pins</code>
Boundary cells insertadas	450	<code>create_boundary</code>
Tap cells insertadas	81	<code>create_tap_cells</code>
Straps PG (M9) / Rails (M1)	15 / 80	<code>create_power_straps</code>

Cuadro 7: Resumen geométrico y de red de potencia del diseño.

Una captura PNG del floorplan inicial, con sus tap-cells y ubicacion de los pines se muestra en la Figura 43.

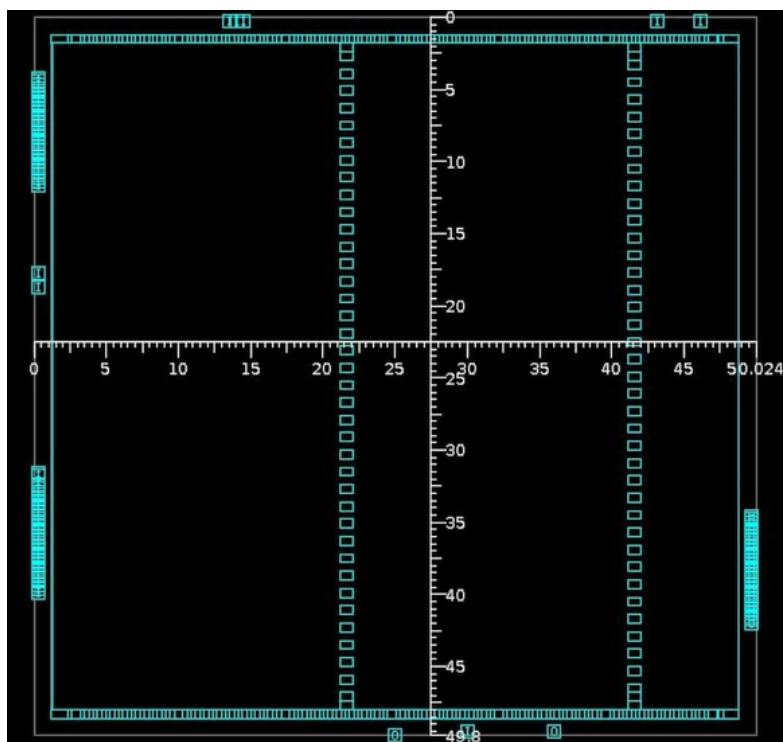


Figura 43: Floor-plan del bloque `ALU_votegui`. Las reglas resaltan la altura del *die* ( $49,8\mu\text{m}$ ) y el ancho ( $50,024\mu\text{m}$ ).

## 2. Conexión de alimentación (VDD) y masa (VSS)

Red	Pines conectados (previo/final)	Reporte
VDD	0 / 531	
VSS	0 / 531	Power/Ground Connection Summary

Cuadro 8: Conectividad completa de PG tras la inserción automática.

En la Figura 44 , se muestra resaltado en amarillo la conexión de los *stripes* en la capa numero 9 de metal (M9, vertical) de alimentación VDD. De la misma forma, en rojo se aprecia el ruteo de VSS. Los *rails* horizontales corresponden a M1.

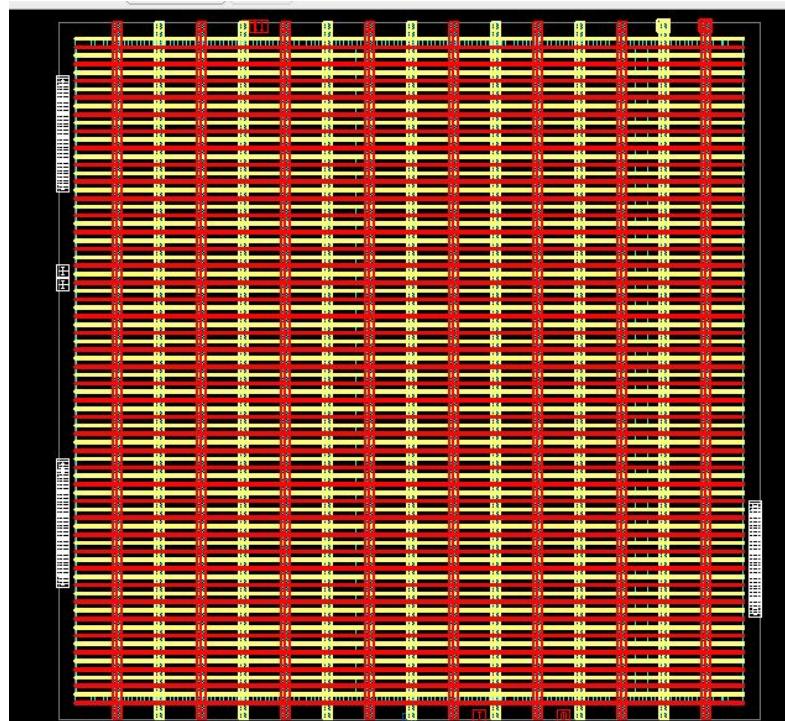


Figura 44: Conexión de *stripes* en M1 y M9 de VDD y VSS

Este ruteo superior en M9 de la PG luego desciende en capas a través de VIAS hasta M1. Los pines de las celdas estándar y macros utilizaran esta malla para alimentar sus pines de VDD y VSS.

En la Figura 45, se puede observar un conjunto de vias superpuestas de la Power Grid.

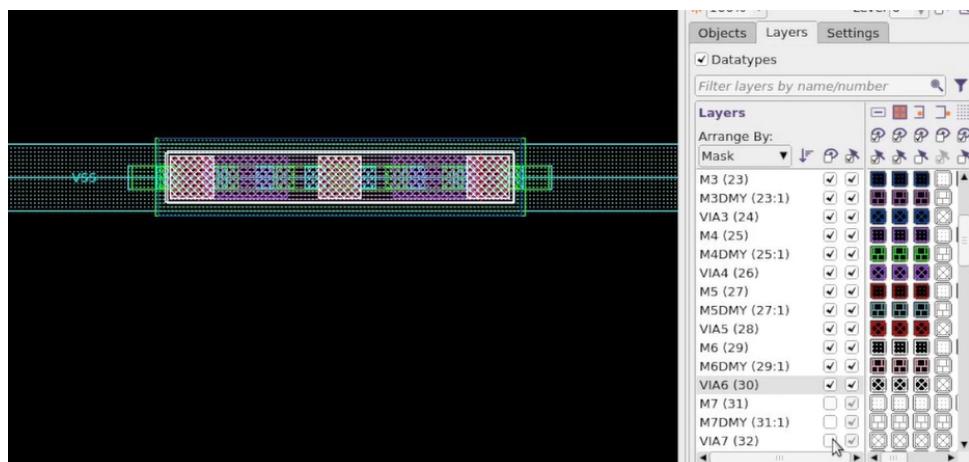


Figura 45: VIA0 hasta VIA6 de la net VSS, apiladas

### 3. Celdas *Tap Filler* y *Boundary*

Las **tap fillers** son celdas de relleno sin lógica que el flujo inserta automáticamente entre celdas funcionales para cubrir los espacios vacíos en las filas. Su inclusión es necesaria por las siguientes razones:

- **Continuidad de pozos (well taps).** Garantizan el contacto periódico del substrato y del *N-Well/P-Well* con las redes de VDD/VSS para prevenir *latch-up* y asegurar la integridad eléctrica del cuerpo de los transistores.
- **Cumplimiento de reglas de diseño (DRC).** Rellenan huecos que de otro modo violarían reglas mínimas de densidad de difusión o de continuidad de pozos impuestas por la tecnología.
- **Extensión de la red de potencia.** Algunas variantes (*tap fillers with PG*) añaden straps de VDD/VSS a lo largo de la fila, evitando discontinuidades en el ruteo de alimentación.

Las **boundary cells** (también llamadas *edge* o *endcap cells*) son celdas especiales que se colocan de forma automática o manual a lo largo del perímetro del bloque lógico para asegurar una correcta transición entre la matriz de celdas estándar y los límites físicos del chip. Sus funciones principales son:

- **Continuidad eléctrica y mecánica**  
Mantienen la integridad de las redes de alimentación (VDD/VSS) en los bordes y evitan dislocaciones de silicio que puedan generar esfuerzo mecánico o delaminación.
- **Cumplimiento de reglas de diseño (DRC)**  
Ajustan las distancias de *nwell/pwell*, *diffusion*, metal y *implant* en los extremos para que las reglas de la tecnología no queden violadas allí donde termina la última celda estándar.
- **Protección contra latch – up y ESD**  
Integran taps o guard rings adicionales, de modo que las regiones de pozo y substrato en el borde queden bien sujetas a potencial fijo, reduciendo la ganancia del parásito *pnpn* y proporcionando un primer nivel de contención para descargas electrostáticas.
- **Alineación y densidad**  
Garantizan que el *row gap* y la rejilla (*row/site grid*) queden cerrados, evitando huecos que puedan complicar la colocación de filler cells o de rutas de señal.

**Inserción automática (Fusion Compiler)** En Fusion Compiler la inserción se realiza típicamente al final de *place\_opt* o mediante

```
create_boundary_cells -lib CELL_LIB \
                      -type {endcap tap corner} \
                      -add_pins_on_rails
```

lo cual genera e inserta cada categoría en su posición óptima respetando reglas de “keep-out” y de discontinuidad de voltaje.

### 4. Definición MCMM (esquinas PVT)

Esquina	Proceso	V <sub>DD</sub>	T [°C]	Parasitaje
setup_ss0p72v125c	SS	0.72 V	125	C <sub>max</sub>
hold_ff0p88v125c	FF	0.88 V	125	C <sub>min</sub>

Cuadro 9: Esquinas creadas mediante *mcmm\_setup.tcl*.

### 5. Relojes y restricciones

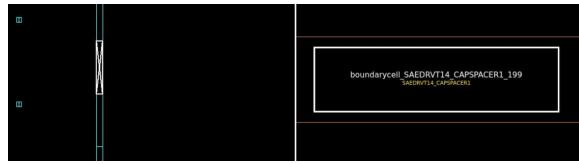
Habiendo definido las restricciones temporales previamente en este informe, aquí se resume los valores de tales archivos:

- **Reloj virtual:** *vir\_CLOCK\_MAIN* definido para *input/output delays*.
- **Reloj generados:** *q\_div\_2\_reg*, *q\_div\_4\_reg*, *q\_div\_8\_reg*;
- **Agrupación de paths:** grupos *in2reg*, *reg2out*, *in2out*; se aplican los límites globales *slews* ≤ 300 ps y *fan-out* ≤ 40 establecidos en *config\_herramienta.tcl*.

Para mas información sobre los archivos *constraints*, revisar la sección 2.2.2.



(a) Distribución de *tap fillers*.



(b) Celdas de borde (*boundary cells*).

Figura 46: Vista detallada del borde del bloque: (a) inserción regular de *tap fillers* para anclar pozo y substrato; (b) *boundary cells* que aseguran continuidad de pozos y líneas de potencia, además de cumplir reglas DRC en el perímetro.

### 3.4. Colocación

La etapa de *placement* constituye el primer paso puramente físico dentro del flujo de *place & route*. Su objetivo es transformar el netlist lógico —obtenido tras la síntesis de la ALU aritmético-lógica de 32 bits— en una distribución legal de celdas estándar sobre el silicio, cumpliendo simultáneamente tres metas fundamentales:

1. **Cierre de temporización preliminar.** Las celdas se ubican de forma que las rutas críticas queden lo bastante cortas para satisfacer la frecuencia objetivo (100 MHz) con *slack* no negativo antes de la inserción del árbol de reloj.
2. **Minimización de área y congestión.** La herramienta agrupa lógicamente bloques relacionados, controla la densidad y reserva canales de enrutamiento para evitar *hot-spots* y solapes que compliquen las fases posteriores de ruteo.
3. **Preparación para CTS y optimización posterior.** Se generan *checkpoints* intermedios que capturan decisiones clave—mapa inicial, legalización, optimización temprana—y se etiquetan las redes de reloj como ideales, sentando las bases para la posterior construcción del árbol de reloj y las iteraciones de *place opt*.

En el contexto de este proyecto académico, la ALU presenta una complejidad moderada (2000 puertas NAND equivalentes), lo que permite ilustrar el algoritmo industrial de Synopsys Fusion Compiler sin exceder recursos de cómputo. La correcta ejecución de la colocación asegura que el diseño llegue a la etapa de síntesis de reloj con un piso libre de violaciones geométricas, con *timing* preliminar cerrado y con métricas de área y potencia alineadas a los objetivos del sistema en chip (SoC).

Se presenta, de forma condensada, el recorrido completo que sigue la herramienta Synopsys *Fusion Compiler* para transformar el netlist lógico de la ALU en un diseño físicamente colocado y libre de violaciones geométricas. El flujo se divide en nueve pasos consecutivos que abarcan desde la carga de configuración hasta la generación del *checkpoint* que servirá de punto de partida para la síntesis de reloj.

Puede apreciarse en el Cuadro 10 y la Figura 47.

Cuadro 10: Desglose de la etapa `compile_placement`.

#	<b>Step</b>	<b>Utilidad</b>	<b>Comandos / procs clave</b>
0	Carga configuración	Inicializa librerías, reglas de tecnología, variables globales y directorios de salida.	<code>source config_herramienta.tcl, config_tecnologia.tcl, config_design.tcl, set_host_options, set_app_options</code>
1	Mapping lógico inicial	Re-mapeo del netlist a celdas estándar; aún sin colocación física.	<code>compile_fusion -to initial_map</code>
2	Optimización lógica	Simplificaciones, retiming ligero y duplicado de lógica con retardo estimado.	<code>compile_fusion logic_opto, set_qor_strategy -metric timing</code>
3	Colocación inicial	Ubicación gruesa de celdas, cálculo de densidad y congestión preliminar.	<code>compile_fusion initial_place</code>
4	DRC inicial	Legalización y chequeos geométricos (ancho, espaciado, solapes).	<code>compile_fusion initial_drc, flow_check_design -step initial_drc</code>
5	Optimización inicial	Inserción de búfers y reubicación de celdas críticas para mejorar WNS/TNS.	<code>compile_fusion initial_opto</code>
6	Colocación final	Segundo pase de <i>placer</i> con modelo de retardo, minimiza congestión residual.	<code>compile_fusion final_place</code>
7	Opto final	Cierre definitivo de <i>timing</i> con parasitarios refinados y re-legalización.	<code>compile_fusion final_opto, set_ideal_network [get_clocks *]</code>
8	Reportes y guardado	Generación de métricas (área, <i>timing</i> , potencia, congestión) y checkpoint final.	<code>report_qor -summary, write_qor_data, save_block -as compile_placement_final</code>

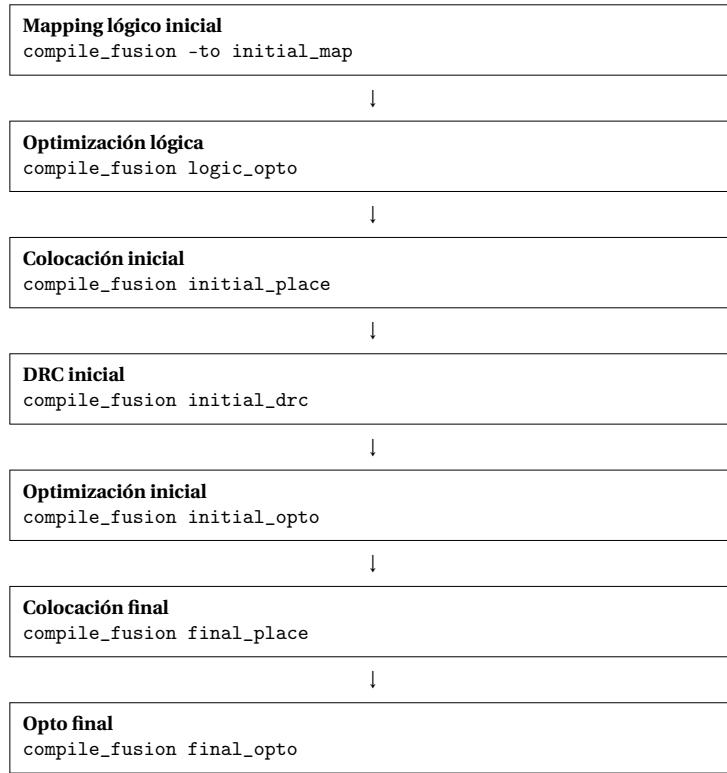


Figura 47: Steps de la fase *compile\_placement* en el flujo de *Fusion Compiler*.

## Análisis QoR

### 1. Temporización

Durante la fase de colocación la herramienta necesita estimar retardos para guiar la optimización, pero aún no dispone de la RC exacta de cada metal. Los recursos que Fusion Compiler tiene para aproximar los valores de temporización se detallan en el Cuadro 11.

Cuadro 11: Modelos de temporización empleados durante *compile\_placement*.

Nivel	Estrategia	Idea clave
1. Celdas	Delay de puerta exacto	Se emplea la caracterización de la biblioteca (tablas NLD-M/CCS), por lo que el retardo interno y la capacitancia de salida de cada celda ya son definitivos.
2. Inter-conexión	Parasitics estimados ( <i>Virtual Route</i> )	Con las celdas colocadas se calcula la distancia Manhattan entre pines y se genera un enrutamiento global virtual. A cada tramo se le asignan resistencias y capacitancias promedio (valores del archivo de tecnología), obteniendo un modelo RC lo bastante preciso para guiar la optimización temprana.
3. Reloj	Red ideal ( $skew = 0$ )	Los nets de reloj se marcan con <i>set_ideal_network</i> ; durante <i>placement</i> su retardo se fija en cero. Así se evita sobre-optimizar antes de CTS, fase en la que se construirá el árbol real y se añadirá el <i>skew</i> verdadero.

Por ende, el retardo de interconexión se approxima mediante una “virtual route” dependiente de la distancia y las reglas tecnológicas, mientras que los retardos de puerta son definitivos y la red de reloj se considera ideal. Esto permite cerrar temporización de forma preliminar y guiar la colocación sin necesidad de haber enrutado el circuito. Para matizar estos números, es conveniente comentar que hay sólo un escenario activo a la hora de calcular la temporización: el escenario de *hold* se ha desactivado, solo se evalúa el corner *setup\_ss0p72v125c*.

Resulta conveniente re-activar el escenario de *hold* en etapas siguientes para evitar sorpresas.

También cabe comentar que al revisar el .log de la etapa, durante el *placement* de alto *fan-out* apareció la advertencia “*Circuit has no valid timing endpoints*”, bajo el código “PLACE-015”. Después se actualizó el gráfico de tiempos y el QoR final refleja endpoints válidos, pero si vuelve a aparecer podría indicar que alguna SDC no se cargó antes de la fase actual.

Cuadro 12: Resumen de métricas de temporización al finalizar `compile_placement`.

Métrica (func::setup_ss0p72v125c)	Escenario activo	
	Diseño global	
WNS	0 ns (sin violaciones <i>setup</i> )	0 ns
TNS	0 ns	0 ns
Hold	0 ns (sin violaciones)	0 ns

El Cuadro 12 muestra que este punto el diseño cumple las restricciones de tiempo: no hay trayectorias con slack negativo ni en *setup* ni en *hold*.

El reporte no muestra el valor de WNS porque, al no haber slack negativo, la herramienta lo deja en blanco (“–”) y sólo imprime TNS = 0 ns.

En cuanto a las transiciones (*slew*), el límite activo es de 0.30ns. La herramienta no detecta violadores.

Para el chequeo de carga, la comprobación de capacitancia está deshabilitada y se detectan 0 violaciones de *fan-out*.

En el aspecto de *clock-gating*, hay 0 ICG insertados, los 36 FF permanecen sin gating porque su señal *enable* es constante 1. Con la advertencia CGT-3014 se ve que el flujo “*physically-aware clock gating*” quedó desactivado.

Debido a la falta de caracterizaciones a nivel de librería y parámetros que linken registros single-bit a multi-bit, las 32 celdas secuenciales no podrán convertirse a registros multi-bit en esta corrida, dejando el ratio de banking en 0. Las instancias MBFF afectadas se mantienen como registros discretos, desaprovechando ahorro de área y potencia.

## 2. Área, utilización y congestión

Para evaluar la huella física que deja la etapa `compile_placement` se analiza, por un lado, la distribución por **tipo de celda** y, por otro, la participación de cada **grupo de threshold voltage** (VT). El área total de *core* es  $A_{core} = 2258.9 \mu\text{m}^2$ .

Cuadro 13: Breakdown por categoría. Área en  $\mu\text{m}^2$

Categoría	Instancias	Área [ $\mu\text{m}^2$ ]	% celdas	% core
Lógica funcional	240	198.2	46.2	8.8
Buffers & Inversores	111	34.0	7.9	1.5
Flip-flops / registros	36	24.2	5.6	1.1
Spare / ECO	14	13.6	3.2	0.6
Filler / decap / tie	164	123.6	28.8	5.5
<b>Total celdas colocadas</b>	<b>565</b>	<b>428.9</b>	<b>100</b>	<b>19.0</b>
<b>Whitespace</b>	–	1 830.0	–	81.0

### (a) Desglose por tipo de celda

- La optimización física añade un 116 % de sobre-área con respecto al netlist lógico (de 198 a  $429 \mu\text{m}^2$ ), esencialmente por buffers, celdas *filler* y repuestos ECO.
- Con sólo un 19 % de ocupación, se preserva el 81 % de *whitespace*, lo que garantiza ruteo holgado y espacio para CTS/ECOs.

Cuadro 14: Distribución por VT. Área en  $\mu\text{m}^2$

Grupo VT	Instancias	Área [ $\mu\text{m}^2$ ]	% celdas	% core
RVT ( <i>Regular</i> )	217	268.3	60.8	11.9
HVT ( <i>High</i> )	89	102.7	24.9	4.6
SLVT ( <i>Super-Low</i> )	52	57.9	14.3	2.6
<b>Total</b>	<b>358</b>	<b>428.9</b>	<b>100</b>	<b>19.0</b>

(b) Desglose por grupo de *threshold voltage*

- La mayoría de la lógica usa RVT, equilibrando rendimiento y consumo.
- HVT se reserva para *filler/tie* y buffers ECO, reduciendo fuga.
- Un 14 % de SLVT acelera rutas críticas sin penalizar potencia global.

Estos datos proceden de los informes `report_utilization`, `report_area` y `report_threshold_voltage_groups`, confirmando que el diseño queda físicamente holgado y energéticamente balanceado tras `compile_placement`.

En la Figura 48 se puede observar una vista del bloque entero, a través de la GUI de la herramienta, luego de la etapa de colocación. Los puertos de entrada tanto como de salida se encuentran sobre los bordes, en blanco. Las celdas combinacionales, secuenciales y solo físicas se encuentran en lilac. Las celdas tap-filters pueden verse en cyan.

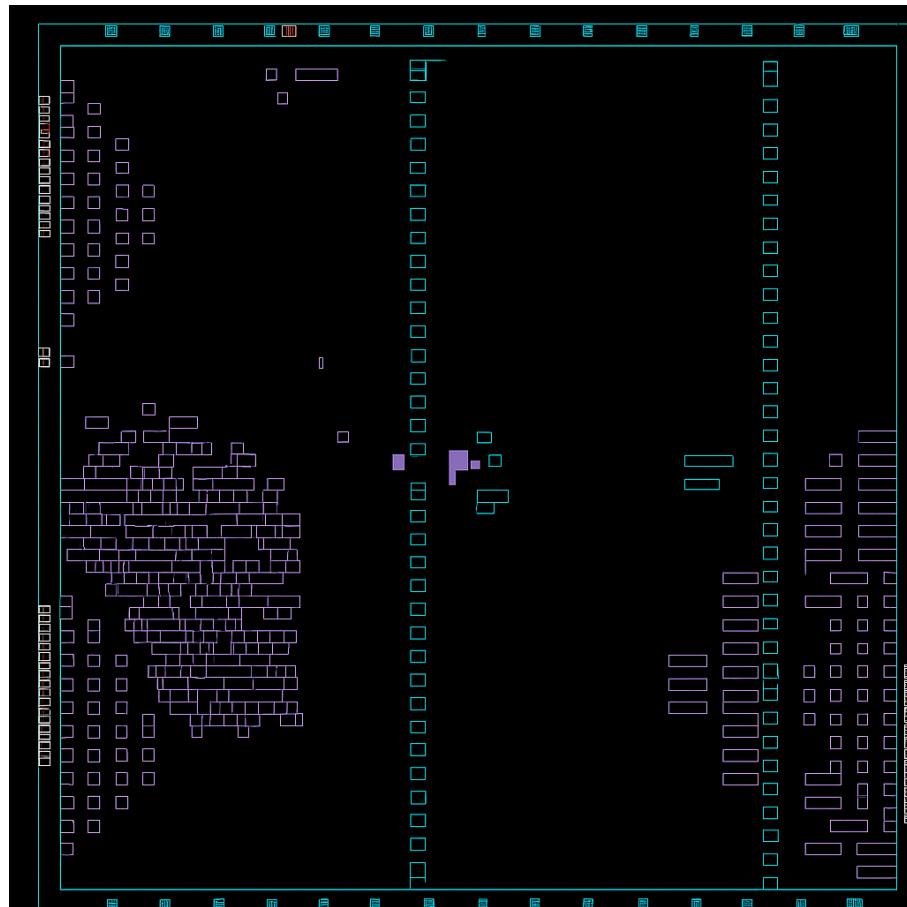


Figura 48: Vista de la base de datos post-colocación de celdas

<sup>1</sup> El número de instancias de VT sólo contempla celdas funcionales; *filler* y *tie* no llevan atributo de VT declarado.

Cuadro 15: Resumen de congestión tras `compile_placement`

Capa	Overflow total	Máx. por g-cell	% Overflow	Comentario
M4	3 tracks	1	0.058 %	Pico puntual, rápido de aliviar
M5	2 tracks	3	0.015 %	Local, aislado
M6	8 tracks	1	0.145 %	Mayor carga vertical (power stripes)
Resto	0	0	0 %	Sin congestión

El *overflow* global asciende a 14 tracks (0,109 %), con una ocupación promedio de 3,4 % en el eje vertical y 3,2 % en el horizontal; los picos de utilización (81–100 % vertical y 93 % horizontal) se concentran en apenas el 0,3 % de las *g-cells* y se deben a cruces de *buses* y *power stripes*, no a regiones extensas. La capacidad media disponible es de 9–10 pistas/*g-cell* en M2–M3 y de 5 pistas/*g-cell* en M4–M8, más que suficiente para disipar los 4 200–4 300  $\mu\text{m}$  de *wire-length* estimado. En conjunto, el mapa confirma que la red es plenamente ruteable y libre de *hot-spots*; el pequeño overflow de M6 (0,145 %) proviene de las *power stripes* y se subsana en el ruteo detallado mediante ajustes de ancho y vías, mientras que los picos de 90–100 % aparecen en muy pocas celdas y no representan un cuello de botella global.

### 3.5. Síntesis de árbol de reloj

Tras completar la etapa de colocación, en donde se fijaron las posiciones definitivas de las celdas, la Síntesis de Árbol de Reloj (CTS) construye la red física que distribuye la señal de reloj a todos los flip-flops con latencia y *skew* controlados; para ello inserta buffers, inversores y lógica de *clock-gating*, además de aplicar reglas de metal ancho (NDR) que reducen ruido y caída de IR.

El resultado es un diseño con un árbol de reloj balanceado que sirve de punto de partida al enrutador global, cerrando así la brecha entre colocación y ruteo definitivo.

En el Cuadro 16 y la Figura 16, se detalla el flujo de la etapa, con sus *scripts*, comandos nativos de la herramienta y procedimientos clave.

Cuadro 16: Pasos principales de la etapa *Clock Tree Synthesis* (CTS)

# Step	Utilidad	Comandos / procs clave
0 Carga configuración	Inicializa librerías y reglas de tecnología; abre la BD de <i>placement</i> .	<code>source config_herramienta.tcl,</code> <code>config_tecnologia.tcl,</code> <code>config_design.tcl</code>
1 Copia bloque <i>placement</i>	Parte del checkpoint legalizado.	<code>copy_block compile_placement_final</code>
2 Inserta celdas de interfaz	Define clocks, latencias y <i>uncertainty</i> .	<code>source clock_interfaces.tcl</code>
3 Aplica reglas NDR	Metal ancho y mayor separación para la red de reloj.	<code>source cts_ndr.tcl</code>
4 Construye y enruta árbol	Inserta buffers/inverters y enruta bajo NDR.	<code>source clock_opt_cts.tcl;</code> <code>clock_opt -from build_clock_tree</code>
5 Optimización post-CTS	Rebalancea <i>skew</i> y latencia, legaliza.	<code>source clock_opt_opto.tcl;</code> <code>clock_opt -from final_opto</code>
6 Reportes y check-point final	Documenta <i>skew/timing</i> y guarda la db.	<code>report_clock_tree -summary;</code> <code>report_timing;</code> <code>write_db cts_final;</code> <code>save_block -as cts_final</code>

Una vez completado el *placement*, la síntesis del árbol de reloj inicia con la fase de preparación. Se cargan los archivos de configuración y la base de datos legalizada para alinear librerías, reglas de metal y variables de entorno. A continuación se crea una copia del bloque de *compile\_placement*, de modo que cualquier modificación realizada por CTS sea reversible. Sobre este bloque se insertan las **celdas de interfaz de reloj**: se declaran los dominios, se fijan latencias y márgenes de incertidumbre, y se marcan las celdas con propósito *cts* para que el motor las use sin restricciones. Finalmente se aplican las *Non-Default Rules* (NDR) específicas del reloj, asignando metales más anchos y mayor separación para minimizar acoplamiento, *IR-drop* y electromigración durante el ruteo.

Con el entorno listo, la fase ejecutiva sintetiza y depura el árbol. El script `clock_opt_cts.tcl` invoca `clock_opt` para insertar buffers e inversores según la topología H-tree indicada y rutear cada segmento bajo las NDR definidas. Terminado este paso, una pasada de optimización posterior (`clock_opt_opto.tcl`) rebalancea el *fan-out*, ajusta longitudes y recoloca celdas para cerrar *skew* y latencia sin violar transición ni capacitancia.

Por último, se generan los reportes de *clock-tree* y *timing*, se escribe la base de datos (`write_db`) y se guarda un *checkpoint* etiquetado `cts_final`, que servirá de punto de partida para la etapa de ruteo global.

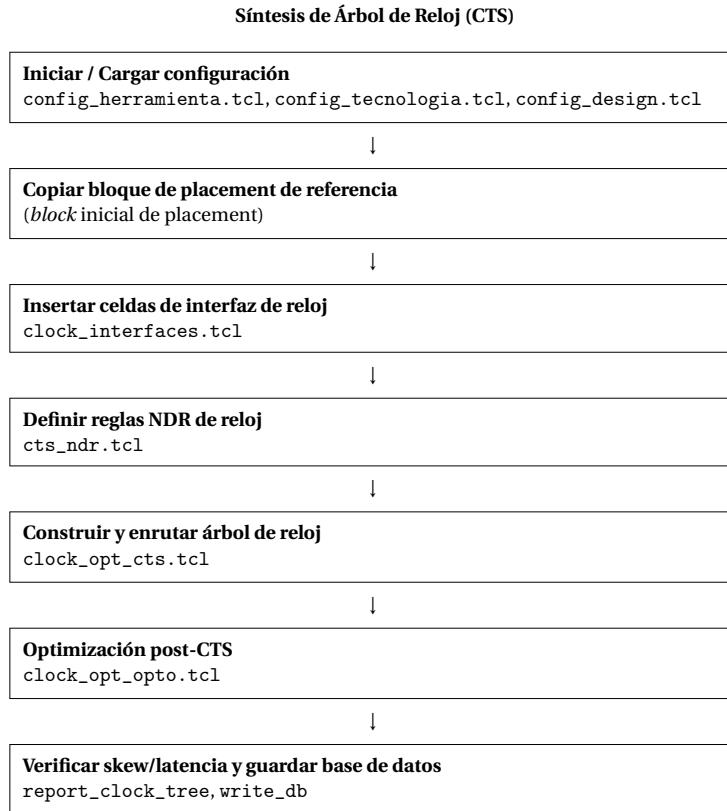


Figura 49: Steps de la fase *Síntesis de árbol de reloj (CTS)* en el flujo de diseño físico.

La etapa de síntesis de arbol de reloj se divide principalmente en los pasos `clock_opt_cts` y `clock_opt_opto`. El flujo las lanza en dos invocaciones distintas y cada una escribe sus propios logs y reportes en un subdirectorio diferente. Esto se hace, en parte, para poseer check-points escalonado y una depuración más sencillas, que en diseños grandes (mas de 1 millón de celdas) puede resultar muy útil.

Los pasos se mantienen separados para tener una trazabilidad fina, poder comparar cómo mejoran el skew, la latencia y el slack entre la construcción inicial y la optimización.

## Análisis QoR *clock\_opt\_cts*

### 1. Reloj propagados

Cuadro 17: Reloj propagados tras *clock\_opt\_cts*

Tipo	Nombre	Observaciones	# Sinks	Buffers / Inv.	WireLength (μm)	Área (μm <sup>2</sup> )
Maestro	CLOCK_MAIN	Parte del pin i_clock	33	7 buff. + 12 inv.	206,65	7,24
Generado	gen_clk_div2	/2 del maestro (q_div_2_reg)	–	hereda red	–	–
Generado	gen_clk_div4	/4 del maestro (q_div_4_reg)	–	hereda red	–	–
Generado	gen_clk_div8	/8 del maestro (q_div_8_reg)	–	hereda red	–	–

En la tabla 17 se muestran los 4 relojes propagados, de los cuales uno es el maestro y los otros tres son generados de tal.

### 2. Valores de latencia

La **Latencia del reloj** (*insertion delay*) es el tiempo que transcurre desde el pin raíz del reloj hasta que el flanco llega a cada *sink*. Los reportes de CTS entregan dos límites para cada nodo:

- **Latencia temprana** (*early*): calculada bajo condiciones rápidas (proceso-rápido,  $V_{DD}$  alto,  $T$  baja); se usa en el análisis de *hold*.
  - **Latencia tardía** (*late*): calculada bajo condiciones lentas (proceso-lento,  $V_{DD}$  bajo,  $T$  alta); se usa en el análisis de *setup*.
- Un árbol de reloj bien balanceado mantiene ambas latencias acotadas y con poca dispersión entre *sinks*, asegurando márgenes de *timing* adecuados en todos los escenarios.

Cuadro 18: Latencias temprana/tardía del árbol de reloj tras *clock\_opt\_opto*

Clock	Corner	Lat. rise (ns)	Lat. fall (ns)
CLOCK_MAIN	hold_ff0p88v125c	0,0374 / 0,0390	0,0374 / 0,0390
	setup_ss0p72v125c	0,0655 / 0,0686	0,0655 / 0,0686
gen_clk_div2	hold	0,0373 / 0,0391	0,0373 / 0,0391
	setup	0,0681 / 0,0718	0,0681 / 0,0718
gen_clk_div4	hold	0,0427 / 0,0459	0,0427 / 0,0459
	setup	0,0766 / 0,0826	0,0766 / 0,0826
gen_clk_div8	hold	0,0444 / 0,0486	0,0444 / 0,0486
	setup	0,0805 / 0,0881	0,0805 / 0,0881

**Lat. rise / Lat. fall.** Son las latencias de inserción del reloj separadas por flanco:

- **Lat. rise** – tiempo desde el pin raíz del reloj hasta la llegada del **flanco ascendente** (baja → alta) a cada *sink*.
- **Lat. fall** – tiempo desde el pin raíz hasta la llegada del **flanco descendente** (alta → baja).

La herramienta los reporta por separado porque los buffers/inversores del árbol pueden presentar retardos distintos para subida y bajada; además, conocer ambos permite detectar distorsiones de ciclo de trabajo y verificar correctamente los análisis de *setup* y *hold*.

En los reportes de CTS, cada latencia se entrega como un *pair early / late*:

- El **primer número** (*early*) es la latencia mínima, calculada en el caso “rápido” (proceso–fast,  $V_{DD}$  alto, temperatura baja). Se usa para comprobar márgenes de *hold*.
- El **segundo número** (*late*) es la latencia máxima, calculada en el caso “lento” (proceso–slow,  $V_{DD}$  bajo, temperatura alta). Se usa para comprobar márgenes de *setup*.

De este modo, en la columna **Lat. rise** se muestra *early-rise / late-rise*, y en **Lat. fall** *early-fall / late-fall*. El par encierra la ventana de variación en la que realmente puede llegar el flanco del reloj bajo todas las combinaciones de proceso, voltaje y temperatura.

### 3. Calidad de reloj

Cuadro 19: Inserción y *skew* global del reloj (después de `clock_opt_cts`)

Corner	Insertion Delay (ns)	Skew global (ns)
hold_ff0p88v125c	<b>0,108 – 0,111</b>	$\approx 0,068$
setup_ss0p72v125c	<b>0,175 – 0,179</b>	$\approx 0,106$

El *insertion delay* es la diferencia entre el reloj en el pin raíz y el peor sink; los valores se refieren a la última corrida de optimización de área.

El *skew* global es la dispersión máxima entre sinks después de balancear el árbol.

### 4. Asignación de Metales

La sección `Clock-nets statistics` del informe CTS muestra `min layer = M5` y `max layer = M8`; por tanto, ningún segmento del reloj se enruta por debajo de **M5** ni por encima de **M8**.

El script `cts_ndr.tcl` aplica la regla `CLK_NDR` con:

```
set_clock_routing_rules -layers {M8 M7 M6 M5}
```

donde se fija que:

- los troncales (*root/trunk*) se enrutan en **M8** (horizontal) y **M7** (vertical);
- las ramas intermedias (*branches*) se permiten en **M6**;
- los *leaves/taps* que llegan a cada flop se limitan a **M5**.

En consecuencia, toda la red de reloj se mantiene en las capas superiores **M5–M8**, reservando las inferiores (**M1–M4**) para señales de datos y reduciendo resistencia, acoplamiento y *IR-drop*.

En la figura 50 se observa el ruteo de la señal de *clock* por tales metales.

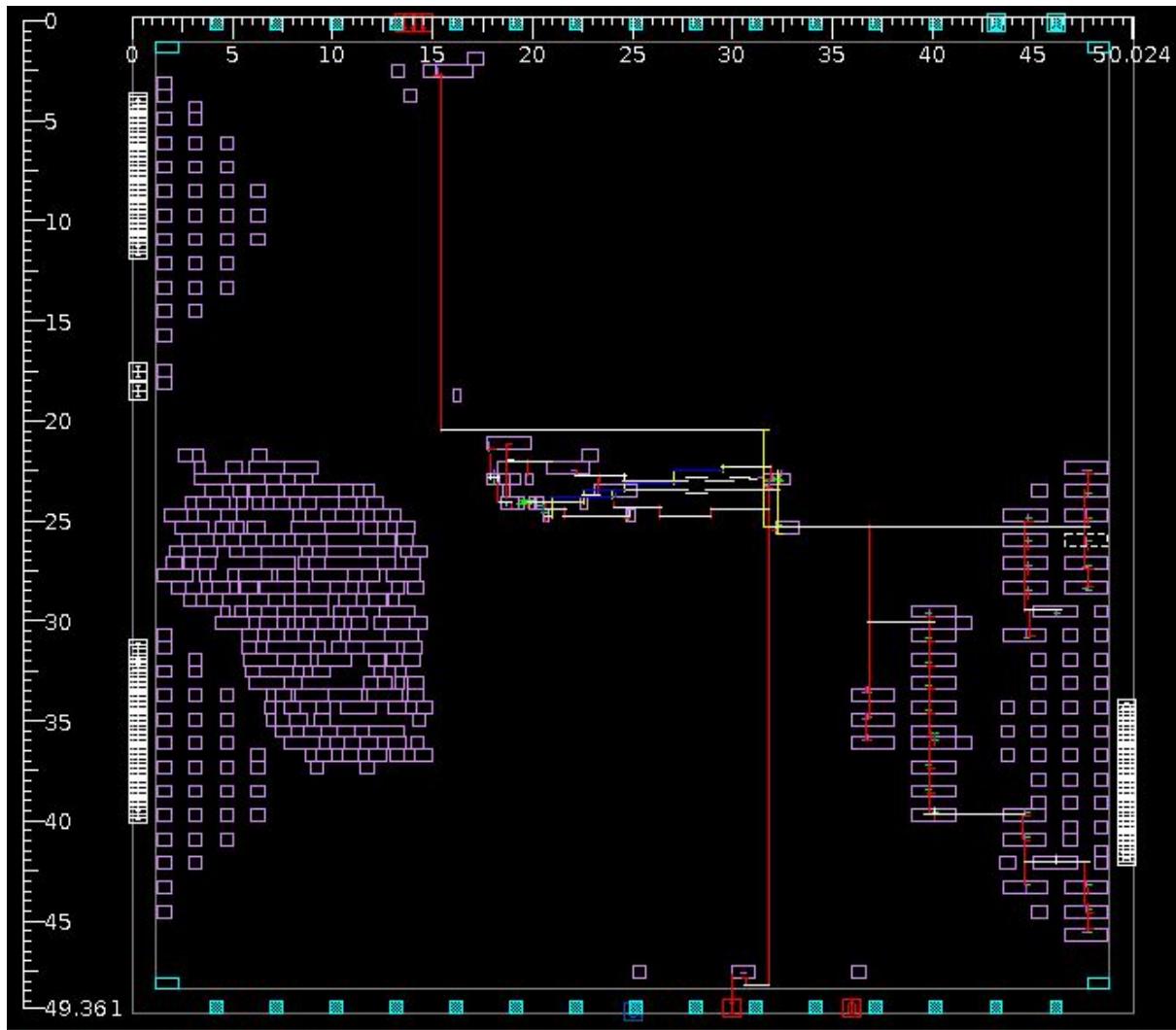


Figura 50: Ruteo de la señal de *clock*

## 5. Conclusión de CTS QoR

El reloj principal (CLOCK\_MAIN) presenta un skew inferior a 70 ps y una latencia de inserción <0.18 ns en los dos corners críticos, con tres relojes derivados perfectamente balanceados y sin violaciones de timing. El árbol resultante es compacto ( $\approx 207 \mu\text{m}$  de metal) y ligero (7 buffers más 12 inversores), cumpliendo las restricciones de transición y capacitancia indicadas en los archivos de log.

### 3.6. Enrutamiento

Una vez finalizada la etapa de Síntesis de Árbol de Reloj, donde ya se encuentran ruteadas las señales de los distintos relojes del diseño y la base de datos se encuentra lista para ser expuesta al enrutamiento.

Para lograr el enrutamiento del cableado tipo *señal*, se utilizan dos *enrutadores*:

- **Enrutador global:**

- Divide el bloque en *celdas de ruteo globales* (GRCs) y calcula la capacidad de pistas disponible en cada una.
- Asigna caminos aproximados para las nets, sin colocar aún las formas metálicas reales.
- El objetivo es identificar rutas factibles y estimar congestión, generando reportes de “overflow” (que refiere a cuando la demanda de pistas excede la capacidad).
- Se realizan fases iniciales y de re-enrutamiento para aliviar la congestión, ajustando las rutas hasta alcanzar un nivel aceptable.

- **Enrutador detallado:**

- Toma como base los caminos aproximados del *global routing* y de la asignación de tracks.

- Coloca las formas metálicas exactas en capas específicas, resolviendo todas las violaciones de reglas de diseño (DRC).
- Implementa optimización de reglas de manufactura, electromigración, conteo de vías y longitud de cables.
- Si encuentra una violación, desecha la ruta conflictiva y la vuelve a trazar hasta converger con un diseño legal, con un tope indicado de iteraciones.

Se concluye que el *global routing* es una visión “macro”, planifica caminos y detecta congestión, mientras que el *detail routing* corresponde a una ejecución “micro”, concreta las rutas físicas y asegura el cumplimiento de reglas de diseño.

A través de estas dos herramientas se construyen las conexiones físicas que interconectan todas las celdas estándar, macros y pines de entrada/salida, respetando las restricciones de temporización, integridad de señal y consumo de potencia.

Esta etapa del flujo de diseño se gestiona mediante dos *scripts* principales: `route_auto.tcl` y `route_opt.tcl`. Ambos scripts encapsulan la ejecución de comandos nativos de *Fusion Compiler* y procedimientos definidos en los archivos de utilidades del proyecto.

- `route_auto.tcl`: se encarga de ejecutar el comando nativo del mismo nombre, el cual engloba las fases de enrutamiento global, asignación de tracks y enrutamiento detallado. Para ello, invoca procedimientos definidos en `procs.tcl` que configuran el entorno, verifican dependencias de etapas previas (`,clock_opt.opto`) y gestionan los directorios de salida. El resultado de esta ejecución es un diseño completamente ruteado con todas las conexiones físicas implementadas de acuerdo con las reglas tecnológicas establecidas en los archivos de configuración.
- `route_opt.tcl`: Una vez finalizado el enrutamiento inicial, llama al comando nativo `route_opt`, encargado de realizar optimizaciones posteriores al enrutamiento detallado. Entre las tareas que se ejecutan están: la corrección incremental de violaciones de reglas de diseño (DRC), la optimización de tiempos de propagación, y la reparación de problemas de integridad de señal o electromigración. También se invocan procedimientos auxiliares para manejar *ECO routing* (que modifica el enrutamiento existente solo en las zonas afectadas por cambios lógicos, temporales o físicos sin volver a enrutar todo el chip), ajustar longitudes críticas de interconexiones, y mejorar la distribución de vías.

En el Cuadro 51 se muestra el flujo de diseño de la etapa de ruteo, con sus comandos y procedimientos más relevantes.

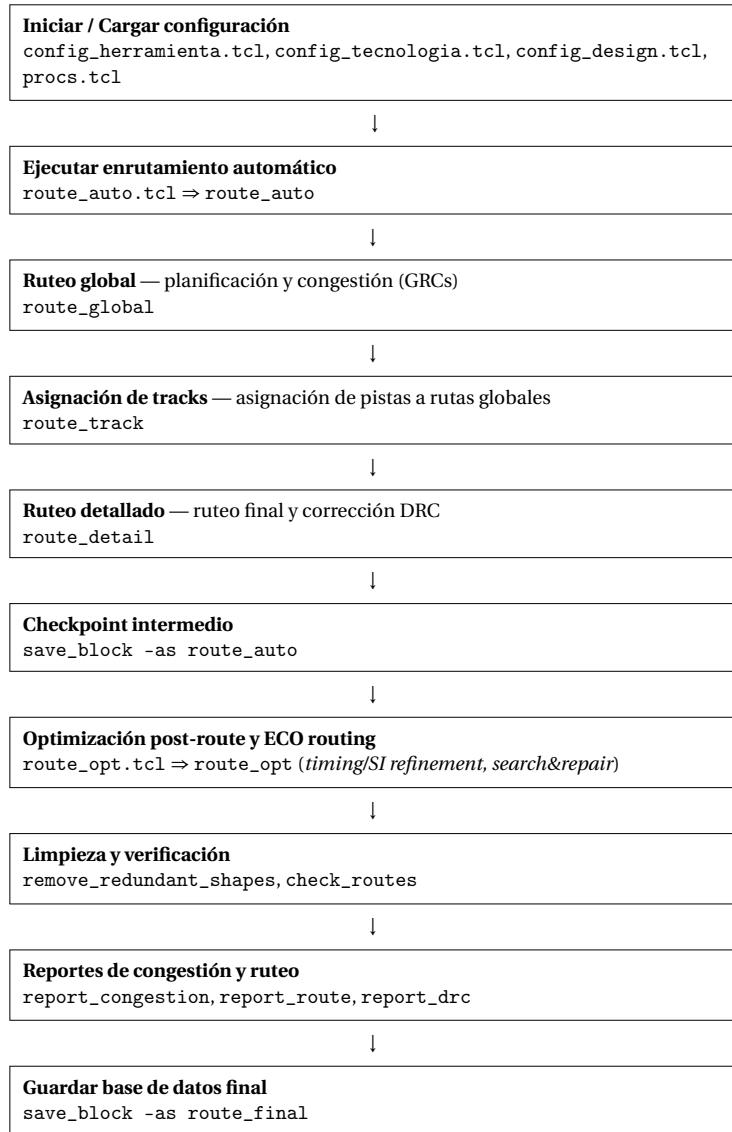


Figura 51: Flujo de *Routing* con comandos nativos y scripts principales

Al concluir la etapa de *Routing*, el diseño del ALU de 32 bits presenta todas las interconexiones físicas completadas y optimizadas. En esta fase, se procura garantizar que no existan redes abiertas ni cortocircuitos, que las reglas de diseño físicas se cumplan, y que las rutas críticas satisfagan los requisitos de temporización establecidos mediante el cierre preliminar de *timing* y de integridad de señal. En caso de finalizar esta etapa con cortocircuitos o red abiertas, o cualquier tipo de violación física, deberá resolverse en una etapa ECO posterior.

El resultado de esta etapa es un diseño completamente ruteado y verificado internamente, almacenado en una base de datos de estado final mediante:

```
save_block -as route_final
write_def route_final.def
write_verilog route_final.v
```

Estos archivos constituyen la referencia física y lógica que se utilizará en las etapas posteriores de verificación y *sign-off*.

El diseño aún no está listo para ser enviado a fabricación. Antes del *tape-out*, debe pasar por las verificaciones de *sign-off* que aseguran su manufacturabilidad, equivalencia funcional y confiabilidad eléctrica. Entre ellas se incluyen:

- **DRC/LVS:** comprobación de reglas de diseño y correspondencia entre el layout y el netlist lógico.
- **Extracción parasítica:** obtención de resistencias y capacitancias reales de interconexiones.

- **Análisis estático de temporización (STA):** verificación final del cumplimiento de tiempos con parasitismos extraídos.
- **Análisis de IR drop y electromigración (EM):** evaluación de estabilidad de la red de alimentación y confiabilidad a largo plazo.
- **Verificación formal (FEC):** comprobación de equivalencia lógica entre el netlist funcional y el netlist físico.

Una vez superadas estas verificaciones, el diseño se considera listo para la generación de los archivos de salida (*GDSII/OASIS*) y su posterior envío a la *foundry* para fabricación. El proceso finalizado de pasar de una descripción de alto nivel de un chip (RTL) a un archivo GDSII, que constituye el plano físico final utilizado para su fabricación, constituye la principal tarea de un ingeniero de diseño físico.

### ***App-options* y modificaciones de usuario**

Durante la implementación física, las *application options* (Opciones de aplicación) controlan y guían internamente el comportamiento de la herramienta utilizada. Así, ajustan sus niveles de esfuerzo, sus objetivos de temporización o consumo.

Estas opciones guían a la herramienta para decidir qué algoritmos aplicar, cuánto tiempo invertir en la búsqueda de soluciones y qué criterios priorizar. Así, por ejemplo, podrán enfocarse en cierre de temporización o en utilizar las celdas estándar de menor consumo de potencia.

Para ilustrar lo anterior, en esta etapa de ruteo, las opciones:

```
route.detail.timing_driven y  
route.global.timing_driven
```

activan el ruteo impulsado por temporización, haciendo que el enrutador priorice las redes críticas y minimice los retardos de propagación tanto a nivel global como detallado.

La opción route.track.crosstalk\_driven habilita un modelo de ruteo sensible al acoplamiento eléctrico entre pistas, reduciendo interferencias y mejorando la integridad de señal), mientras que

```
route.detail.insert_redundant_vias_layer_order_low_to_high
```

permite insertar vías redundantes desde las capas inferiores hacia las superiores, incrementando la robustez en la grilla de alimentación del diseño y mitigando riesgos de electromigración.

Asimismo, parámetros como

```
route.common.concurrent_redundant_via_mode_reserve_space ó  
route.detail.eco_max_number_of_iterations 10
```

controlan el esfuerzo de reparación incremental en la etapa de *ECO routing*, asegurando que la herramienta disponga de espacio para optimizaciones posteriores sin violar las reglas de diseño.

## **Análisis QoR**

### **1a. Temporización: Violaciones de transición**

Cuadro 20: Verificación de violaciones de transición y capacitancia después del enrutamiento

Modo	Esquina (Corner)	Tipo de violación	Cantidad
func	hold_ff0p88v125c	<i>max_transition</i>	0
func	setup_ss0p72v125c	<i>max_transition</i>	0
func	setup_ss0p72v125c	<i>max_capacitance</i>	0
<b>Total de violaciones</b>			<b>0</b>

Como se ve en el Cuadro 20, no hay violaciones de las restricciones establecidas al inicio del proyecto, términos de transiciones y capacitancias. En caso de no haberlo especificado en los constraints, se utilizan los provistos por default en el flujo de diseño.

### **1b. Temporización: Violaciones de *setup* y *hold***

Para ambos escenarios func::setup\_ss0p72v125c y func::hold\_ff0p88v125c la herramienta reporta que el WNS (*Worst Negative Slack*, peor retardo negativo) no tiene valores negativos, el TNS (*Total Negative Slack*,

retardo negativo acumulador total) es 0 y la cantidad de caminos con violaciones también es 0, lo que concluye que el diseño no tiene violaciones de temporización.

## 2. Reporte de celdas del diseño

Cuadro 21: Resumen de información de celdas del diseño post-routing

Tipo de celda	Cantidad	% del total	Área	% del área total
<i>Standard cells</i>	377	41	212,765	66
<i>Filler cells</i>	1	0	0,400	0
<i>Corner pad cells</i>	4	0	2,664	1
<i>Otros</i>	527	57	104,429	32
<b>Celdas especiales</b>	7	1	3,419	1
<i>Tie off</i>	7	1	3,419	1
<b>Tipo de umbral (VT)</b>				
<i>HVT (High VT)</i>	81	9	60,206	18
<i>NVT (Normal VT)</i>	818	89	253,835	79
<i>Otros</i>	10	1	6,216	2
<b>Celdas fijas y móviles</b>				
<i>Celdas fijas</i>	670	73	192,030	59
<i>Celdas móviles</i>	239	26	128,227	40
<b>Clasificación funcional</b>				
<i>Combinacionales</i>	871	95	270,707	84
<i>Secuenciales</i>	38	4	49,550	15
<b>Buffers e inversores</b>				
<i>Buffers</i>	108	11	45,466	14
<i>Inversores</i>	13	1	4,928	1
<i>Buffer/Inverter</i>	121	13	50,394	15
<b>Celdas adicionales</b>				
<i>Spare cells</i>	14	1	13,631	4
<i>Flip-flops</i>	38	4	49,550	15
<i>Mux logic</i>	1	0	1,021	0

En el Cuadro 21 se puede observar el desglose de celdas por cantidad y área. El 95 % de celdas combinacionales se espera para una ALU de 32 bits compacta con la utilización resultante. El resultado de VTs esta dominado por NVT, con una fracción de HVT de aproximadamente el 9 %.

En cuanto a la temporización, se requirieron buffers e inversores en torno al 24 % del conteo total de celdas, un valor razonable para cerrar el slew, las cargas y las rutas críticas.

Como mejora, podría explorarse la posibilidad de re-dimensionar el VT selectivo de algunas celdas en redes no críticas para recortar potencia estática y lograr una ligera reducción de buffering, si los márgenes de temporización así lo permitieran.

Cuadro 22: Referencia de celdas utilizadas en el diseño

Nombre	Tipo	Cant.	Ancho	Alto	Area	SiteArea
SAEDRVT14_CAPSPACER1	end_cap	154	0,074	0,600	0,044	1
SAEDRVT14_CAPB2	end_cap	138	0,296	0,600	0,178	4
SAEDRVT14_CAPTR2	end_cap	108	0,296	0,600	0,178	4
SAEDRVT14_BUF_3	lib_cell	105	0,666	0,600	0,400	9
SAEDRVT14_TAPDS	well_tap	101	0,888	0,600	0,533	12
SAEDRVT14_TAPDN	well_tap	40	0,888	0,600	0,533	12
SAEDRVT14_TIE0_1	lib_cell	33	0,740	0,600	0,444	10
SAEDRVT14_AO22_QPS	lib_cell	32	2,146	0,600	1,288	29
SAEDRVT14_ADBF_V1_QPS	lib_cell	30	1,702	0,600	1,021	23
SAEDRVT14_AO22_1	lib_cell	28	0,888	0,600	0,533	12
SAEDRVT14_OR2_QPS	lib_cell	26	0,518	0,600	0,311	7
SAEDRVT14_AN2_QPS	lib_cell	25	0,666	0,600	0,400	8
SAEDRVT14_AN2_MM	lib_cell	18	0,518	0,600	0,311	7
SAEDRVT14_CAPTAP6	well_tap	17	0,592	0,600	0,355	8
SAEDRVT14_OR2_1	lib_cell	17	0,518	0,600	0,311	7
SAEDRVT14_TIE0_4	lib_cell	16	0,740	0,600	0,444	10
SAEDSLVT14_INV_0p5	lib_cell	14	0,296	0,600	0,170	4
SAEDSLVT14_INV_0p75	lib_cell	13	0,370	0,600	0,222	5
SAEDRVT14_NR2_QPS	lib_cell	12	0,666	0,600	0,400	8
SAEDRVT14_CAPBIN3	corner	2	1,110	0,600	0,666	15
SAEDRVT14_CAPBIN13	corner	2	1,110	0,600	0,666	15
SAEDRVT14_NR2_ECO_1	lib_cell	1	0,926	0,600	0,556	14
SAEDRVT14_ND2_ECO_1	lib_cell	1	0,926	0,600	0,556	14
SAEDRVT14_ND2_ECO_2	lib_cell	1	1,036	0,600	0,622	14
SAEDRVT14_AO22_QPS	lib_cell	1	0,814	0,600	0,488	11
SAEDRVT14_MUX2_2	lib_cell	1	1,036	0,600	0,622	14
SAEDRVT14_ADFH_QPS	lib_cell	1	1,110	0,600	0,666	15
SAEDRVT14_FDSEQ_V2LP	lib_cell	1	0,850	0,600	0,510	9
SAEDRVT14_AN2_1	lib_cell	1	0,518	0,600	0,311	7

2

Las celdas listadas en el Cuadro 22 corresponden a distintos grupos funcionales definidos por la biblioteca SAED14, definidos en el manual *SAED\_EDK14\_FINFET Educational Design Kit (EDK)*. Las celdas *end\_cap* (SAEDRVT14\_CAPSPACER1, CAPB2, CAPTR2) se utilizan para cerrar los extremos de las filas de celdas estándar, garantizar continuidad de pozos y difusiones y evitar violaciones de reglas de diseño en los bordes.

Las celdas *well\_tap* (TAPDS, TAPDN, CAPTAP6) conectan los pozos N y P a las redes de alimentación (VDD/VSS), proporcionando el sesgo adecuado del sustrato y reduciendo el riesgo de *latch-up*.

Las celdas de tipo *corner* (CAPBIN3, CAPBIN13) se colocan en esquinas de la matriz para completar la estructura física del *core* y mantener la integridad geométrica de los pozos.

Por su parte, las celdas *lib\_cell* implementan la funcionalidad lógica del diseño: los inversores (SAEDSLVT14\_INV\_0p5, INV\_0p75) generan la inversión lógica; los buffers (SAEDRVT14\_BUF\_3) proporcionan ganancia de conducción para manejar cargas mayores; las puertas combinacionales básicas incluyen AND de dos entradas (AN2\_QPS, AN2\_MM), OR (OR2\_QPS, OR2\_1), y NOR (NR2\_QPS); las puertas compuestas (AO22\_QPS, AO22\_1) realizan operaciones AND-OR según la topología descrita en la librería.

Asimismo, las celdas ECO (NR2\_ECO\_1, ND2\_ECO\_1, ND2\_ECO\_2) son variantes optimizadas para modificaciones incrementales durante *Engineering Change Orders*.

Finalmente, celdas especializadas como MUX2\_2 implementan multiplexores de dos entradas, y FDSEQ\_V2LP corresponde a un flip-flop D secuencial con capacidad de captura según el esquema especificado por la biblioteca.

<sup>2</sup> El área (*Area*) está expresada en micrómetros cuadrados ( $\mu\text{m}^2$ ) y se obtiene como *Width*  $\times$  *Height*. Por ejemplo, para la celda SAEDRVT14\_CAPSPACER1:  $0,074 \mu\text{m} \times 0,600 \mu\text{m} \approx 0,044 \mu\text{m}^2$ , valor que coincide con el reportado en la tabla. Por su parte, *SiteArea* no está en unidades físicas, sino que representa el número de sitios de colocación que ocupa la celda (es una magnitud adimensional). Un “site” corresponde a la unidad básica de fila en la tecnología SAED14, cuyo tamaño coincide con el área de la celda mínima ( $0,074 \times 0,600 \approx 0,044 \mu\text{m}^2$ ). De este modo, una celda mínima tiene *SiteArea* = 1, mientras que una celda con *SiteArea* = 4 ocupa cuatro sitios de ancho.

Cuadro 23: Distribución de celdas por grupo de *threshold voltage* (VT)

Grupo VT	Total (%)	Repeaters (%)	Comb. (%)	Registers (%)	Secuenciales (%)	Clock net (%)	Macro (%)	Physical-only (%)
HVT	81 (8,91 %)	0 (0,00 %)	45 (4,95 %)	33 (3,63 %)	0 (0,00 %)	3 (0,33 %)	0 (0,00 %)	0 (0,00 %)
RVT	818 (89,99 %)	106 (11,66 %)	163 (17,93 %)	2 (0,22 %)	0 (0,00 %)	4 (0,44 %)	0 (0,00 %)	543 (59,74 %)
SLVT	10 (1,10 %)	0 (0,00 %)	0 (0,00 %)	0 (0,00 %)	0 (0,00 %)	10 (1,10 %)	0 (0,00 %)	0 (0,00 %)
<b>Total</b>	<b>909 (100%)</b>	<b>106 (11,66 %)</b>	<b>208 (22,88 %)</b>	<b>35 (3,85 %)</b>	<b>0 (0,00 %)</b>	<b>17 (1,87%)</b>	<b>0 (0,00 %)</b>	<b>543 (59,74 %)</b>

Cuadro 24: Distribución de área por grupo de *threshold voltage* (VT)

Grupo VT	Área (%)	Repeaters (%)	Comb. (%)	Registers (%)	Secuenciales (%)	Clock net (%)	Macro (%)	Physical-only (%)
HVT	60,206 (18,80 %)	0 (0,00 %)	16,561 (5,17 %)	42,313 (13,21 %)	0 (0,00 %)	1,332 (0,42 %)	0 (0,00 %)	0 (0,00 %)
RVT	253,835 (79,26 %)	41,958 (13,10 %)	90,221 (28,17 %)	3,374 (1,05 %)	0 (0,00 %)	0,932 (0,29 %)	0 (0,00 %)	117,349 (36,64 %)
SLVT	6,216 (1,94 %)	0 (0,00 %)	0 (0,00 %)	0 (0,00 %)	0 (0,00 %)	6,216 (1,94 %)	0 (0,00 %)	0 (0,00 %)
<b>Total</b>	<b>320,257 (100%)</b>	<b>41,958 (13,10 %)</b>	<b>106,782 (33,34 %)</b>	<b>45,688 (14,27 %)</b>	<b>0 (0,00 %)</b>	<b>8,480 (2,65 %)</b>	<b>0 (0,00 %)</b>	<b>117,349 (36,64 %)</b>

En tecnologías CMOS avanzadas, las bibliotecas estándar incluyen variantes de celdas con distinta tensión umbral (*threshold voltage*, VT), lo que permite equilibrar velocidad y consumo.

Las celdas **HVT** (High-VT) poseen una tensión umbral más elevada, lo que reduce la corriente de fuga y por tanto la potencia estática, a costa de un mayor retardo. Son apropiadas en trayectos no críticos o de baja actividad.

Las celdas **RVT** (Regular-VT) representan el punto intermedio entre desempeño y fuga, y suelen dominar la implementación, como ocurre en este diseño.

Por último, las celdas **SLVT** (Super-Low-VT) presentan una tensión umbral reducida que mejora significativamente la velocidad de commutación, permitiendo cerrar las restricciones de *setup* en caminos críticos, aunque incrementan la corriente de fuga. El uso conjunto de estas variantes permite optimizar el trade-off entre retardo, consumo dinámico y fuga estática durante el flujo de síntesis y de *place & route*.

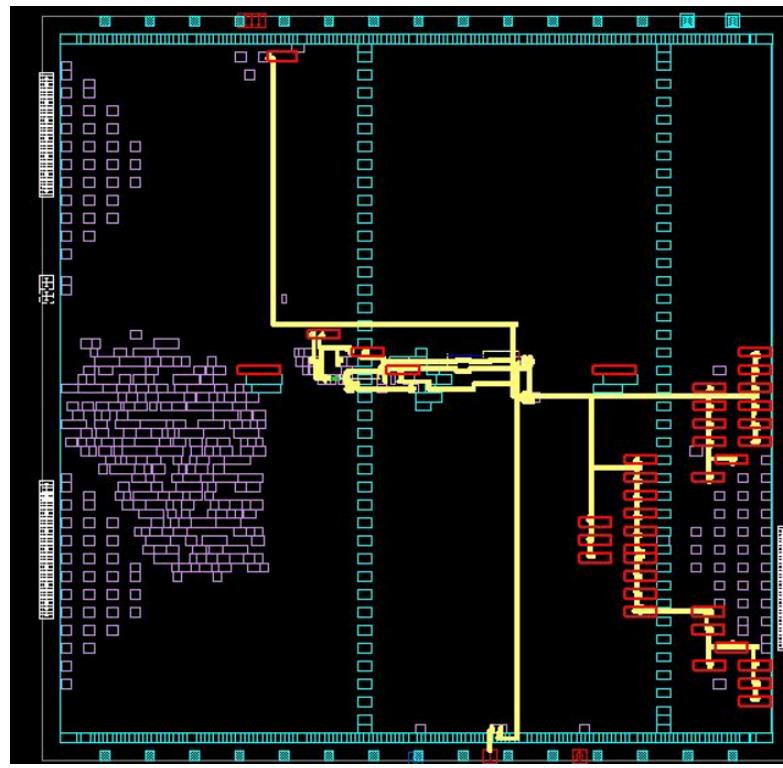


Figura 52: En rojo celdas de tipo secuencial y en amarillo net de clock que provee la temporización

En la Figura 52 se ven resaltados en rojo las celdas estándar que tienen función secuencial (i.e. registros single bit). En amarillo, se resaltan las nets de reloj que comuta tales celdas. Las celdas de color violeta corresponden a celdas con función combinacional (i.e. buffers, MUXES, inversores, entre otras)

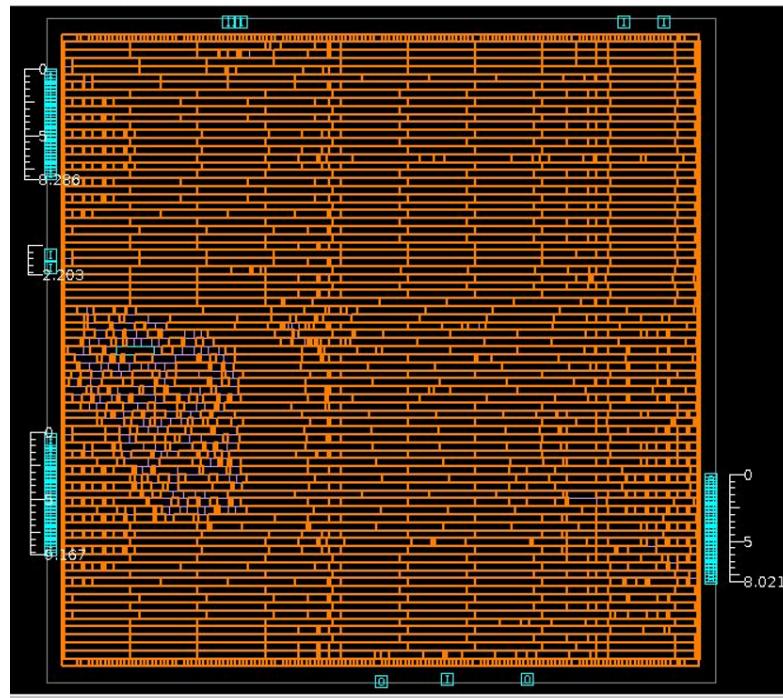


Figura 53: En naranja las celdas "*physical only*"

En la Figura 53 se pueden apreciar la cantidad de celdas con funcionalidad exclusivamente "física", que son agregadas en el flujo de manera automática. Su objetivo es cumplir requisitos físicos de la tecnología y de manufactura. Ejemplos de estas son:

- *filler cells*: rellenan huecos entre celdas estándar
- *end-cap cells*: cierre de filas y bordes del *core*
- *well-taps*: anclaje de pozos a VDD/VSS para evitar *latch-up*
- *decap cells*: desacoplo de alimentación
- diodos: para evitar DRCs de antenna

En el caso de este diseño se ve una mayoría de celdas físicas dado el bajo porcentaje de utilización para el floorplan utilizado. Lejos de significar una mala implementación, propone un cumplimiento asegurado de las reglas físicas.

### 3. Potencia

En el reporte de potencia generado tras la etapa de *routing*, se observa que el escenario `func::hold_ff0p88v125c` sólo informa potencia de fuga, mientras que los términos asociados a *switching power* aparecen como N/A. Esto se debe a que dicho corner está definido en el entorno de análisis exclusivamente como un escenario de verificación de *hold* y de *leakage*, por lo que el motor de potencia no activa los cálculos de potencia dinámica (interna o de conmutación). En contraste, el escenario `func::setup_ss0p72v125c` sí está marcado como escenario dinámico, por lo que Fusion Compiler evalúa en él las componentes de potencia interna, potencia de conmutación y potencia dinámica total. Esta separación entre corners se fundamenta en la variación de parámetros eléctricos del proceso: los corners “fast” tienden a presentar mayores corrientes de fuga y se utilizan para analizar la potencia estática y los requisitos de *hold*, mientras que los corners “slow” se emplean para capturar el peor caso de retardo y de actividad dinámica, ya que la energía por transición depende tanto de la capacitancia cargada como del tiempo de conmutación.

Por ello, la estimación completa de potencia dinámica (*switching + internal*) se asocia únicamente al corner de *setup*, mientras que el análisis de fuga se reporta en ambos escenarios, que es la potencia que se reporta como consumo dado su pesimismo.

Es importante no mezclar *leakage* y *dynamic* del mismo corner, dado que cada tipo de potencia se evalúa en el escenario que mejor representa su peor caso. El Cuadro 25 resume los valores obtenidos en cada corner.

Cuadro 25: Resumen de potencia por escenario de análisis post-route

Escenario	Leakage Power (mW)	Internal Power (mW)	Switching Power (mW)	Total Dynamic (mW)
<code>func::hold_ff0p88v125c</code>	$2,254 \times 10^{-2}$	N/A	N/A	N/A
<code>func::setup_ss0p72v125c</code>	$1,978 \times 10^{-2}$	$3,141 \times 10^{-3}$	$3,500 \times 10^{-3}$	$6,641 \times 10^{-3}$

### 4. Cableado

La implementación fue terminada 454 nets de señal y con 13 nets de reloj, de las 12 cumplen con reglas NDR de doble espaciado y doble ancho.

El *wirelength* total del bloque reportado por la herramienta es de aproximadamente  $4285 \mu\text{m}$ . Por su parte, la longitud total del árbol de reloj asciende a  $217,884 \mu\text{m}$ . A partir de estos valores, la longitud asociada exclusivamente a las señales de datos puede estimarse como:

$$WL_{\text{signal}} \approx 4285 - 218 \approx 4067 \mu\text{m}.$$

Cuadro 26: Resumen de métricas globales de ruteo

Métrica	Valor
Total Wire Length	$4282 \mu\text{m}$
Total Number of Contacts	2710
Total Number of Wires	2902
Total Number of PtConns	121
Total Number of Routed Wires	2902
Total Routed Wire Length	$4270 \mu\text{m}$
Total Number of Routed Contacts	2710

Cuadro 27: Uso de capas metálicas (*Layer Usage*) después del enrutamiento

Capa	Longitud ruteada [ $\mu\text{m}$ ]
M1	3
M2	852
M3	1187
M4	985
M5	686
M6	507
M7	54
M8	10
M9	0
MRDL	0

En la Figura 54 se observa el ruteo de señal además de reloj en los cables sobre los distintos metales M1-M8. Cada color en los cables representa un Metal distinto.

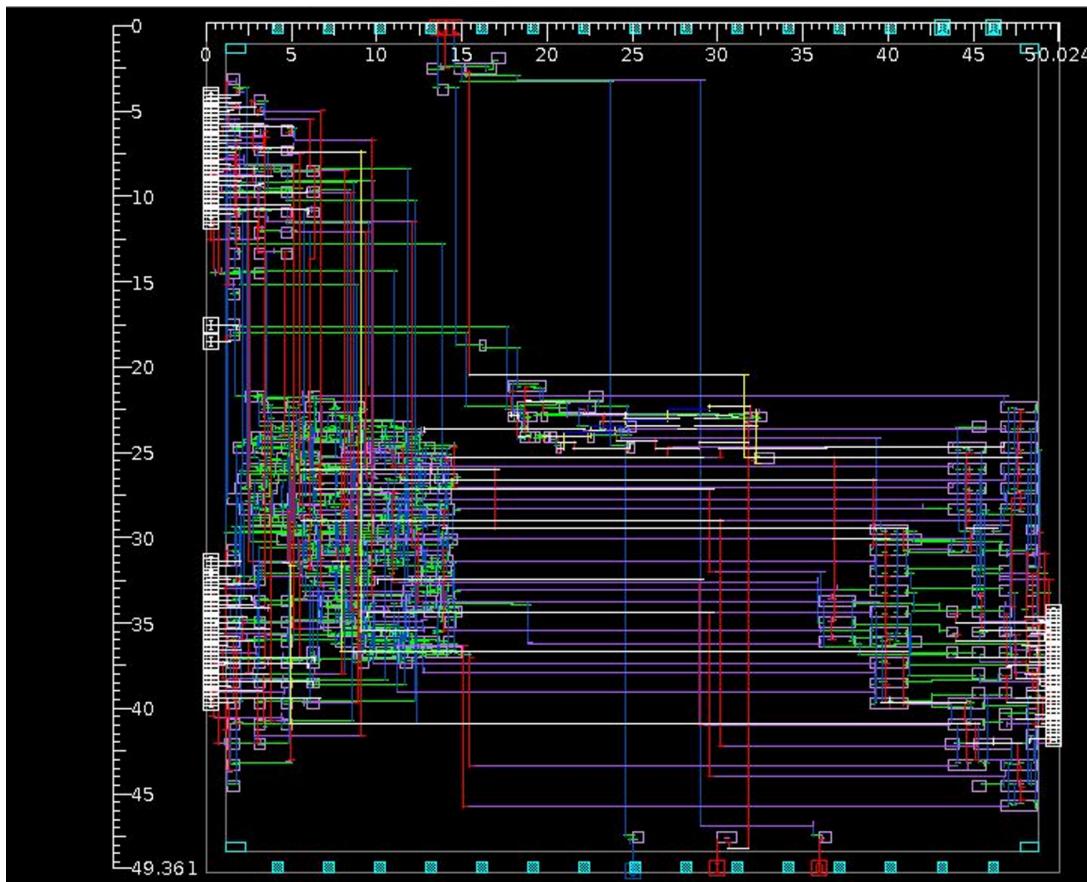


Figura 54: Ruteo de la señal de señal y reloj

<sup>3</sup>El *Total Number of PtConns* corresponde al número total de puntos en los cuales el ruteo debe garantizar continuidad eléctrica, tales como uniones entre segmentos metálicos, bifurcaciones de una red o conexiones directas a los pines de las celdas.

Cuadro 28: Conteo total de vías utilizadas en el diseño

Capa de Vía	Número total de vías
VIA1	842
VIA2	1149
VIA3	284
VIA4	240
VIA5	173
VIA6	14
VIA7	8

La distribución del *wirelength* por capa metálica muestra que el ruteo del ALU se concentra principalmente en las capas intermedias M2–M6, mientras que M1 y las capas superiores presentan un uso marginal.

El reducido uso de M1 (sólo 3  $\mu\text{m}$ ) es coherente con un diseño estándar basado en celdas de librería, donde la mayor parte del ruteo interno de celdas se resuelve dentro de la propia celda y el ruteo entre instancias se desplaza a capas superiores. En la interfaz gráfica de la herramienta se puede ver, como un elemento geométrico, el bloqueo de Metal 0 y Metal 1 correspondientes a ruteos internos de la celda.

Las capas M2, M3 y M4 concentran la mayor longitud (852, 1187 y 985  $\mu\text{m}$ , respectivamente), lo que indica que la interconexión local del bloque se implementa preferentemente en estas capas, mientras que M5 y M6 se utilizan como recursos adicionales para aliviar la congestión y cerrar la temporización.

Por otro lado, el hecho de que M9 y MRDL presenten longitud ruteada nula es consistente con el tamaño reducido del diseño y su naturaleza académica: en esta configuración tecnológica, las capas metálicas más altas suelen reservarse para ruteo global de muy larga distancia, distribución de reloj a nivel de chip o mallas de alimentación y *redistribution layers* (MRDL) en contextos de integración a nivel de *package*. Es decir, dado que el bloque podría ser integrado en un chip mayor junto con otros bloques y un *nivel superior*, estas capas superiores se utilizan para la conexión de los llamados *bumps*, que se utilizarán posteriormente para la conexión dentro del empaquetado del chip.

Dado que el ALU se implementa como un bloque pequeño, sin red de potencia compleja ni interconexiones globales extensas, no es necesario emplear M9 ni MRDL, y el ruteo se puede completar de forma eficiente utilizando únicamente las capas inferiores e intermedias, por lo que se concluye que el interconexionado y el uso de las capas de metales es el adecuado para la integración del diseño propuesto.

Por último, el Cuadro 28 muestra la cantidad de vías utilizadas en el diseño. Se observa que no hay vías mayores a VIA7, que verifica el hecho de que no hay metales más arriba que Metal 8.

### 3.7. Trabajo Práctico de laboratorio: Cuestionario

## 4. Cuestionario de integración final

### 4.1. Enunciado del cuestionario

#### 1. Flujo RTL → GDSII

- 1.1. Explicar brevemente qué se entiende por flujo *RTL to GDSII* en el contexto de un diseño digital ASIC.
- 1.2. Enumerar etapas principales del flujo de diseño físico utilizadas en el trabajo (por ejemplo, síntesis lógica, floorplanning, etc.) y describir en una línea el objetivo de cada una.
- 1.3. ¿Cuál es la diferencia entre un RTL y un netlist?

#### 2. Tecnología y librerías de celdas

- 2.1. ¿Qué es una *celda estándar* (*standard cell*) y qué información provee típicamente una librería de celdas estándar (archivos LIB/NDM/LEF)?
- 2.2. Explicar la diferencia entre celdas HVT, NVT y LVT/SLVT en una tecnología FinFET de 14 nm, y cómo impactan en retardo, consumo dinámico y fuga (*leakage*).
- 2.3. ¿Qué son las celdas *physical only*? Dar al menos dos ejemplos y explicar su función.

#### 3. Floorplanning y red de alimentación

3.1. Mencionar tres decisiones clave que se toman durante el *floorplanning* de un bloque, y cómo afectan el diseño posterior.

3.2. Explicar brevemente el propósito de:

- anillos de potencia (*power rings*),
- mallas o *stripes* de potencia (*power grid*),
- celdas *well tap*.

#### 4. Relojes y CTS

4.1. Definir los conceptos de *skew* de reloj, *latencia temprana* y *latencia tardía*.

4.2. Explicar cuál es el objetivo principal de la etapa de *Clock Tree Synthesis* (CTS) en un flujo de *Place & Route*.

4.3. ¿Por qué se suelen usar reglas especiales de ruteo (NDR, *Non-Default Rules*) para redes de reloj?

#### 5. Temporización estática (STA)

5.1. Definir *setup time* y *hold time* de un flip-flop.

5.2. Explicar qué son *WNS* (*Worst Negative Slack*) y *TNS* (*Total Negative Slack*) y cómo se interpretan en un reporte de STA.

5.3. ¿Por qué se analizan distintos *PVT corners* (por ejemplo, *func setup ss* y *func hold ff*) para verificar temporización?

#### 6. Routing

6.1. Diferenciar entre *enrutamiento global* (*global routing*) y *enrutamiento detallado* (*detail routing*).

6.2. ¿Qué es la *congestión de ruteo* y qué significa que exista *overflow* en una región de GRCs?

6.3. Explicar brevemente qué es el *ECO routing* y en qué situaciones se utiliza.

#### 7. Potencia

7.1. Definir y diferenciar *leakage power* y *switching (dynamic) power*.

7.2. ¿Por qué tiene sentido analizar la potencia en distintos *corners* de funcionamiento?

#### 8. App options y parametrización de la herramienta

8.1. ¿Qué son las *app options* en una herramienta EDA y cómo influyen en el flujo de diseño?

8.2. Dar un ejemplo de ellas y explicar cómo orienta a la herramienta a tomar decisiones.

#### 9. Calidad final del diseño

9.1. Mencionar al menos cuatro tipos de métricas que se utilizan para evaluar la calidad del diseño post-routeo.

9.2. Explicar qué se entiende por *overhead físico* en un bloque digital y qué factores pueden incrementarlo.

#### 10. ALU y arquitectura

10.1. Describir brevemente qué operaciones implementa la ALU de 32 bits utilizada en el trabajo.

10.2. Explicar la función del módulo *clock\_mux* y cómo se implementa el divisor de frecuencia de reloj dentro de la ALU..

#### 11. Análisis del árbol de reloj (parte práctica)

A partir de la tabla de latencias temprana/tardía del árbol de reloj (por ejemplo, Tabla 18):

11.1. Identificar la latencia máxima observada para *CLOCK\_MAIN* en el *corner* de setup y compararla con la de *gen\_clk\_div8*.

11.2. Discutir si las diferencias de latencia entre los relojes derivados *gen\_clk\_div2*, *gen\_clk\_div4* y *gen\_clk\_div8* son razonables para este diseño.

#### 12. Resumen de celdas y tipos de VT

Usando la tabla de resumen de celdas (por ejemplo, Tabla 21):

- 12.1. ¿Qué porcentaje del área total del diseño corresponde a celdas NVT y qué porcentaje a celdas HVT?
- 12.2. Discutir brevemente por qué, en un bloque pequeño como esta ALU, puede predominar NVT sobre HVT y qué impacto tendría cambiar más celdas a HVT.
- 13. Tipos de celdas físicas** A partir de la tabla de referencia de celdas (por ejemplo, Tabla 22):
- 13.1. Identificar qué tipos de celdas corresponden a: end\_cap, well\_tap, corner, lib\_cell.
  - 13.2. Explicar la función de las celdas SAEDRVT14\_CAPSPACER1 y SAEDRVT14\_TAPDS dentro del layout de la ALU.
- 14. Wirelength y uso de capas de metal** A partir del reporte de enrutamiento:
- 14.1. Indicar el *wirelength* total del bloque y qué capa de metal concentra la mayor longitud ruteada.
  - 14.2. Proponer una razón por la cual capas superiores (por ejemplo M8, M9) pueden no haberse utilizado o tener muy poco uso en un bloque pequeño como esta ALU.
- 15. Potencia por modo y corner** Usando la tabla de potencia de la tesis:
- 15.1. Indicar en qué corner se observa la mayor potencia de fuga (*leakage*) y en cuál se observa la mayor potencia dinámica.
  - 15.2. Justificar por qué tiene sentido que la potencia de fuga sea mayor en ciertos corners y la dinámica en otros.
- 16. Constraints de temporización**
- 16.1. A partir de los constraints utilizados, indicar:
    - la máxima transición permitida para las señales;
    - la máxima capacitancia de carga permitida.
  - 16.2. Explicar cómo constraints demasiado agresivos podrían impactar en:
    - el área total,
    - el esfuerzo de la herramienta en optimización de timing.
- 17. Análisis del RTL de la ALU**
- 17.1. Dibujar un diagrama de bloques simple indicando:
    - los puertos de entrada,
    - el núcleo combinacional de operaciones ,
    - el registro de salida,
    - y el registro de o\_valid.
  - 17.2. Por qué crees que es útil registros las señales de entrada y/o de salida?
- 18. Evaluación del flujo de routing**
- 18.1. Enumerar las principales etapas del script de enrutamiento (route\_auto.tcl + route\_opt.tcl) y describir brevemente qué hace cada una.
  - 18.2. A partir de los reportes de congestión y DRC, indicar si el diseño presenta congestión residual significativa y si se observan violaciones DRC relevantes post-routing.
- 19. Comparación pre-CTS vs post-CTS vs post-routeo**
- 19.1. Discutir cómo cambian típicamente las métricas de temporización entre:
    - el diseño tras placement (pre-CTS),
    - el diseño post-CTS,
    - y el diseño post-routing.
- 20. Reflexión sobre el flujo y las herramientas**
- 20.1. Señalar dos ventajas y dos desventajas de utilizar un flujo automatizado con scripts (Tcl) sobre el uso puramente interactivo de la herramienta.
  - 20.2. Proponer al menos una mejora posible para una futura iteración del diseño de la ALU (por ejemplo: incorporar más modos, mejorar el clocking, probar otra mezcla de VTs, optimizar la malla de potencia, cambiar el radio de aspecto del floorplan.).

## 4.2. Guía de soluciones y criterios de corrección

A continuación se presenta una guía sintética de las respuestas esperadas. No se trata de respuestas textuales únicas, sino de los conceptos clave que deberían aparecer en cada ítem.

### 1. Flujo RTL → GDSII

- 1.1. Indicar que es el proceso completo mediante el cual se parte de una descripción a nivel RTL (por ejemplo, Verilog/VHDL) y se llega a un layout físico en formato GDSII listo para fabricación. Incluye síntesis lógica, implementación física (floorplanning, placement, CTS, routing) y verificaciones finales (STA, DRC, etc.).
- 1.2. Mencionar etapas típicas del flujo y su objetivo, por ejemplo:
  - Síntesis lógica: transformar RTL en un netlist de puertas/celdas estándar.
  - Floorplanning: definir área, forma, relación de aspecto, ubicación de macros y anillos de potencia.
  - Placement: ubicar celdas estándar dentro del área disponible.
  - CTS: construir el árbol de reloj, controlando skew y latencia.
  - Routing: enrutar interconexiones (global + detallado) cumpliendo reglas DRC.
  - Signoff: STA final, verificación física (DRC/LVS), análisis de potencia.

- 1.3. Señalar que:
  - RTL: descripción de alto nivel y *comportamental* del circuito (procesos síncronos, operadores aritméticos/lógicos), generalmente independiente de la tecnología.
  - Netlist: descripción *estructural* en términos de instancias y conexiones de puertas/celdas de una librería específica. Es el resultado de la síntesis lógica del RTL.

### 2. Tecnología y librerías de celdas

- 2.1. Una *standard cell* es un bloque lógico predefinido y caracterizado (por ejemplo, INV, NAND, flip-flop) basado en el nodo de tecnología con el cual se está implementando el bloque/chip. La librería incluye:
  - modelos de temporización y potencia (LIB/NDM),
  - vistas físicas (LEF/GDS: dimensiones, pines, capas),
  - restricciones de uso (cargas máximas, transiciones, etc.).
- 2.2. Explicar que:
  - HVT: alto  $V_t$ , menor fuga (leakage), mayor retardo.
  - NVT: valor intermedio de  $V_t$ , compromiso entre performance y fuga.
  - LVT/SLVT: bajo  $V_t$ , mayor velocidad pero mayor fuga.

En 14 nm FinFET se combinan distintas VTs para balancear timing y consumo.

- 2.3. Celdas *physical only*: no implementan lógica funcional, sino que sirven para cumplir reglas físicas de la tecnología. Ejemplos:
  - *well taps*: conectan el pozo a VDD/VSS.
  - *end caps*: terminan las filas y protegen bordes.
  - *fillers*: llenan huecos para asegurar continuidad de alimentación y pozo.

### 3. Floorplanning y red de alimentación

- 3.1. Decisiones esperadas:
  - Definir el tamaño y relación de aspecto del bloque.
  - Ubicar pines y macros (si existen).
  - Diseñar la red de alimentación.
- 3.2. Debe mencionarse que estas decisiones afectan congestión, timing, IR-drop y facilidad de ruteo.
  - *Power rings*: distribuyen VDD/VSS alrededor del bloque.
  - *Power grid/stripes*: llevan la alimentación hacia el interior y hacia las filas de celdas estándar.
  - *Well taps*: fijan el potencial de los pozos y reducen riesgo de latch-up y ruido.

#### 4. Relojes y CTS

- 4.1.
  - Skew: diferencia de tiempos de llegada del reloj entre dos flops (origen y destino).
  - Latencia temprana: menor tiempo de propagación desde la fuente de reloj hasta un punto.
  - Latencia tardía: mayor tiempo de propagación desde la fuente hasta ese punto.
- 4.2. Objetivo principal: construir una red de reloj (árbol o malla) que entregue una señal de reloj con skew y latencias controladas a todos los sinks, respetando restricciones de transición, carga y SI.
- 4.3. Porque el reloj es muy sensible a SI y variaciones. Las NDR permiten:
  - usar metales más anchos y/o más separados,
  - reducir resistencia y acoplamientos,
  - mejorar jitter y robustez de la red de reloj.

#### 5. Temporización estática (STA)

- 5.1.
  - *Setup time*: tiempo mínimo antes del flanco de reloj en que la entrada del flop debe estar estable.
  - *Hold time*: tiempo mínimo después del flanco de reloj en que la entrada debe permanecer estable.
- 5.2.
  - WNS: *Worst Negative Slack*, el slack más negativo del diseño (el peor camino).
  - TNS: *Total Negative Slack*, suma de todos los slacks negativos.

Indican la severidad y cantidad de violaciones de timing.
- 5.3. Porque distintos PVT corners representan condiciones extremas:
  - corner lento (SS, baja tensión, alta T) suele ser crítico para setup,
  - corner rápido (FF, alta tensión, baja T) suele ser crítico para hold.

#### 6. Routing

- 6.1.
  - Enrutamiento global: trabaja sobre celdas de ruteo (GRCs), asigna rutas aproximadas y estima congestión.
  - Enrutamiento detallado: genera las formas metálicas finales cumpliendo DRC y reglas de manufactura.
- 6.2. Congestión: la demanda de pistas excede la capacidad disponible en una región. *Overflow* cuantifica cuántas pistas faltan respecto de la capacidad de las GRCs.
- 6.3. ECO routing: enrutamiento incremental utilizado tras cambios locales (ECOs) en la lógica o el netlist, para evitar rehacer todo el ruteo desde cero.

#### 7. Potencia

- 7.1.
  - *Leakage power*: potencia debida a corrientes de fuga incluso cuando el circuito no commute.
  - *Switching (dynamic) power*: potencia asociada a la comutación de nodos capacitivos ( $P \approx \alpha CV^2 f$ ).
- 7.2. Porque cada corner representa una condición distinta de proceso, voltaje y temperatura, y la potencia (tanto fuga como dinámica) puede variar fuertemente con estos parámetros. Analizar múltiples corners permite dimensionar correctamente consumo en operación y en modos de baja actividad.

#### 8. App options y parametrización de la herramienta

- 8.1. Son parámetros de configuración de la herramienta EDA que controlan esfuerzo, modos y estrategias de optimización (por ejemplo, énfasis en timing, área, potencia, SI). Orientan cómo el motor interno toma decisiones durante síntesis, placement, CTS, routing, etc.
- 8.2. Ejemplo de respuesta:
  - Una opción del tipo `route.detail.timing_driven` o `route.global.timing_driven`: al activarla, el enrutador prioriza redes críticas de temporización frente a redes no críticas.
  - Otra opción posible: un parámetro de esfuerzo de optimización de timing (*high/medium/low*) que hace que la herramienta inserte más buffers, repita más iteraciones, etc.

En todos los casos, la idea clave es que la app option modifica la estrategia interna de la herramienta.

## 9. Calidad final del diseño

9.1. Métricas válidas incluyen, por ejemplo:

- utilización de área,
- WNS/TNS de setup y hold,
- número de violaciones DRC,
- wirelength total,
- número de buffers añadidos,
- consumo de potencia (leakage y dinámica).

9.2. *Overhead físico*: área adicional por encima del mínimo funcional. Se incrementa por:

- buffers/inversores de timing,
- inserción de celdas de relleno, taps, end caps,
- márgenes de SI y constraints agresivos,
- estructuras adicionales (por ejemplo, lógica de test, redundancias).

## 10. ALU y arquitectura

10.1. Debe mencionarse que la ALU implementa al menos:

- operaciones lógicas: AND, OR, XOR,
- operación aritmética.

En el trabajo se ve explícitamente la combinación de módulos xor\_op, sum\_op, and\_op, or\_op y el multiplexor de operación.

10.2. Explicar que `clock_mux` selecciona entre varias frecuencias de reloj: el reloj original y sus divisiones (DIV2, DIV4, DIV8) generadas por registros `q`, `q_2`, `q_3`. El reloj seleccionado gobierna los registros de entrada, de salida y el registro de `o_valid`, ajustando la frecuencia efectiva de operación de la ALU.

## 11. Análisis del árbol de reloj (parte práctica)

11.1. Se debe leer de la tabla de latencias (p. ej. Tabla 18) cuál es la latencia máxima de `CLOCK_MAIN` en el corner de setup y compararla numéricamente con la de `gen_clk_div8`. Se espera un comentario del estilo: “`gen_clk_div8` tiene una latencia algo mayor/menor que `CLOCK_MAIN`”.

11.2. Se espera un juicio cualitativo: si las diferencias de latencia entre `gen_clk_div2`, `gen_clk_div4`, `gen_clk_div8` son pequeñas y coherentes con el árbol de reloj y la lógica de división, entonces el diseño es razonable. Si alguna latencia es muy distinta, puede motivar una discusión sobre el balance del árbol.

## 12. Resumen de celdas y tipos de VT

12.1. Se debe extraer de la tabla (p. ej. Tabla 21) el porcentaje de área asociado a NVT y HVT. Se espera que NVT represente la mayor parte del área y HVT una fracción menor, coherente con los datos de la tabla.

12.2. Explicar que en un bloque pequeño y relativamente sencillo como esta ALU puede priorizarse NVT para facilitar el cierre de timing; aumentar HVT reduciría la fuga pero podría empeorar el slack de setup y requerir mayor esfuerzo de optimización.

## 13. Tipos de celdas físicas

13.1. Relacionar:

- `end_cap`: celdas de terminación de fila.
- `well_tap`: celdas de conexión de pozo.
- `corner`: celdas especiales de esquina (por ejemplo, en pads o bordes del bloque).
- `lib_cell`: celdas lógicas funcionales (INV, AND, FF, etc.).

13.2. ■ SAEDRVT14\_CAPSPACER1: celda de tipo *end cap/spacer* que se coloca al final de las filas para cumplir reglas físicas y de continuidad.

■ SAEDRVT14\_TAPDS: celda *well tap* que conecta el pozo al potencial adecuado, contribuyendo a la integridad eléctrica.

## **14. Wirelength y uso de capas de metal**

14.1. Se debe leer del reporte de routing:

- el wirelength total del bloque (en  $\mu\text{m}$ ),
- la capa que concentra la mayor longitud ruteada (por ejemplo M2, M3 o M4).

14.2. Explicar que en un bloque pequeño la distancia a cubrir es reducida, por lo que las capas bajas/intermedias suelen ser suficientes. Las capas superiores (M8, M9) se reservan generalmente para distribución global de reloj/potencia o para bloques de mayor escala, por lo que pueden no utilizarse o usarse muy poco.

## **15. Potencia por modo y corner**

15.1. El estudiante debe identificar, a partir de la tabla de potencia, qué corner muestra la mayor potencia de fuga y cuál la mayor potencia dinámica. Es razonable esperar:

- mayor leakage en corners con alta temperatura o  $V_t$  bajos,
- mayor potencia dinámica en corners de operación funcional con mayor frecuencia/actividad.

15.2. Justificar con argumentos físicos: la fuga aumenta con la temperatura y con  $V_t$  reducidos; la potencia dinámica crece con frecuencia de reloj y actividad de conmutación.

## **16. Constraints de temporización**

16.1. Se debe tomar de los scripts SDC/constraints:

- el valor de `set_max_transition` (en ns),
- el valor de `set_max_capacitance` (en pF o equivalente).

16.2. Explicar que constraints más estrictos:

- pueden aumentar el área utilizada (más buffers, celdas de mayor tamaño),
- obligan a la herramienta a realizar más iteraciones y optimizaciones de timing,
- pueden aumentar también consumo dinámico por mayor conmutación de buffers.

## **17. Análisis del RTL de la ALU**

17.1. El diagrama de bloques esperado debe incluir:

- puertos de entrada (datos, reloj, reset, enable, operación),
- núcleo combinacional (operaciones lógicas/aritméticas + multiplexor de resultado),
- registro de salida de datos (`o_data`),
- registro de `o_valid`.

No se exige una notación específica, pero sí reflejar claramente bloques y conexiones.

17.2. Se espera que se mencione beneficios de registrar entradas y/o salidas, por ejemplo:

- mejorar el cumplimiento de temporización,
- reducir la carga sobre la lógica combinacional previa/posterior,
- facilitar la integración con otros bloques (interfaces sincronizadas),
- mejorar la robustez frente a glitches y variaciones del camino combinacional.

## **18. Evaluación del flujo de routing**

18.1. Describir, por ejemplo:

- `route_auto.tcl`: llamada a `route_auto` que ejecuta global routing, asignación de tracks y detail routing inicial.
- `route_opt.tcl`: etapas de optimización post-route, ECO routing, búsqueda y reparación de violaciones (DRC/timing/SI), inserción de vías redundantes, etc.

18.2. El alumno debe analizar los reportes para responder si hay:

- overflow residual significativo o no,
- violaciones DRC relevantes post-routing o si el diseño queda "limpio".

## **19. Comparación pre-CTS vs post-CTS vs post-routeo**

### 19.1. Respuesta esperada:

- pre-CTS: timing calculado con reloj ideal, sin red física de reloj, suele ser más optimista.
- post-CTS: se añade el árbol de reloj, aparecen retardos y skew; puede degradar slack de setup y mejorar hold.
- post-routing: se agregan retardos de interconexión finales; el slack puede cambiar nuevamente, apareciendo nuevos caminos críticos o ajustándose los existentes.

## 20. Reflexión sobre el flujo y las herramientas

- 20.1. ■ Ventajas de usar scripts: reproducibilidad, automatización, facilidad para barrer parámetros y versiones, menor error humano en operaciones repetitivas.  
■ Desventajas: curva de aprendizaje más alta, depuración de scripts puede ser compleja, menor interacción visual inmediata.

20.2. Ejemplos de mejoras posibles:

- incorporar más modos de operación o esquinas de análisis,
- ajustar estrategia de clocking o refinamiento del árbol de reloj,
- cambiar la mezcla de VTs para optimizar potencia/timing,
- optimizar la malla de potencia o el radio de aspecto del floorplan,
- refinar constraints (por ejemplo, transiciones/capacitancias) de acuerdo a los resultados obtenidos.

## 5. Bibliografía

- Pola, Ariel (2022). Apuntes de curso, Diseño Digital Avanzado. Fundacion Fulgor
- Dr. V.M. Senthilkumar & Ms.M.Anusha. Lecture notes, VLSI Design, Malla Reddy College of Engineering & Technology
- Synopsis (19 de diciembre de 2022), What is Electronic Design Automation – How it works, <https://www.synopsys.com/glossary/is-electronic-design-automation.html>
- Bhasker, J., Chadha, R. (2009). Static Timing Analysis for Nanometer Designs. Springer.
- Golshan, K. (2007). Physical Design Essentials: An ASIC Design Implementation Perspective. Springer.
- Valencia, M. Joyner (2005). Lecture notes, Intro to VLSI Design.
- Melikyan, V.; Martirosyan, M. (2018). 14nm Educational Design Kit: Capabilities, Deployment and Future. Proceedings of the 7th Small Systems Simulation Symposium
- W. -K. Loo, K. -S. Tan and Y. -K. Teh, "A study and design of CMOS H-Tree clock distribution network in system-on-chip," 2009 IEEE 8th International Conference on ASIC, Changsha, China, 2009
- A. B. Chong, "Hybrid Multisource Clock Tree Synthesis," 2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Dubai, United Arab Emirates, 2021

### 5.1. Videos útiles

1. <https://www.youtube.com/watch?v=iDhpKq09XIo&t=10s> - EDA (Electronic Design Automation) Explained in 90 Seconds | Synopsys
2. <https://www.youtube.com/watch?v=2ehSCWoaoQ> - Intel Fab TOUR | Linus Tech Tips

### 5.2. Recursos útiles recomendados (online y de acceso gratuito)

A continuación se listan algunos recursos externos recomendados para profundizar en los conceptos de diseño físico digital, flujo RTL-to-GDSII y uso de herramientas EDA. Se han seleccionado únicamente recursos cuyo contenido puede consultarse de manera gratuita (aunque algunos ofrecen certificados pagos opcionales).

## Cursos académicos en línea

### ■ NPTEL – “VLSI Physical Design” (IIT Kharagpur)

Curso gratuito que introduce el flujo de diseño físico VLSI, incluyendo algoritmos y estructuras de datos utilizados en *floorplanning*, *placement*, CTS y *routing*. Es especialmente relevante para comprender las etapas de implementación física descritas en este trabajo.

*Sitio web:* <https://nptel.ac.in/courses/106105161>

### ■ NPTEL – “VLSI Physical Design with Timing Analysis” (IIT Roorkee)

Curso que complementa el diseño físico con un fuerte énfasis en análisis de temporización, manejo de *setup/hold*, redes de reloj y cierre de timing, en línea con la sección de CTS y STA desarrollada en esta tesis.

*Sitio web:* <https://nptel.ac.in/courses/108107380>

### ■ edX – “Introduction to VLSI Design” (King’s College London)

Curso introductorio que recorre cómo se diseñan y fabrican circuitos integrados VLSI, con una visión general del flujo RTL-to-GDSII y las principales decisiones de arquitectura y tecnología. La plataforma permite auditar el curso sin coste (el certificado es opcional y de pago).

*Sitio web:* <https://www.edx.org/learn/engineering/kings-college-london-introduction-to-vlsi-design>

### ■ Coursera – “VLSI CAD: Logic to Layout”

Curso orientado a las herramientas CAD que se emplean para llevar un diseño ASIC desde una descripción lógica hasta su implementación física (*layout*). En Coursera es posible acceder al contenido en modo *oyente* (*audit*) de forma gratuita.

*Sitio web:* <https://www.coursera.org/learn/vlsi-cad-logic>

## Programas académicos de EDA

### ■ Programas académicos de Siemens EDA y Synopsys

Tanto Siemens EDA como Synopsys disponen de programas específicos para instituciones educativas (por ejemplo, el *EDA Higher Education Software Program* de Siemens, o el programa académico de Synopsys), que ofrecen acceso sin coste a herramientas EDA y a material formativo en línea para estudiantes y docentes. Estos programas permiten reproducir en el ámbito académico flujos de diseño físico similares a los utilizados en la industria.

En este contexto, resulta especialmente valioso que la institución en la que se desarrolla este trabajo considere formalizar su participación en los programas académicos de los principales proveedores de EDA. La adhesión a iniciativas como las de Siemens EDA o Synopsys no sólo facilitaría el acceso de estudiantes y docentes a herramientas industriales de última generación, sino que también permitiría aprovechar bibliotecas de cursos, seminarios y laboratorios virtuales de forma gratuita. Esto redundaría en una formación más cercana a la práctica profesional, fortalecería las asignaturas de diseño digital y diseño físico, y ampliaría las oportunidades de vinculación entre el ámbito académico y la industria de los circuitos integrados.

## Recursos gratuitos de vendors EDA (vía programas académicos)

### ■ Siemens EDA – EDA Higher Education Software Program

Programa académico que ofrece acceso a herramientas EDA de Siemens y a una biblioteca de cursos y contenidos docentes gratuitos para universidades. Es un punto de partida recomendado para instituciones que deseen incorporar flujos industriales en asignaturas de diseño físico.

*Sitio web:* <https://www.sw.siemens.com/en-US/academic/educators/eda-higher-education-software/>

### ■ Siemens EDA – Plataforma Explore / Aprisa Digital Implementation

Biblioteca de recursos on-demand sobre implementación digital con Aprisa. Siemens indica que, una vez registrado, el acceso a la librería *Explore Aprisa* es gratuito, incluyendo vídeos, seminarios y ejemplos de flujo de *place & route*.

*Sitio web:* <https://blogs.sw.siemens.com/aprisa/>

### ■ Siemens Xcelerator Academy – cursos para estudiantes

Siemens ofrece cursos online gratuitos para estudiantes y docentes a través de Xcelerator Academy, cubriendo diversas áreas de diseño y simulación. Estos recursos pueden complementar la formación en herramientas EDA comerciales.

*Sitio web:* <https://www.sw.siemens.com/en-US/xcelerator-academy-training/>

- **Synopsys – Academic & Research Alliances (SARA) y SolvNetPlus**

A través del programa universitario de Synopsys, las instituciones académicas pueden obtener acceso a herramientas EDA y a material formativo en línea (videos, entrenamientos on-demand) disponible en SolvNetPlus. El acceso al contenido formativo es gratuito para los usuarios académicos incluidos en el programa.

*Sitio web:* <https://www.synopsys.com/academic-research/students.html>