Treball Fi de Grau

Grau d'Enginyeria Electrònica de Telecomunicació

Síntesi lògica i física d'un processador RISC-V en la tecnologia TSMC 65 nm

Manel Amat Huerta

Director: Raimon Casanova Mohr

Dept. De Microelectrònica i Sistemes Electrònics

Escola d'Enginyeria
Universitat Autònoma de Barcelona (UAB)

Febrer 2024

Resum i Paraules clau

Aquest treball té l'objectiu de dur a terme una implementació física del processador CVA6, compatible amb el set d'instruccions RISC-V, mitjançant la tecnologia TSMC de 65nm. Per fer-ho, l'alumne ha realitzat una formació bàsica en l'especialitat de disseny VLSI i ha aplicat els coneixements assolits en desenvolupar aquest projecte, realitzant els processos de síntesi lògica i física a partir de la descripció RTL del *core* CV32A60X amb la finalitat d'obtenir un *layout* funcional a una freqüència de treball raonable. Per assolir aquest propòsit s'ha fet ús dues eines de síntesi de Cadence: Genus Synthesis Solution i Innovus Implementation System.

Paraules clau: VLSI, RISC-V, CVA6, síntesi lògica, síntesi física, standard cell, clock tree, crosstalk.

Resumen y Palabras clave

Este trabajo tiene como objetivo llevar a cabo una implementación física del procesador CVA6, compatible con el set de instrucciones RISC-V, mediante la tecnología TSMC de 65nm. Para ello, el alumno ha realizado una formación básica en la especialidad de diseño VLSI y ha aplicado los conocimientos adquiridos en desarrollar este proyecto, realizando los procesos de síntesis lógica y física a partir de la descripción RTL del *core* CV32A60X con la finalidad de obtener un *layout* funcional a una frecuencia de trabajo razonable. Para lograr este propósito, se han utilizado dos herramientas de síntesis de Cadence: Genus Synthesis Solution e Innovus Implementation System.

Palabras clave: VLSI, RISC-V, CVA6, síntesis lógica, síntesis física, standard cell, clock tree, crosstalk.

Abstract and Keywords

The aim of this project is to carry out a physical implementation of the CVA6 processor, compatible with the RISC-V instruction set, using TSMC 65nm technology. For this purpose, the student has made a basic training in the VLSI design specialty and has applied the knowledge acquired in developing this task, performing the logical and physical synthesis processes from the RTL description of the CV32A60X core in order to obtain a functional layout at a reasonable working frequency. To achieve this objective, two Cadence synthesis tools have been used: Genus Synthesis Solution and Innovus Implementation System.

Keywords: VLSI, RISC-V, CVA6, logic synthesis, physical synthesis, standard cell, clock tree, crosstalk.

Agraïments

Vull donar les gràcies al director del treball, Raimon Casanova, per haver-me donat la oportunitat de formar-me en aquesta especialitat i dur a terme aquest projecte sota el seu mentoratge. La seva experiència, dedicació i orientació han sigut clau per donar forma a la direcció del treball i millorar-ne la qualitat.

Ensems, vull agrair a la meva família, amics i companys de grau pel seu suport durant tot el procés.

Per últim, vull expressar el meu agraïment a la Universitat Autònoma de Barcelona i a OpenHW Group per proporcionar-me els recursos necessaris per dur a terme aquest treball de fi de grau.

Taula de continguts

ÍNDEX DE FIGURES

ÍNDEX DE TAULES

1. INTRODUCCIÓ

En un món on l'electrònica tendeix a ser més petita, ràpida, complexa i energèticament eficient, les tasques de disseny en tecnologies *Very Large Scale Integration* (VLSI) han guanyat cada cop més rellevància dins la societat. Actualment hi ha una alta demanda de professionals degut a l'escassetat d'enginyers que treballen en aquest sector. És per això que aquest projecte s'enfoca en formar a l'estudiant en una de les especialitats dins aquest àmbit, específicament en la síntesi lògica i física. Per tal d'assolir aquest propòsit es fa ús d'una de les arquitectures de software més populars avui dia, RISC-V.

El procés de síntesi consisteix en un conjunt de mètodes i eines de treball emprats per convertir el comportament d'un processador, descrit en un tipus específic de *Hardware Description Language* (HDL) i a un alt nivell d'abstracció, en un *layout* físic del disseny [1][2]. Aquest procés es divideix amb dues fases ben diferenciades:

- El procés de síntesi lògica, on s'utilitza la descripció HDL del disseny del processador per convertir-la en una *netlist*[1] a nivell de portes lògiques. El codi de la descripció HDL ha de ser sintetitzable, és a dir, ha de ser descrita en *Register Transfer Language* (RTL) [2].
- El procés de síntesi física, on es realitza una implementació física de la *netlist* generada com a resultat del procés de síntesi lògica. El resultat generat per la síntesi física és una base de dades de les estructures del *layout* [...] utilitzable durant el procés de manufactura del xip [1].

1.1. Objectius

Aquest treball presenta dos objectius complementaris: proporcionar a l'alumne una formació bàsica en l'especialitat de la síntesi lògica i física i alhora fer ús dels coneixements assolits per realitzar la implementació física d'un processador RISC-V amb la tecnologia TSMC de 65nm.

[1] Document ASCII que descriu la connectivitat elèctrica de l'esquemàtic d'un disseny.

1.2. Metodologia

D'acord amb els objectius establerts, el projecte està estructurat en dues etapes: la formació i la síntesi del processador. La memòria està centrada en l'etapa de síntesi, on la formació només apareix de forma indirecta com a suport per justificar els passos realitzats.

1.2.1. Metodologia de la formació

La formació és l'etapa inicial del treball. Aquest té l'objectiu d'adquirir coneixements sobre els diferents passos del procés de síntesi i dels elements que hi formen part. Per dur a terme aquest objectiu s'ha recollit informació de dues fonts principals:

- Static Timing Analysis for Nanometer Designs: A Practical Approach. Aquest llibre de text serveix com a introducció per d'anàlisi temporal estàtic d'un disseny sintetitzat, mètode que és farà servir durant el procés de síntesi lògica en el capítol 3.
- A Practical Approach to VLSI System on Chip (SoC) Design: A Comprehensive Guide. Aquest és un llibre de text dirigit a graduats en enginyeria electrònica i elèctrica que aspiren a ser dissenyadors VLSI. El llibre abasta de forma exhaustiva les bases de l'especialitat sense arribar a aprofundir gaire en temes més complexos, cosa que sí fa el llibre anterior i, per tant, el complementa.

Les fonts addicionals consultades en aquest projecte es troben enumerades a la bibliografia.

1.2.2. Metodologia de la síntesi

La memòria compta amb 3 capítols principals que recullen tant la informació tècnica essencial pel correcte desenvolupament del projecte com la feina realitzada per l'alumne:

- Un capítol dedicat a l'estat de l'art del processador que s'ha fet servir. El capítol es centra en la seva arquitectura i funcionament que, posteriorment, es sintetitzarà per mitjà de la seva descripció RTL.

- Un capítol on es descriu el procés de síntesi lògica i es realitza un conjunt d'anàlisis temporals per verificar el rendiment del circuit lògic generat com a resultat del procés de síntesi.
- Un capítol on es mostra la implementació física del disseny del processador mitjançant el procés de síntesi física fent ús de l'eina de síntesi Innovus Implementation System de Cadence.

Durant l'execució dels processos de síntesi és necessari disposar de determinats tipus de fitxers corresponents a la tecnologia en que es vol implementar el disseny. En el cas d'aquest projecte es faran servir fitxers de la tecnologia TSMC 65nm proporcionada per la Taiwan Semiconductor Manufacturing Company (TSMC).

Respecte al disseny del processador a utilitzar, es tracta d'un processador CV32A6 de la família CORE-V, conegut generalment com CVA6, el qual és una variant *single core* optimitzada i *open source* dels processadors RISC-V. Al tractar-se d'un disseny *open source*, els arxius RTL són fàcilment accessibles i disposa de documentació molt clara [3].

1.3. Per què RISC-V?

RISC-V és una arquitectura de software, també coneguda com *Instruction Set Architecture* (ISA), que està despertant interès dins la industria degut a les seves característiques poc habituals en el mercat dels processadors: accés lliure i modularitat [4][5]. Actualment aquest mercat es troba dominat per les arquitectures x86 i ARM:

- L'arquitectura x86 d'Intel és d'ús exclusiu per 3 companyies a nivell global [6]: AMD, la pròpia Intel i Zhaoxin, una companyia xinesa que fabrica processadors pel mercat xinés [7].
- L'arquitectura ARM funciona amb un model per llicències, de forma que és possible adquirir el *core* però no es pot modificar el conjunt d'instruccions [5]. Aquesta restricció comporta un problema de malbaratament d'àrea i consum, ja que depenent de l'aplicació hi hauran instruccions que no es faran servir, consumint àrea i consum de forma innecessària.

A diferència de les arquitectures x86 i ARM, RISC-V no té propietat intel·lectual i és modular, de forma que es pot dissenyar el processador escollint els blocs de hardware, els subconjunts d'instruccions i el tipus d'arquitectura (32-bits o 64-bits) necessaris pel tipus d'aplicació que es vol fer servir [3][4]. Aquesta llibertat de disseny, juntament amb el fet que és una tecnologia dissenyada amb el propòsit de ser simple, eficient i escalable, li permet funcionar com un *open standard* pel disseny de processadors i ha motivat a empreses clau, com Qualcomm i Alphabet (Google), a apostar per aquesta nova arquitectura de software [4].

La tracció que RISC-V ha generat entre empreses tecnològiques punteres ha fet créixer l'ecosistema empresarial de RISC-V en els últims anys, creant diversos punts de reunió a gran escala on professionals de la industria es reuneixen a dissertar sobre innovació i millores de la fórmula ja establerta. Es creu que, a futur, el seu creixement continuarà augmentant degut als avantatges que suposa la seva modularitat en el disseny de processadors vectorials en comparació als sets d'instruccions *Single Instruction Multiple Data* (SIMD) de la tecnologia x86 [8].

És per aquests motius que s'ha considerat adequat fer servir un processador RISC-V per a que l'estudiant pugui formar-se i realitzar el treball que ens ocupa.

2. ESTAT DE L'ART: ESTRUCTURA I FUNCIONAMENT DEL PROCESSADOR CVA6

Aquest capítol proporciona una vista ràpida del funcionament del CV32A6 i, per extensió, dels fitxers RTL utilitzats per realitzar el procés de síntesi lògica. La informació mostrada al llarg d'aquest capítol ha sigut extreta de la documentació del processador publicada per OpenHW Group [3].

2.1. Introducció

En primer lloc, és important conèixer la distinció entre CVA6 i CV32A6 per tal d'entendre correctament l'estructura i la nomenclatura del processador. El nom CVA6 és una generalització que engloba les *application processor cores* de la família CORE-V (CV) que presenten una *pipeline*[2] de 6 etapes (A6). Dins d'aquesta generalització podem distingir diferents nuclis o *cores* que poden estar configurats per funcionar com un sistema de 32 bits (RV32) o de 64 bits (RV64). En el cas específic d'aquest projecte, el *core* que s'ha fet servir és el CV32A6. Alhora, cada *core* pot tenir diferents versions depenent de com s'han configurat els seus paràmetres (registres L1, unitats funcionals opcionals com MMU, FPU...) [3]. Com que a la secció 2.3 s'aprofundirà més en detall sobre aquests paràmetres i les configuracions disponibles, de moment només és important conèixer que en el context d'aquest treball farem servir la configuració bàsica (CV32A60X).

Pel bé de la simplicitat, i tenint en compte que dins l'abast d'aquest treball només utilitzem del *core* CV32A6 amb la configuració CV32A60X, d'ara endavant es farà ús del terme CVA6 per fer referència a aquest *core* en específic.

CVA6 és una versió optimitzada del processador ARIANE, creat per ETH Zurich i la University of Bologna. El codi de CVA6 ha sigut escrit en SystemVerilog i mantingut per OpenHW Group [3].

Tal com la documentació [3] explica, CVA6 presenta una estructura amb *pipeline* de 6 etapes *in-order single issue* que, en el cas del *core* utilitzat en aquest projecte, implementa un set d'instruccions RISC-V de 32 bits. El propòsit d'aquesta CPU és poder córrer un sistema operatiu a una velocitat i

[2] Conjunt d'elements de hardware que executen una instrucció de forma seqüencial, on la sortida d'un element és l'entrada del següent.

nombre d'Instruccions Per Cicle (IPC) dins del rang del que es consideren valors raonables. Per tal d'assolir aquests objectius, l'estructura de CVA6 està dissenyada de forma que la *pipeline* de 6 etapes permet aconseguir la velocitat desitjada i utilitza una *scoreboard*[3] per arribar a un nombre d'IPC prou elevat.

Addicionalment, el processador compta amb tots els requeriments de mode privilegiat de RISC-V per poder córrer un sistema operatiu com Linux.

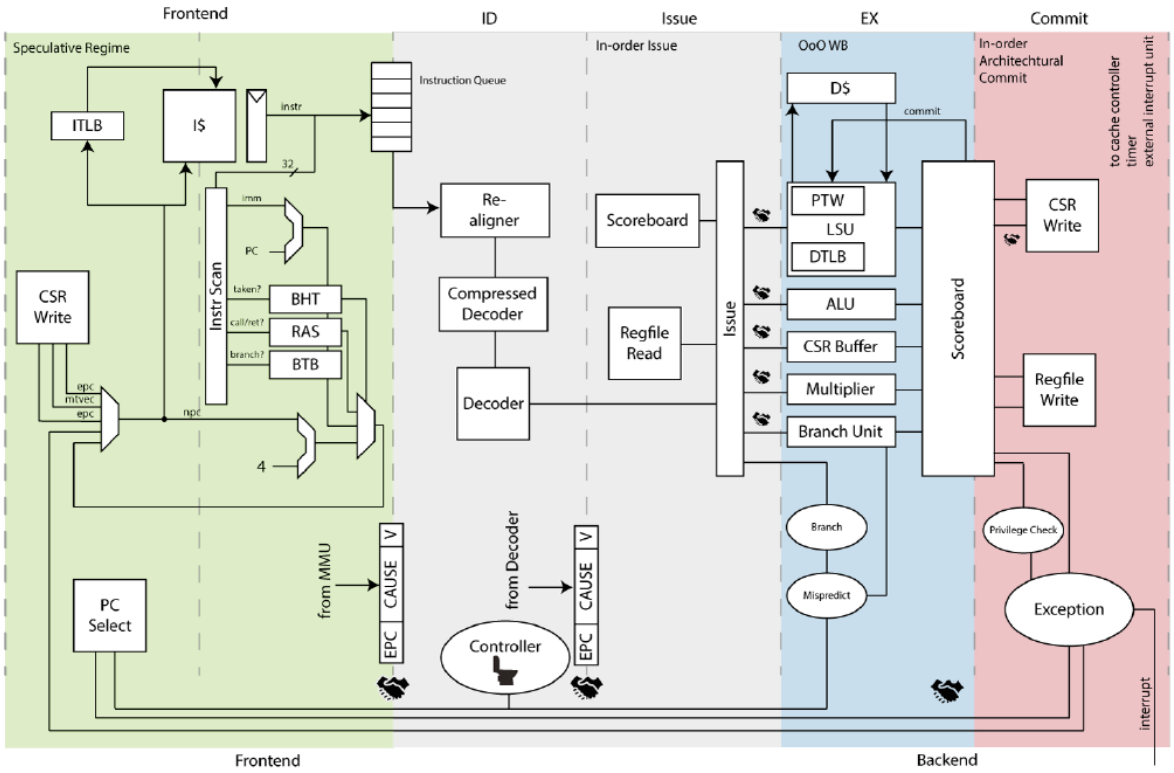


Fig 1. Esquema de l'estructura del processador CVA6.
OpenHW Contributors, "CVA6 User Manual," 2023. Accessed: Dec. 01, 2023. [Online]. Available: https://github.com/openhwgroup/cva6/tree/master

2.2. Arquitectura i mòduls

L'arquitectura hardware del processador gira entorn a la *pipeline* i els 8 mòduls que la vertebren: FRONTEND, DECODE, ISSUE, EXECUTE, COMMIT, CONTROLLER, CACHES i CSRFILE [3]. Els 6 primers mòduls s'encarreguen de dur a terme les 6 etapes de la *pipeline* mentre que el mòdul

[3] Bloc de hardware encarregat de controlar el flux de dades entre registres i unitats funcionals.

CACHES implementa els *caches* de dades i instruccions i el mòdul CSRFILE conté els registres. Les 6 etapes de la *pipeline* són les següents:

- Frontend: consta de dues sub-etapes:
    - Generació del *Program Counter* (PC): genera el següent PC.
    - Cerca d'instrucció: controla el mòdul CACHES, demanant dades o realineant-les a la cua d'instruccions per enviar-les posteriorment al mòdul DECODE.
- ID (Instruction Decoding) : interpreta les instruccions enviades des de l'etapa de Cerca d'instrucció per descodificar les i enviar la informació al mòdul ISSUE.
- Issue: rep les instruccions descodificades i les envia a les unitats funcionals (ALU, *Branch Unit*, *Load Store Unit* [LSU], *Memory Management Unit* [MMU], *Multiplier* i *CSR Buffer*) corresponents.
- EX (Execution): s'encarrega de controlar cadascuna de les unitats funcionals de forma independent per executar les instruccions enviades pel mòdul ISSUE.
- Commit: porta un registre de les instruccions executades i actualitza l'estat arquitectural. Per dur a terme aquesta funció, es modifiquen els registres del mòdul CSRFILE.

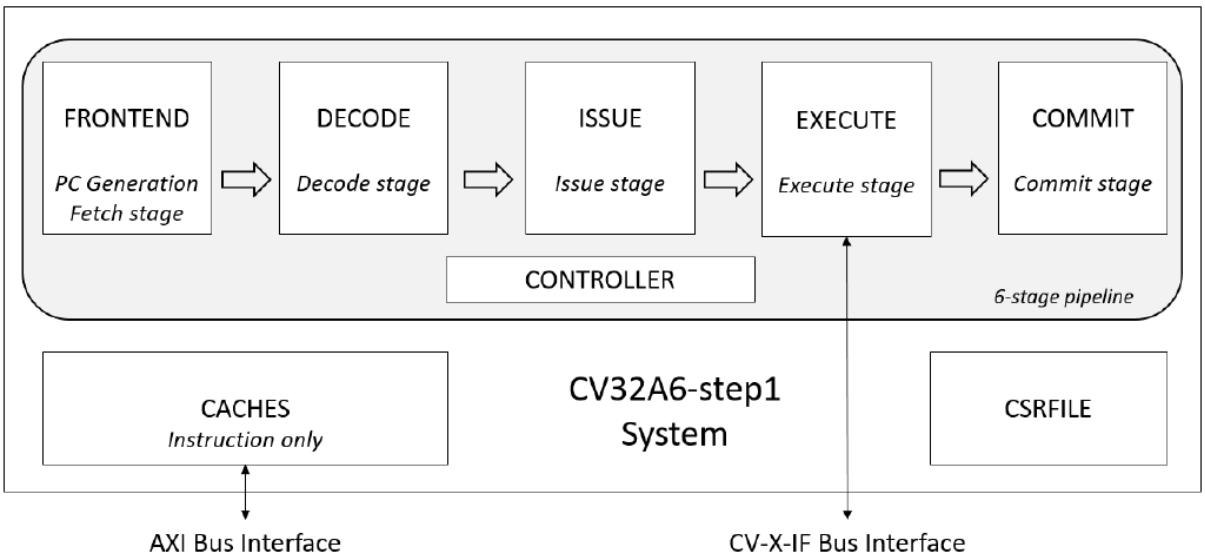


Fig 2. Diagrama de blocs de la pipeline i els mòduls que la formen.
OpenHW Contributors, "CVA6 User Manual," 2023. Accessed: Dec. 01, 2023. [Online]. Available: https://github.com/openhwgroup/cva6/tree/master

2.3. Paràmetres de CV32A60X

Degut a que CVA6 és una CPU on els seus paràmetres són altament configurables, OpenHW Group proposa una sèrie de configuracions que ells mateixos verifiquen [3]. Entre els paràmetres a tenir en compte a l'hora d'escollir la configuració es troben:

- *Target*: El tipus de circuit integrat.
- XLEN: La longitud de la cadena de dades del processador RISC-V.
- *Floating Point Uni*t (FPU): Coprocessador encarregat de realitzar operacions matemàtiques amb coma flotant.
- CV-X-IF: Es tracta d'una funcionalitat d'interfície disponible en algunes configuracions que permet vincular diversos coprocessadors simultàniament.
- *Memory Management Unit* (MMU): S'encarrega de la traducció d'adreces i dels accessos a memòria.

ID Configuration	Target	ISA	XLEN	FPU	CV-X-IF	MMU	L1 D$	L1 I$
CV32A60X	ASIC	IMC	32	Yes	Yes	Sv32	None	16kB
cv32a6_imacf_sv32	FPGA	IMACF	32	Yes	TBD	Sv32	32kB	16kB
cv32a6_imac_sv32	FPGA	IMAC	32	No	TBD	Sv32	32kB	16kB
cv64a6_imacfd_sv39	ASIC	IMACFD	64	Yes	Yes	Sv39	16kB	16kB
Cv32a6_imac_sv0	ASIC	IMAC	32	No	Yes	None	None	4kB

Taula 1. Configuracions del core CV32A6.

Quan es va iniciar l'etapa de síntesi d'aquest treball de fi de grau, al Desembre de 2023, l'única configuració del *core* CV32A6 que ja havia sigut verificada era la configuració base, CV32A60X. És per aquest motiu que es va escollir aquesta configuració, la qual presenta una *data length* de 32 bits, una memòria sv32 i un registre d'instruccions de 16kB tal com es mostra a la Taula 1.

Lamentablement, no s'ha pogut aconseguir accés a les memòries SRAM. Es faran servir *flip flops D* en el seu lloc, factor que es reduirà el rendiment del disseny sintetitzat durant els capítols 3 i 4.

Per últim, cal afegir que el codi RTL (el qual descriu el funcionament i estructura del CVA6 i que es farà servir en els següents capítols del treball) es troba dins d'un repositori GitHub públic d'OpenHW Group. L'enllaç al repositori es referenciat al final del document [3].

3. SÍNTESI LÒGICA

Aquest capítol recull el procés de síntesi lògica del sistema CVA6 fent servir Genus Synthesis Solution en mode *Stylus Common UI.* L'objectiu d'aquest capítol es obtenir una síntesi lògica el més optimitzada possible. Per fer-ho, primer és necessari realitzar una primera síntesi on es tingui en compte el rendiment temporal. Posteriorment, es podrà optimitzar altres aspectes del disseny com el consum.

Per comprovar l'eficàcia de la optimització, es duu a terme un anàlisi temporal estàtic (*Static Timing Analysis*; STA), el qual és un mètode habitual en síntesi per analitzar les característiques temporals del disseny i verificar que aquest funciona correctament a una freqüència especifica. El STA es realitza per blocs, els quals reben el nom de *Design Under Anlysis* (DUA). En el cas del disseny que abasta aquest projecte inclou un únic bloc i per tant no és necessari fer distincions entre elements interiors i exteriors al DUA [9] [10].

La *netlist* generada conté informació rellevant que pot ser analitzada per dur a terme una optimització més eficaç. L'objectiu principal és optimitzar el *timing* però, considerant que es tracta d'un processador, és recomanable mantenir uns valors acceptables de velocitat en termes de freqüència. És per aquest motiu que s'han realitzat un conjunt de processos de síntesi amb la intenció de comprovar la velocitat màxima a la que pot arribar el processador abans de començar a tenir *timing violations*.

3.1. Estructura dels fitxers RTL

El repositori CVA6, el qual conté la descripció d'alt nivell del processador, es troba estructurat de forma que cada directori top-level conté funcionalitats externes al disseny mateix. En el context d'aquest projecte, s'ha fet servir la mateixa estructura amb modificacions en el directori pd/synth [3]:

- ci: *Scriptware* per CI.
- common: Conté fitxers de codi font fets servir pel *core* CVA6 i un SoC anomenat COREV APU dissenyat per aplicacions amb FPGAs i que es troba fora de l'abast del projecte.
- core: És on es troba el codi font del CVA6. Conté els fitxers RTL de les variacions de 32 i 64 bits juntament amb els arxius de les configuracions disponibles.

- corev_apu: Conté el codi font del SoC. Es tracta d'una *Accelerated Processing Unit* (APU) molt simple que afegeix una sèrie de perifèrics al *core* CVA6: UART, GPIO, un controlador de SRAM...
- pd: És el directori principal de treball on es dur a terme aquest projecte. Conté els fitxers de síntesi lògica i física:
    - common: Conté fitxers que es fan servir en els diversos scripts de síntesi lògica i física.
    - synth: Conté els scripts de síntesi lògica. Aquest directori ha sigut modificat per integrar la tecnologia TSMC de 65nm dins del disseny.
    - pnr: Conté els scripts de síntesi física.
    - workdir: És el directori de treball on es realitza la síntesi lògica i física i s'emmagatzemen els fitxers de sortida generats durant el procés.
- util: Conté les toolchains.
- vendor: Conté blocs/mòduls (IPs) reutilitzades d'altres dissenys que es fan servir al CVA6 com per exemple una ALU FP.
- verif: És l'entorn de verificació del disseny.

3.2. Desenvolupament del procés de síntesi lògica

Per dur a terme la síntesi lògica es requereixen 3 tipus de fitxers que serveixen com a input per Genus durant el procés [2][11]:

- Codi RTL del CVA6: Conté el la descripció HDL del *core* en format de text.
- Biblioteques (*libraries*) de la tecnologia TSMC 65nm que aporten informació sobre les cel·les (cells) que s´hi insereixen per crear el circuit lògic:
    - Biblioteques *standard cell*: Contenen l'estructura de les cel·les i s'explicarà en detall posteriorment en aquest capítol.
    - *Liberty files*: Contenen informació temporal i de consum de les *standard cells* a més de models temporals associats amb altres fenòmens físics (*paths delays*, *interconnect*

delays) que només es tindran en compte durant la síntesi física en el capítol 4 [12]. Les *liberty files* utilitzades en aquest projecte es troben caracteritzades per 7 condicions de treball principals, 42 si es sumen les múltiples variacions disponibles. Tenint en compte que l'objectiu d'aquest capítol és obtenir una *netlist* que sigui funcional en el major nombre de situacions possible amb els recursos que tenim disponibles, es faran servir les condicions de treball on les portes lògiques són més lentes (WCCOM).

Corner	Voltage (V)	Temperature (ºC)	Process (FF: Fast; TT: Typical; SS: Slow)
BCCOM (Best Case)	1.1	0	FF
LTCOM (Low Temperature)	1.1	-40	FF
MLCOM (Militar)	1.1	125	FF
TCCOM (Typical Case)	1	25	TT
WCCOM (Worst Case)	0.9	125	SS
WCLCOM (Worst Case Low temperature)	0.9	-40	SS
WCZCOM (Worst Case Zero temperature)	0.9	0	SS

Taula 2. Caracterització base dels PVT corners.

- Biblioteques PAD: Contenen diferents tipus de cel·les pad utilitzades per els fitxers RTL. [1]
- Biblioteques macro: Són una extensió de les biblioteques de *standard cells* que conté blocs funcionals creats amb aquesta tecnologia. [1]

- *Standard Design Constraint* (SDC) on queden definits els paràmetres desitjats d'àrea (*area constraint*), de temps (*timing constraint*) o de consum (*universal power constraint*) [11]. Tal com s'ha esmentat a la introducció del capítol, el fitxer SDC d'aquest projecte està únicament format per *timing constraints* centrats en definir paràmetres temporals com la freqüència de treball, els retards de input-output, etc. En la següent secció del capítol s'explicaran més en detall els *constraints* utilitzats.

El flux de la síntesi lògica es pot dividir en diverses etapes que, executades seqüencialment, ens permeten obtenir una *netlist* amb les especificacions desitjades. La Figura 3 indica el flux de síntesi habitual [2][11]:

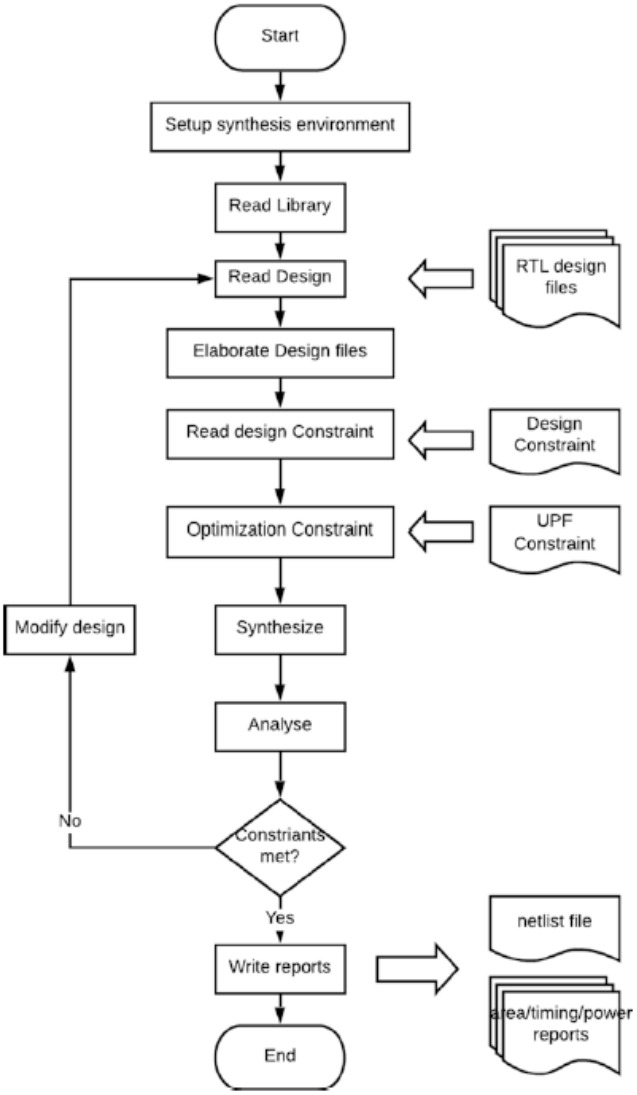


Fig 3. Diagrama de flux del procés de síntesi lògica.
V. S. Chakravarthi, "Fig. 5.2 SOC synthesis flow," in A Practical Approach to VLSI System on Chip (SoC) Design, Springer Nature Switzerland, 2020.

- Setup synthesis environament: Indica a l'eina de síntesi les estructures de les biblioteques que cal llegir durant les diferents etapes i assigna el directori on emmagatzemar els fitxers de sortida generats durant el procés de síntesi.
- Read library: Carrega les biblioteques de la tecnologia (*standard*, *liberty*, PAD i macro).
- Read design: Carrega els fitxers RTL del disseny VLSI i verifica que siguin sintetitzables.

- Elaborate design files: Executa l'eina *elaborate*, la qual s'encarrega d'identificar cadascun dels objectes definits en els fitxers RTL i eliminar lògica redundant.
- Read design constraint: Llegeix les restriccions del fitxer SDC que ha de definir el rendiment de les *standard cells*.
- Optimization constraint: Tal com s'ha mencionat prèviament, la síntesi ha de tenir com a objectiu la optimització d'un factor de rendiment (àrea, temps o consum). Aquest objectiu s'indica com a restricció en el fitxer SDC.
- Synthesize: És l'etapa més important. Converteix la descripció RTL del disseny VLSI en una *netlist* a nivell de portes lògiques.
- Analyse: Analitza la netlist generada per verificar que compleix les especificacions esperades.
- Modify design: Aquesta etapa només té lloc si les especificacions no es compleixen. Caldrà repetir el procés variant els paràmetres que es creguin oportuns per obtenir el resultat desitjat.
- Write reports: Un cop verificat que la netlist compleix amb les especificacions, l'eina de síntesi genera informes sobre diversos àmbits de la síntesi, generalment sobre l'àrea, el temps i el consum.

Tot i que l'eina de síntesi té un flux fixe, aquest pot variar segons el interessos de l'usuari. Es per això que resulta pertinent introduir les instruccions d'execució de les diverses accions que l'eina té a la seva disposició. Hi ha dues formes de fer-ho: introduint les instruccions de forma manual mitjançant el *shell* de Genus o realitzant una *batch execution* per mitjà de scripts. En el cas d'aquest projecte, s'ha optat per crear un script per cadascuna de les diferents etapes juntament amb un script de control encarregat d'executar en cadena la resta de scripts. D'aquesta manera es pot automatitzar el procés i anar ajustant els paràmetres fins obtenir els resultats de síntesi desitjats:

- setup.tcl: Realitza les etapes de Setup, Read library i Read design.
- elaborate.tcl: S'encarrega de dur a terme les etapes de Elaborate design files, Design constraints i Optimization constraint.
- synthesis.tcl: Realitza l'etapa de Synthesize.
- export.tcl: S'encarrega d'emmagatzemar els fitxers de sortida del flux.

Els scripts de síntesi es troben annexats al final del document (Annex 1).

3.3. Anàlisi de la netlist generada

3.3.1. Informació prèvia

El procés de síntesi, tant la part lògica com la física, conté una considerable càrrega teòrica que és important conèixer per poder seguir correctament l'anàlisi dels resultats tant en aquest capítol com en el capítol 4. Aquesta secció pretén sintetitzar tots els conceptes rellevants i la seva relació entre ells.

3.3.1.1. Standard cells

Abans de continuar, és important entendre el concepte de *standard cell* i com aquest es relaciona amb la *netlist*. Conceptualment, la *netlist* és una representació estructural del model de comportament descrit en els fitxers RTL, és a dir: un circuit completament digital [2]. Per implementar-ho, l'eina de síntesi fa ús del que es coneixen com *standard cell libraries*, les quals són una col·lecció de cel·les i blocs lògics predissenyats i prevalidats [11]. Dins aquestes biblioteques podem trobar elements bàsics com portes lògiques i blocs bàsics com *flip flops*, *latches*, *buffers*, etc [11]. D'aquesta forma, la *netlist* queda formada per flip flops que actuen com a registres per llençar i rebre dades. Més endavant entrarem en detall sobre la importància dels registres dins l'anàlisi.

La tecnologia de *standard cells* que es fa servir (TSMC de 65nm) disposa de 2 biblioteques: *Low Power* (LP) i *General Purpose* (GP). La tecnologia LP és *low leakage*, de forma que presenta un consum estàtic molt baix però té l'inconvenient que les seves portes són més lentes que les de la biblioteca GP. Per altra banda, la tecnologia GP té portes més ràpides però un consum estàtic més alt que a LP.

Dins de cada biblioteca, els fabricants acostumen a donar 3 tipus de *standard cells*:

- Unes basades en transistors amb tensions llindar normals.
- Unes basades en transistors amb tensions llindar altes.
- Unes basades en transistors amb tensions llindar baixes.

Els transistors amb una tensió llindar alta presenten uns corrents de fugues baixos però tenen dificultats per arribar a la tensió llindar i commutar, fet que provoca que les portes siguin més lentes.

Els transistors amb una tensió llindar baixa presenten els efectes inversos: commuten amb més facilitat, fent les portes més ràpides però generant uns corrents de fugues alts, augmentant així el consum per *leakage*.

3.3.1.2. Temps de transició

Com aquests 3 tipus de transistors es troben dins la mateixa biblioteca, és tasca de l'eina de síntesi decidir quan posar cada tipus de transistor. És aquí on entra la optimització. Per defecte, l'eina de síntesi optimitza prioritzant sempre la velocitat per sobre de la resta de paràmetres. Això significa que l'eina minimitzarà, inserint sempre el tipus de transistors amb les portes més ràpides però amb l'inconvenient d'incrementar el consum. La solució és definir un temps mínim de transició de pujada i baixada en els flancs de rellotge per tal de limitar l'optimització de la velocitat del circuit i així mantenir un equilibri entre velocitat i consum. D'igual forma, també és recomanable definir un temps màxim de transició de pujada i baixada per evitar que l'eina de síntesi optimitzi massa el consum a costa de sacrificar la velocitat del circuit. Aquests paràmetres temporals són definits en *els timing constraints* del fitxer SDC que l'eina de síntesi fa servir com a input. A la Figura 4 es mostra el fitxer SDC utilitzat, on s'han definit paràmetres únicament temporals com la freqüència de treball (*set clk_period*), la definició del rellotge (*create_clock*) i els temps de transició de pujada (*set_clock_transition -max*) i baixada (*set_clock_transition -min*).

```
1 set clk_name                main_clk
2 set clk_port                clk_i
3 set clk_ports_list          [list $clk_port]
4 set clk_period              12.5
5 set input_delay             1
6 set output_delay            1
7
8 #Clock definition
9 create_clock [get_ports $clk_port] -name $clk_name -period $clk_period
10
11 #Clock network latency
12 set_clock_latency -max 0.8 $clk_name
13
14 #Clock skew can lead to setup violations. Possible value of skew is provided to DC so that it can model for that. Generally setup uncertainty is taken as 10% of the clock.
15 set_clock_uncertainty -setup 1.2 $clk_name
16
17 #Clock skew can lead to hold violations. Possible value of skew is provided to DC so that it can model for that. Generally hold uncertainty is taken as 5% of the clock
18 set_clock_uncertainty -hold 0.6 $clk_name
19
20 #Clock transition
21 set_clock_transition -min 0.5 $clk_name
22 set_clock_transition -max 1 $clk_name
23
24 set_input_delay $input_delay -clock $clk_name [get_ports rst_ni]
25 set_input_delay $input_delay -clock $clk_name [get_ports boot_addr_i]
26 set_input_delay $input_delay -clock $clk_name [get_ports hart_id_i]
27 set_input_delay $input_delay -clock $clk_name [get_ports irq_i]
28 set_input_delay $input_delay -clock $clk_name [get_ports ipi_i]
29 set_input_delay $input_delay -clock $clk_name [get_ports time_irq_i]
30 set_input_delay $input_delay -clock $clk_name [get_ports debug_req_i]
31 set_input_delay $input_delay -clock $clk_name [get_ports cvxif_resp_i]
32
33 set_output_delay $output_delay -clock $clk_name [get_ports *_o]
34
```

Fig 4. Fitxer de constraints utilitzat per realitzar una síntesi tecnologia 80MHz.

3.3.1.3. Clock tree

Els paràmetres d'incertesa (*set_clock_uncertainty*), latència de xarxa (*set_clock_latency*), *input delay* (*set_input_delay*) i *output delay* (*set_input_delay*) definits en el fitxer SDC de la Figura 4 es troben estretament relacionats amb les característiques temporals analitzades al STA i, per extensió, al *clock tree*. Un *clock tree* es una xarxa de distribució de rellotges que abasta tot el disseny. Aquests rellotges es connecten a diferents registres a través de cadenes d'inversors o *clock buffers* que alhora es separen en sub-cadenes, ramificant-se i formant un arbre. Els *end points* del *clock tree* són coneguts com a fulles o *leafs* [12].

La latència de xarxa (*clock network latency*) és el retard de propagació dins el DUA des que es genera el senyal de rellotge fins que aquest arriba al pin de rellotge d'una fulla degut a la presència de cadenes d'inversors en el camí. Per tant, es tracta d'un valor fixat a la pròpia estructura del *clock tree* [12]-[14].

Teòricament, el senyal de rellotge hauria d'arribar a totes les fulles del *clock tree* al mateix temps, formant el que es coneix com a *clock tree* balancejat. Lamentablement, això no és possible aconseguir-ho en la pràctica degut al *skew*, una desviació (positiva o negativa) en el temps de propagació del *timing path* que genera una diferència de temps entre fulles. Addicionalment, el rellotge es veu modificat per el *jitter*, un soroll que s'introdueix en el disseny i provoca desviacions en el període del rellotge. La suma de les desviacions temporals causades per el *skew* i el *jitter* és coneguda com a incertesa (*uncertainty*). Degut a la naturalesa aleatòria del soroll, aquestes variacions poden suposar un problema per dissenys d'alta velocitat (>400MHz) [10][12][13].

És precís tenir en compte que el procés de generació del *clock tree*, conegut com *a Clock Tree Synthesis* (CTS), es duu a terme durant el procés de síntesi física. Els paràmetres descrits al fitxer de *constraints* utilitzat simulen els efectes que produiria la presència d'un *clock tree* a l'anàlisi temporal, com per exemple la incertesa [13].

S'ha considerat interessant per els objectius del treball dur a terme la tasca d'avaluar els efectes temporals que ambdues biblioteques produeixen en el rendiment de la *netlist*.

3.3.2. Anàlisi amb la tecnologia LP

La metodologia que s'ha fet servir per analitzar els resultats consisteix en revisar els *timing paths*[4] de la *netlist* generada en busca de violacions temporals. Els *timing paths* més comuns d'analitzar són els *register-to-register* per la seva simplicitat. En els *timing paths register-to-register*, el *flip flop* de llançament envia la dada al *flip flop* de captura encarregat de emmagatzemar-la. Per garantir que l'emmagatzematge s'ha realitzat amb èxit és essencial que el temps d'arribada (*arrival time*) de la dada sigui inferior al temps requerit (*required time*) de *setup* o *hold* en funció de si els temps s'estan mesurant en el *flip flop* de llançament o en el de captura respectivament. Aquesta diferència entre el temps d'arribada de la dada i el temps requerit és conegut com *slack* i serveix com indicador de l'eficiència del *timing* en el disseny [12][13]:

- Si el *slack* és negatiu, el *flip flop* pot ser que entri en metastabilitat o que el valor emmagatzemat sigui el correcte. Es considera una violació temporal i cal fer canvis en el disseny o els *constraints* de la síntesi per corregir-ho.
- Si el *slack* és positiu, la dada s'emmagatzema correctament al flip-flop. Això implica que encara hi ha espai per optimitzar el disseny.
- Si el *slack* és zero, la dada arriba just a temps. No hi ha marge per l'optimització.

Com en aquest cas es pretén realitzar un anàlisi temporal de la *netlist* lògica, no es disposa d'un *clock tree*. Això significa que no té sentit analitzar els temps de *hold* i, per tant, només s'analitzaran els temps de *setup* i en les condicions de treball en que les portes són més lentes.

Per dur a terme l'anàlisi es realitzaran una sèrie de processos de síntesi lògica amb el fitxer de *constraints* configurat a diferents freqüències. S'ha començat amb la tecnologia de porta més lenta (LP) a una freqüència baixa (80MHz) per poder comprovar que la síntesi lògica es realitza amb èxit, generant una *netlist* funcional.

[4] Conjunt de paths agrupats en funció del endpoint del path. Cada rellotge està associat a un conjunt de paths.

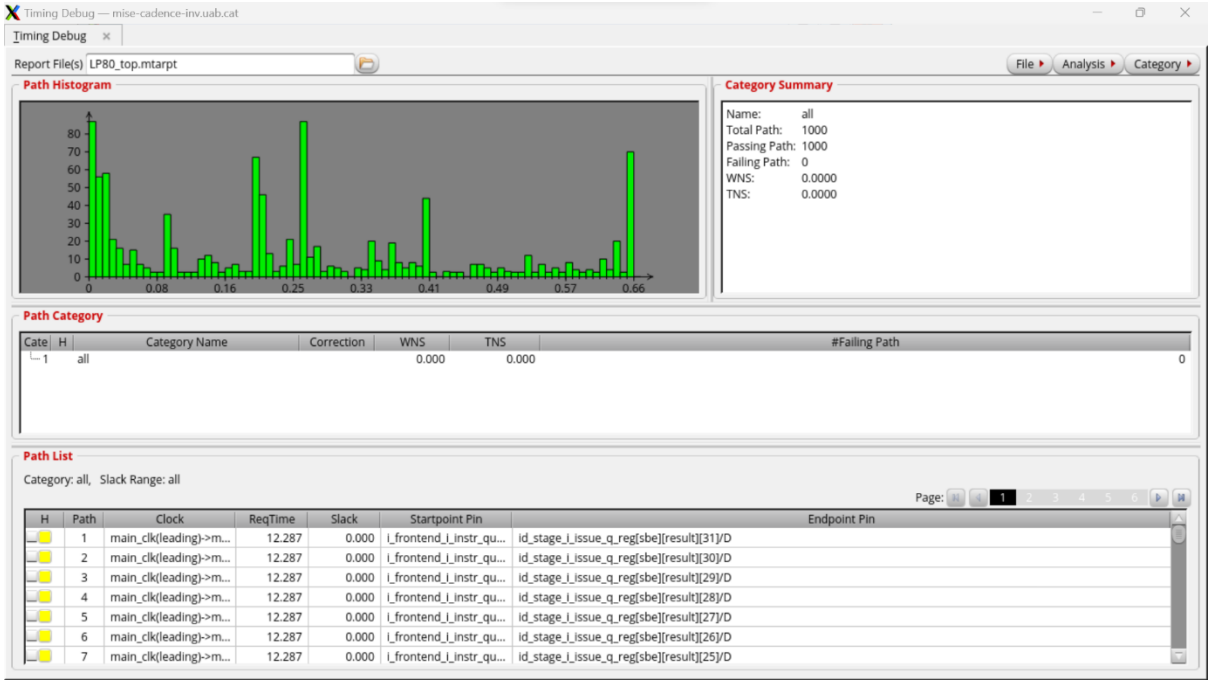


Fig 5. Timing debug per una freqüència de 80MHz amb tecnología de porta LP.

En la Figura 5 es poden observar com tots els *paths* de la *netlist* es troben en verd, indicant que no pateixen de cap *timing violation*.

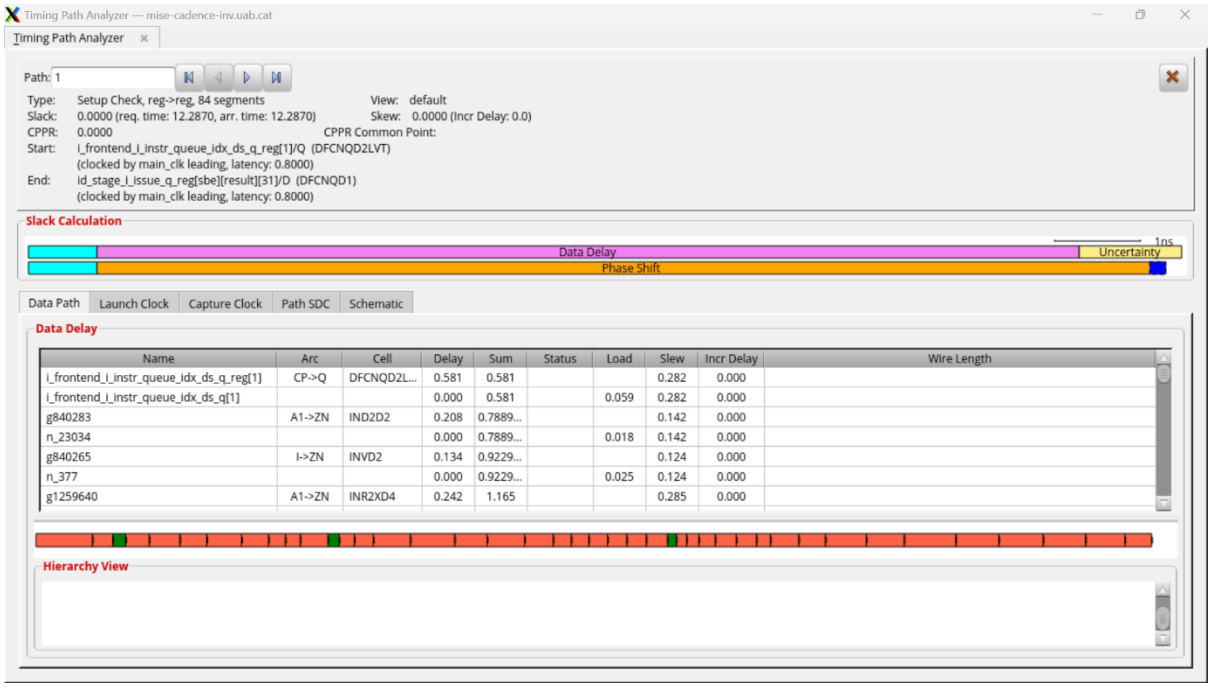


Fig 6. Anàlisi d'un timing path per freqüència de 80MHz amb tecnología de porta LP.

A continuació s'analitza un dels *paths register-to-register*. El temps d'arribada és idèntic al requerit, obtenint un *slack* de 0. Això indica que *netlist* està ben optimitzada per un funcionament a 80MHz. Es realitza una nova síntesi a una freqüència superior.

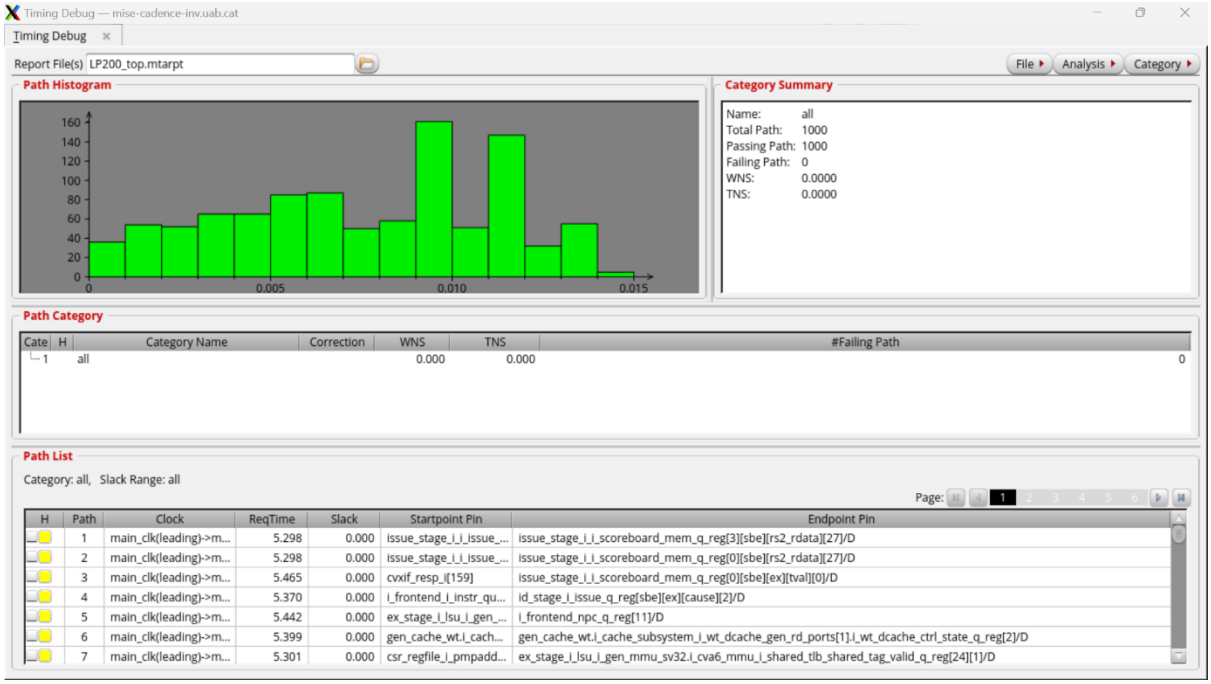


Fig 7. Timing debug per una freqüència de 200MHz amb tecnología de porta LP.

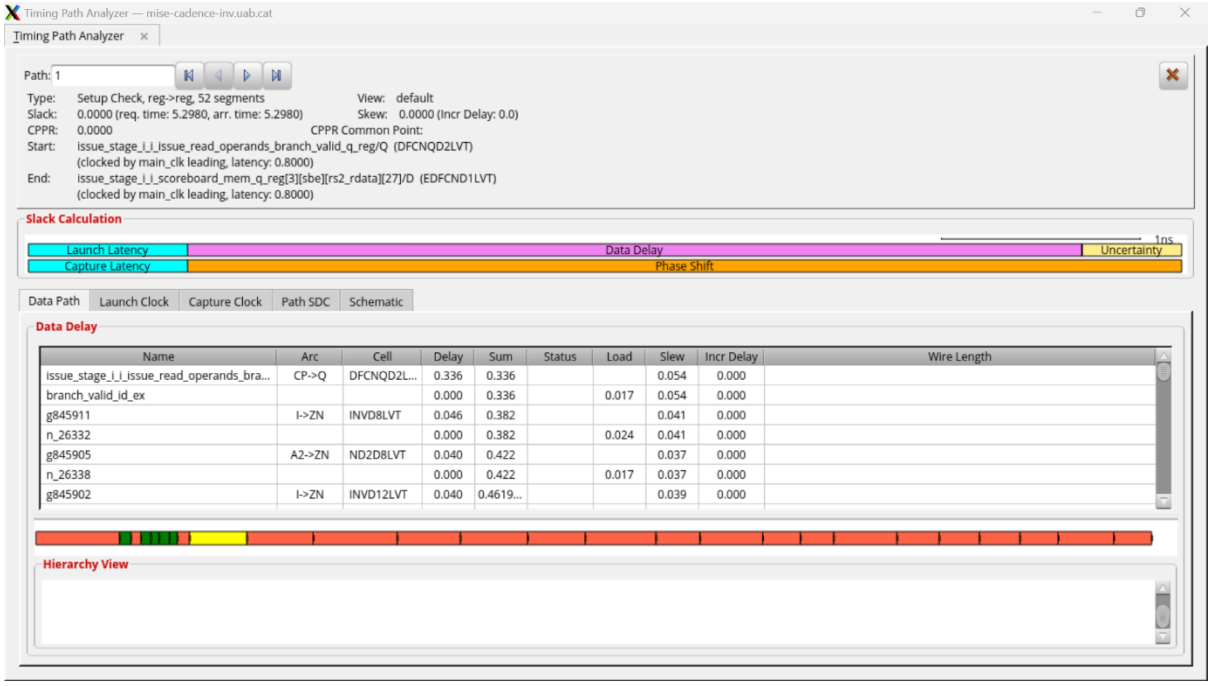


Fig 8. Anàlisi d'un timing path per freqüència de 200MHz amb tecnología de porta LP.

A 200MHz continua tenint un valor de *slack* de 0, de manera que es pot seguir incrementant la freqüència.

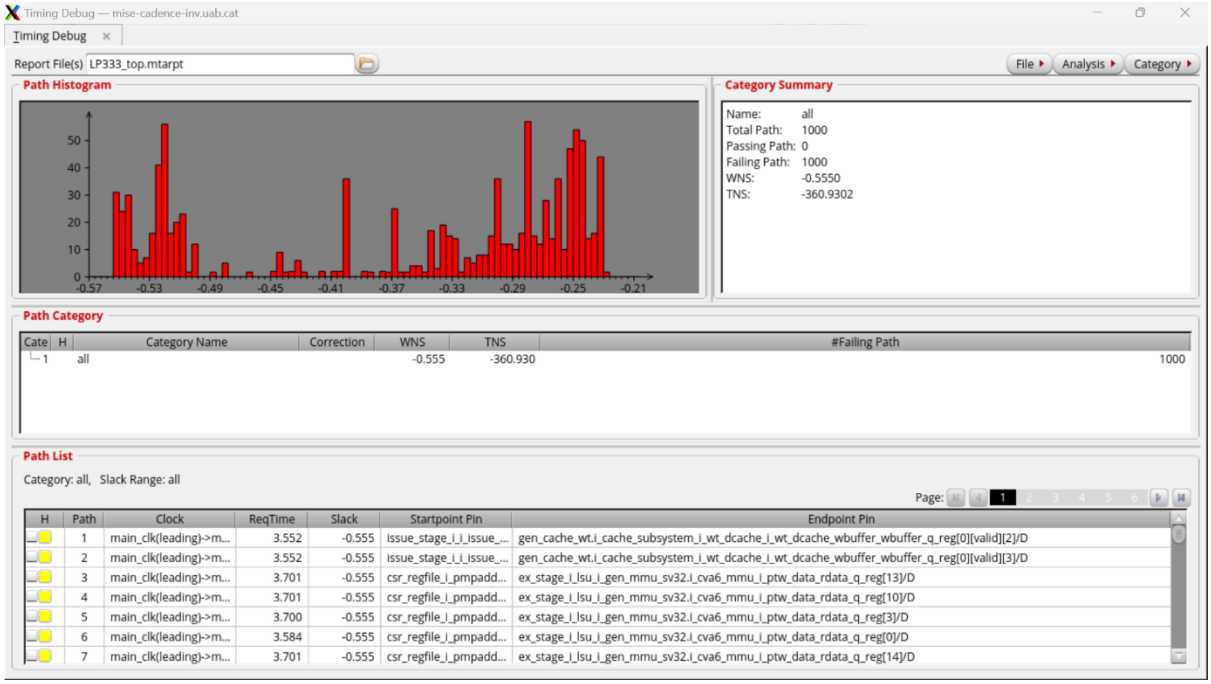


Fig 9. Timing debug per una freqüència de 333MHz amb tecnología de porta LP.

A 333MHz apareixen *timing violations* a tots els *timing paths*. Avaluem un d'ells.

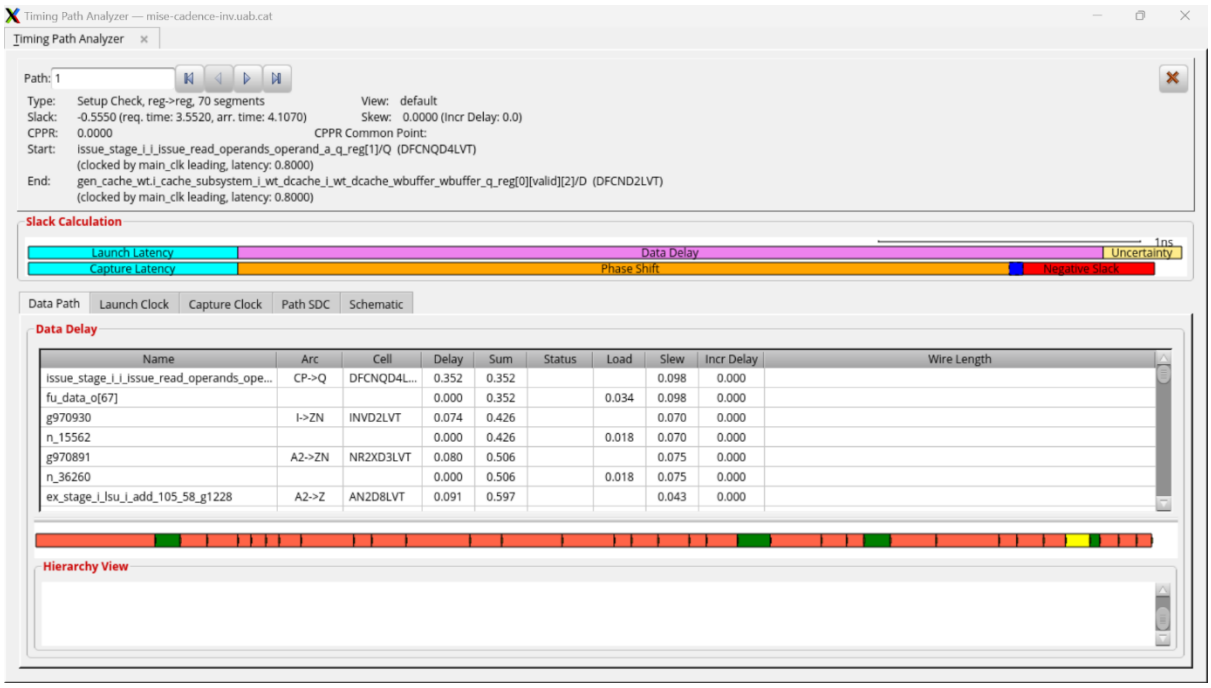


Fig 10. Anàlisi d'un timing path per freqüència de 333MHz amb tecnología de porta LP.

El *timing path* presenta un temps d'arribada (4.1070ns) superior al temps de *setup* requerit (3.5520ns), indicant que el temps de propagació entre portes combinacionals és major que el període del rellotge i generant com a resultat un *slack* negatiu de -0555ns [10][12][13].

Aquest valor de *slack*, apart d'indicar que el *path* no és fiable, és prou gran com per donar a entendre que la freqüència on es comencen a generar violacions temporals és considerablement menor. Així doncs, podem suposar que la freqüència màxima a la que es pot arribar amb la combinació de la tecnologia LP, el disseny i els paràmetres temporals definits al fitxer de *constraints* (apart de la freqüència de rellotge) és d'aproximadament 300MHz.

3.3.3. Anàlisi amb la tecnologia GP

Seguidament, passem a realitzar l'anàlisi de les *netlists* generades a partir de la tecnologia de porta GP. Tal com s'ha mencionat prèviament, les portes de la biblioteca GP són més ràpides que les de la biblioteca LP tot i que presenten un consum més gran. En aquest anàlisi només es té en compte el temps i, per tant, suposem inicialment que els resultats amb aquesta biblioteca seran millors a partir dels 333MHz.

Així doncs, avaluem el *timing* a les mateixes freqüències per fer una comparació més precisa.

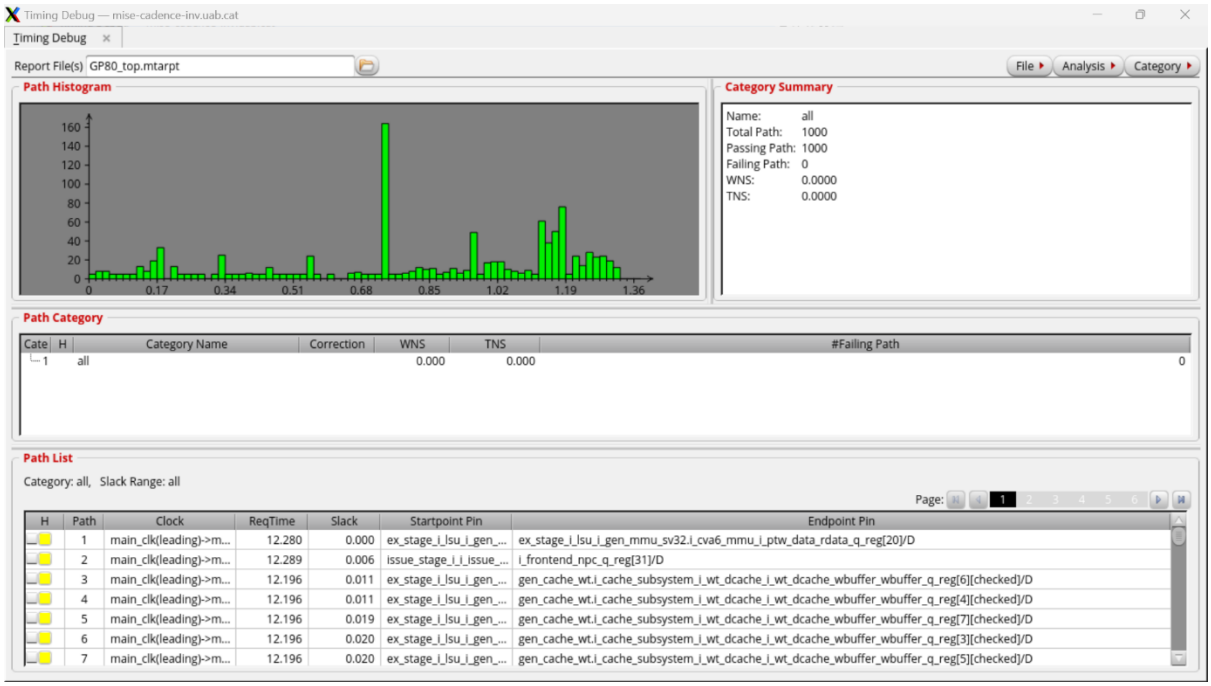


Fig 11. Timing debug per una freqüència de 80MHz amb tecnología de porta GP.

Els *timing paths* a 80MHz són diferents respecte de l'anàlisi amb tecnologia LP però no hi ha *timing violations*.

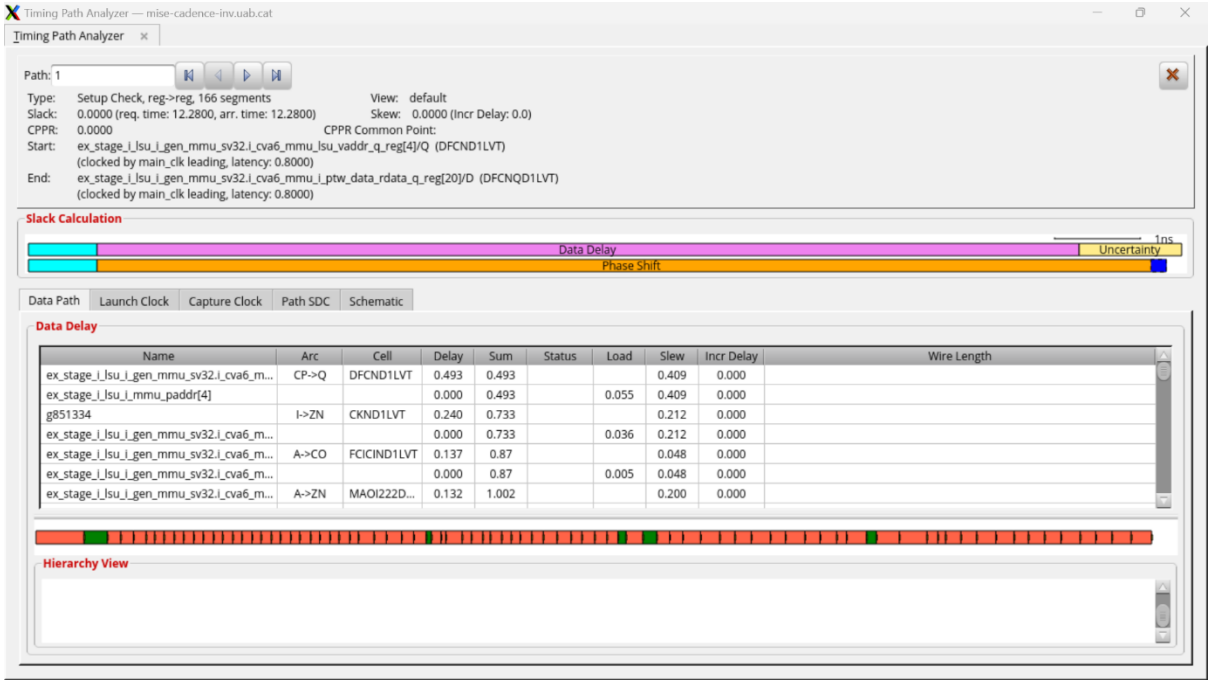


Fig 12. Anàlisi d'un timing path per freqüència de 80MHz amb tecnología de porta GP.

Analitzant un dels *timing paths* es pot observar que el càlcul del *slack* dona zero com a resultat, indicant que el *timing* és òptim.

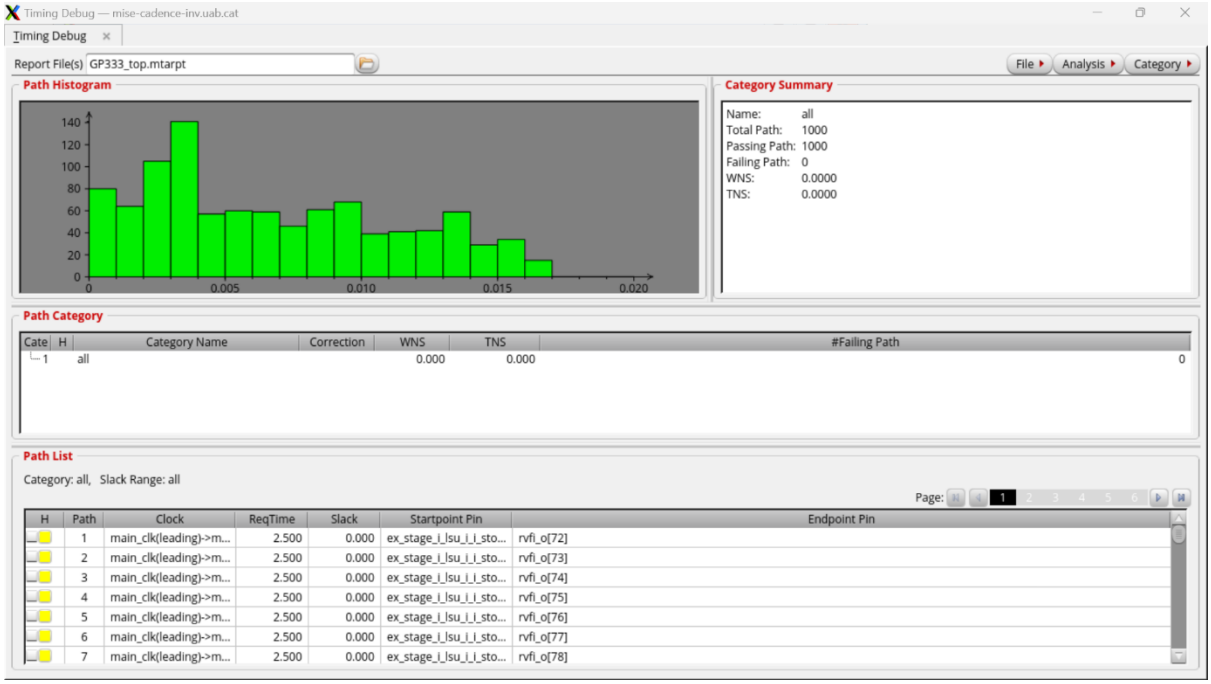


Fig 13. Timing debug per una freqüència de 333MHz amb tecnología de porta GP.

Per el següent anàlisi, incrementem directament fins als 333MHz. A diferència del seu anàleg LP, aquest no presenta *timing violations*.

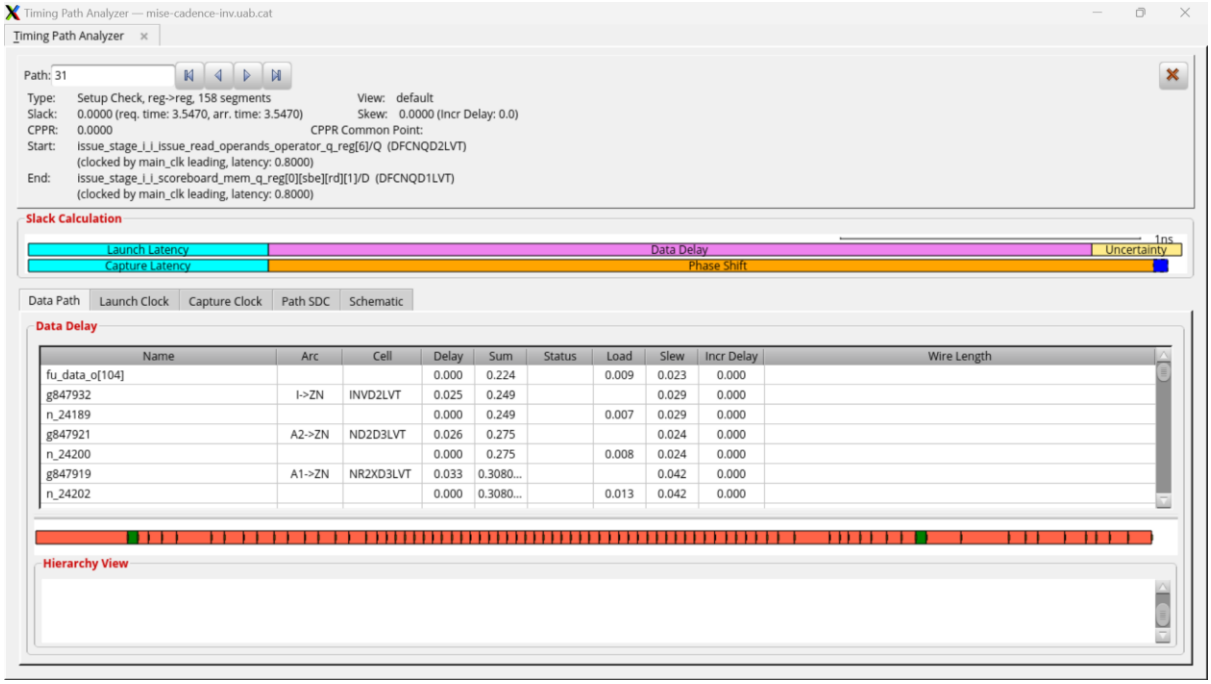


Fig 14. Anàlisi d'un timing path per freqüència de 333MHz amb tecnología de porta GP.

Considerant que la tecnologia de porta GP és més ràpida, obtenir *slack* zero a 333MHz s'ajusta a les nostres prediccions. Per tant, es pot continuar incrementant la freqüència de rellotge.

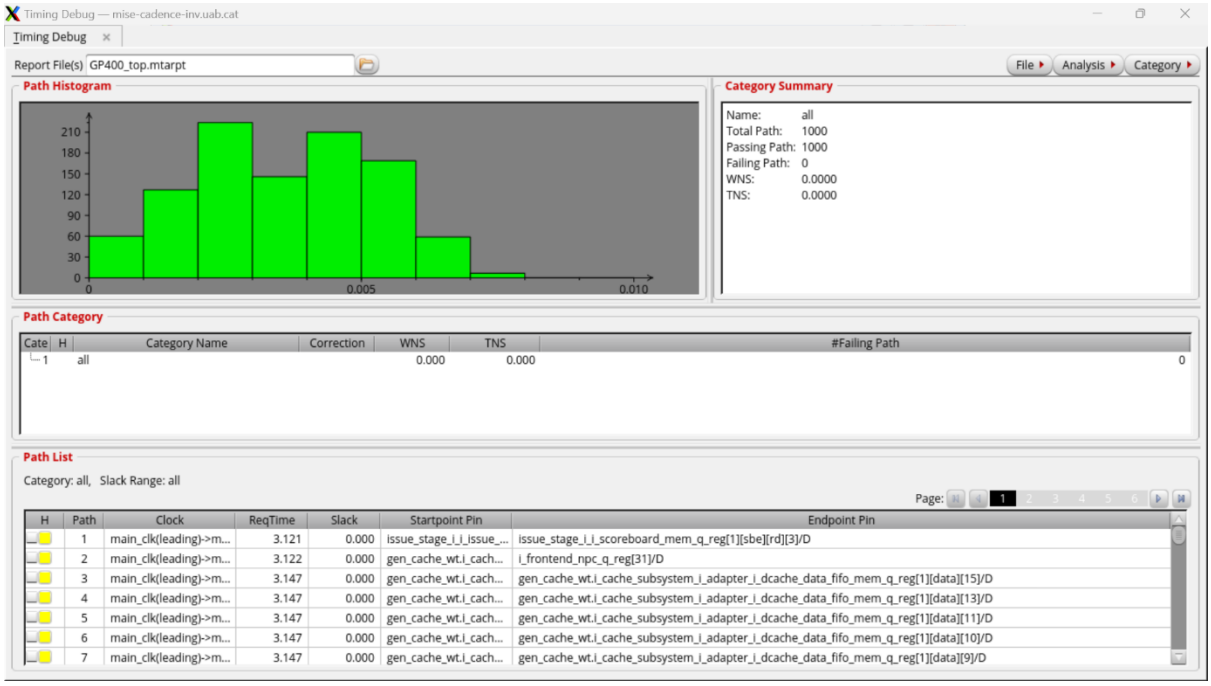


Fig 15. Timing debug per una freqüència de 400MHz amb tecnología de porta GP.

A 400MHz la *netlist* continua funcionant correctament amb uns *timing paths* sense cap violació temporal. A continuació revisem el *slack*.

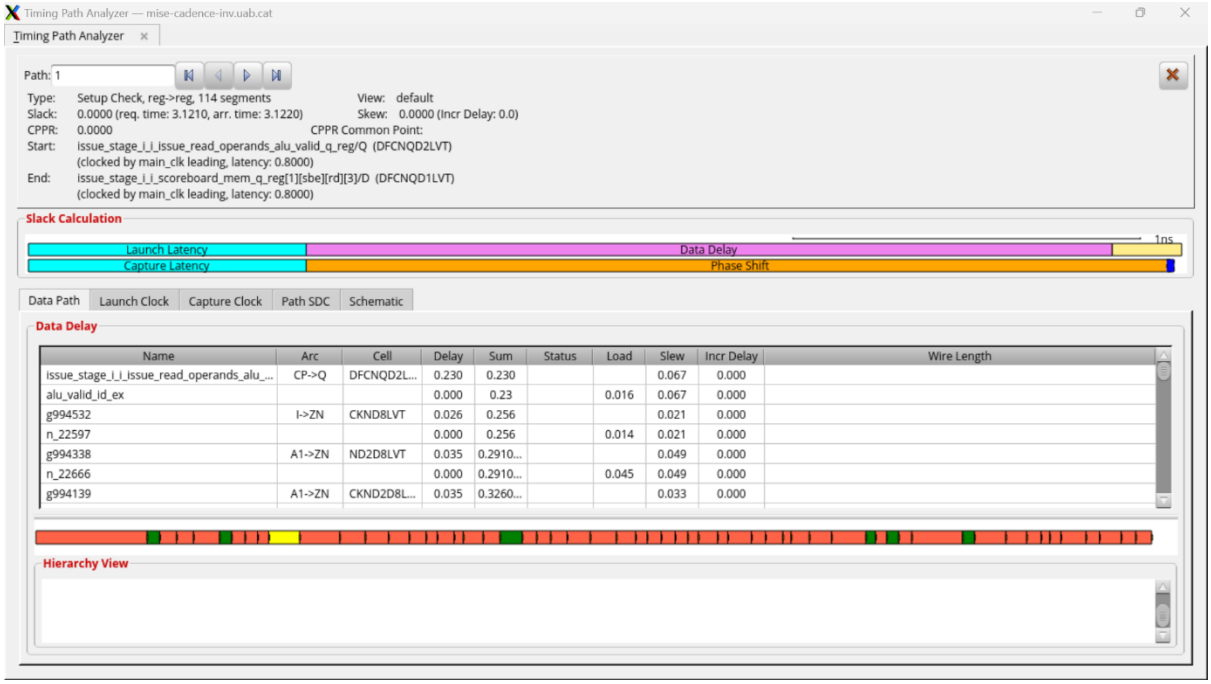


Fig 16. Anàlisi d'un timing path per freqüència de 400MHz amb tecnología de porta GP.

El valor de *slack* present als *timing paths* és de zero. Això és excel·lent perquè verifica que la *netlist* amb aquesta tecnologia de porta pot funcionar a altes velocitats amb bon rendiment temporal. El següent pas és seguir incrementant fins arribar a la freqüència màxima de treball.

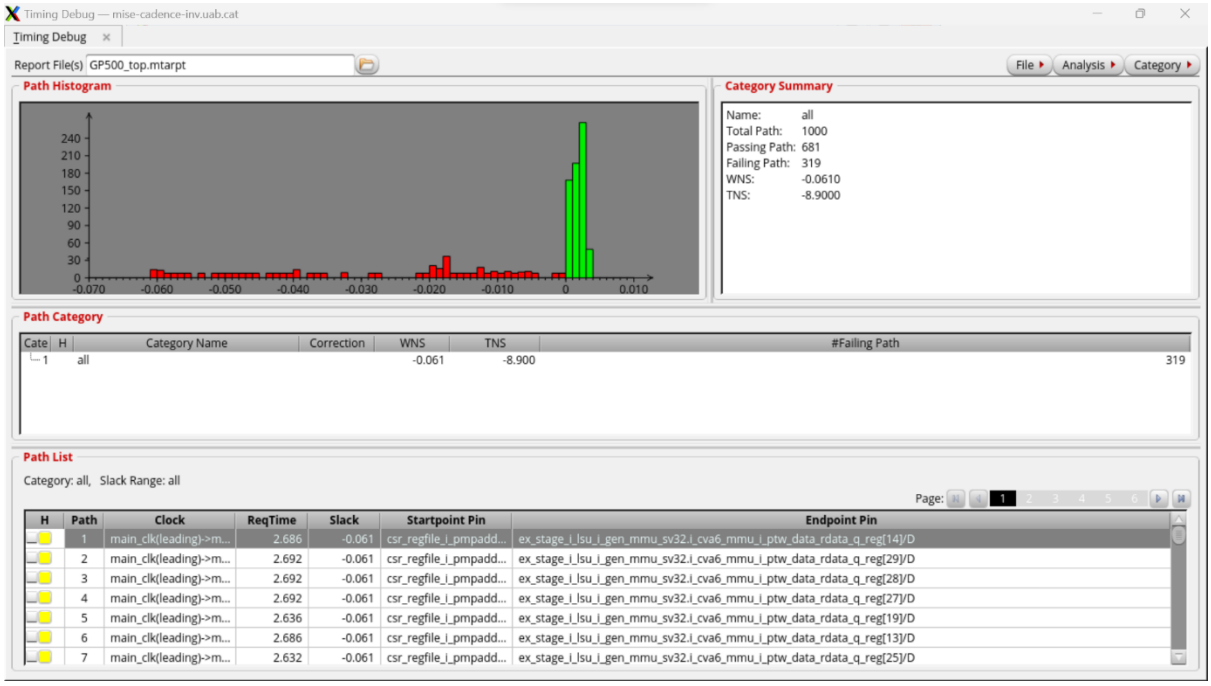


Fig 17. Timing debug per una freqüència de 500MHz amb tecnología de porta GP.

Al incrementar la freqüència fins a 500MHz el rendiment de la *netlist* cau en picat, on només el 68.1% dels *timing paths* analitzats per l'eina són funcionals.

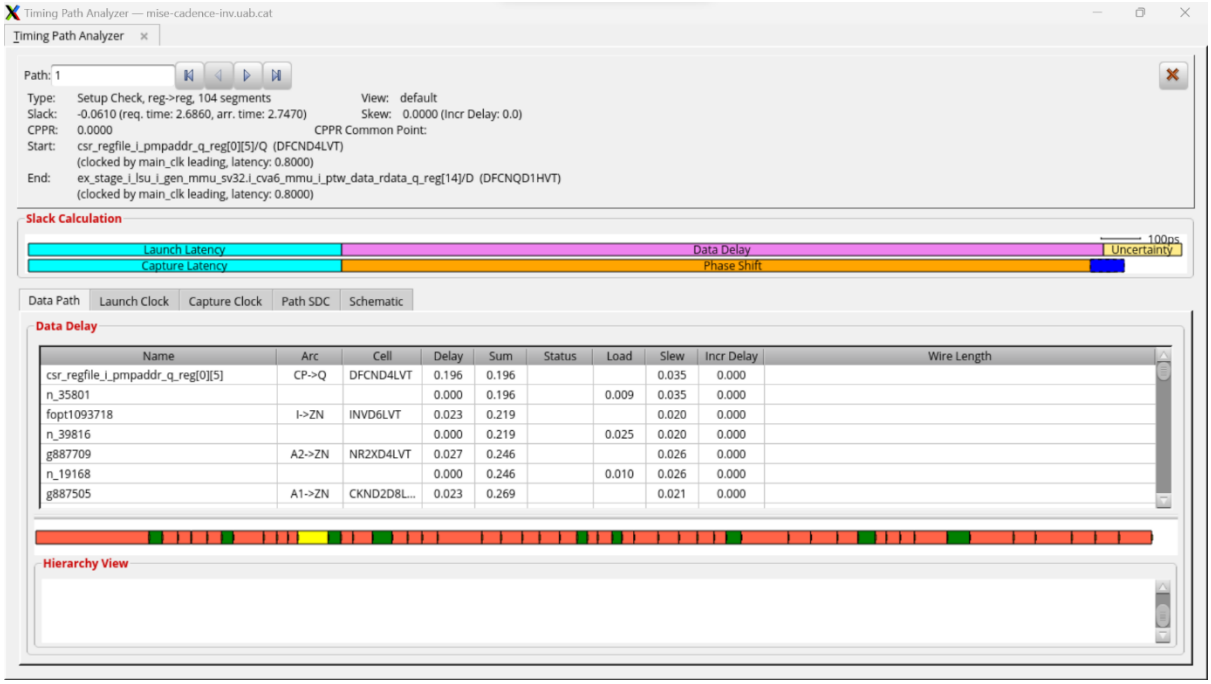


Fig 18. Anàlisi d'un timing path per freqüència de 500MHz amb tecnología de porta GP.

Analitzant un dels *timing paths* fallits s'observa que el temps de propagació o temps d'arribada (2.7470ns) supera el període del rellotge o temps de *setup* requerit (2.6860ns), generant un *slack* negatiu de -0.0610ns. Així com en el cas de 333MHz amb tecnologia LP, aquest alt valor de *slack* indica que la freqüència ha de trobar-se lleugerament per sota de la freqüència utilitzada. La freqüència màxima de treball deu ser d'aproximadament 490MHz [10][12][13].

3.3.4. Conclusions

L'anàlisi s'ha dut a terme amb èxit i obtenint uns resultats lleugerament favorables. Per una banda, s'ha pogut verificar que, fent ús de la tecnologia GP, la *netlist* arriba a freqüències lleugerament altes. Per altra banda, utilitzant la tecnologia LP s'ha comprovat que la *netlist* només funciona correctament a freqüències baixes. Aquests resultats es troben afectats en gran mesura degut a que les memòries utilitzades consisteixen en *flip flops D* i aquestes es troben molt limitades. D'haver pogut fer servir memòries SRAM, la freqüència màxima de treball amb ambdues tecnologies s'hauria vist notablement incrementada.

Per últim, comentar que la intenció inicial a l'hora de realitzar la síntesi lògica del disseny era minimitzar altres aspectes de la *netlist*, com ara el consum. Malauradament, no s'ha disposat de prou temps per continuar optimitzant el disseny.

4. SÍNTESI FÍSICA

4.1. Introducció

Aquest capítol recull el procés de síntesi física del processador CVA6 mitjançant l'ús de l'eina de síntesi Innovus Implementation System. L'objectiu específic per aquesta etapa del projecte és obtenir un *layout* funcional que pugui treballar a la freqüència màxima de treball que es va aconseguir al capítol anterior, és a dir, 400MHz.

4.2. Desenvolupament del procés de síntesi física

Tal com es va mencionar breument al capítol 1, l'eina de síntesi física fa servir com entrades 3 tipus de fitxers [2]:

- La *netlist* generada a partir de la síntesi lògica. Aquest arxiu conté informació de la connectivitat elèctrica del disseny.
- Biblioteques de la tecnologia TSMC 65nm necessàries per dur a terme la implementació física:
    - *Liberty files* per aportar la informació temporal associada a les *standard cells* durant la fase d'interconnexió o *routing*.
    - Fitxers *Layer Exchange Format* (LEF) que contenen informació sobre la descripció física del *layout* [12].
    - Fitxers *captable* i *QRCtechfile* (*Quality RC*) que modelen la resistència i capacitància de les capes de metall [12].
    - Fitxers *constant database* (.cdb) que contenen bases de dades de la tecnologia [12].
- Un fitxer SDC amb els *contraints* optimitzats com a resultat de la síntesi lògica.
- Un fitxer *Multi-Mode Multi-Corner* (MMMC) creat per configurar el STA de forma que, en un mateix anàlisi, doni cobertura a diferents condicions de treball: efectes paràsits d'interconnexió (*RC corners*) i PVT *corners* [15][16].

El flux de síntesi física es divideix en una sèrie de etapes que realitzen funcions complementàries, obtenint com a resultat la implementació física del disseny VLSI. La Figura 19 mostra les diverses etapes [9][12]:

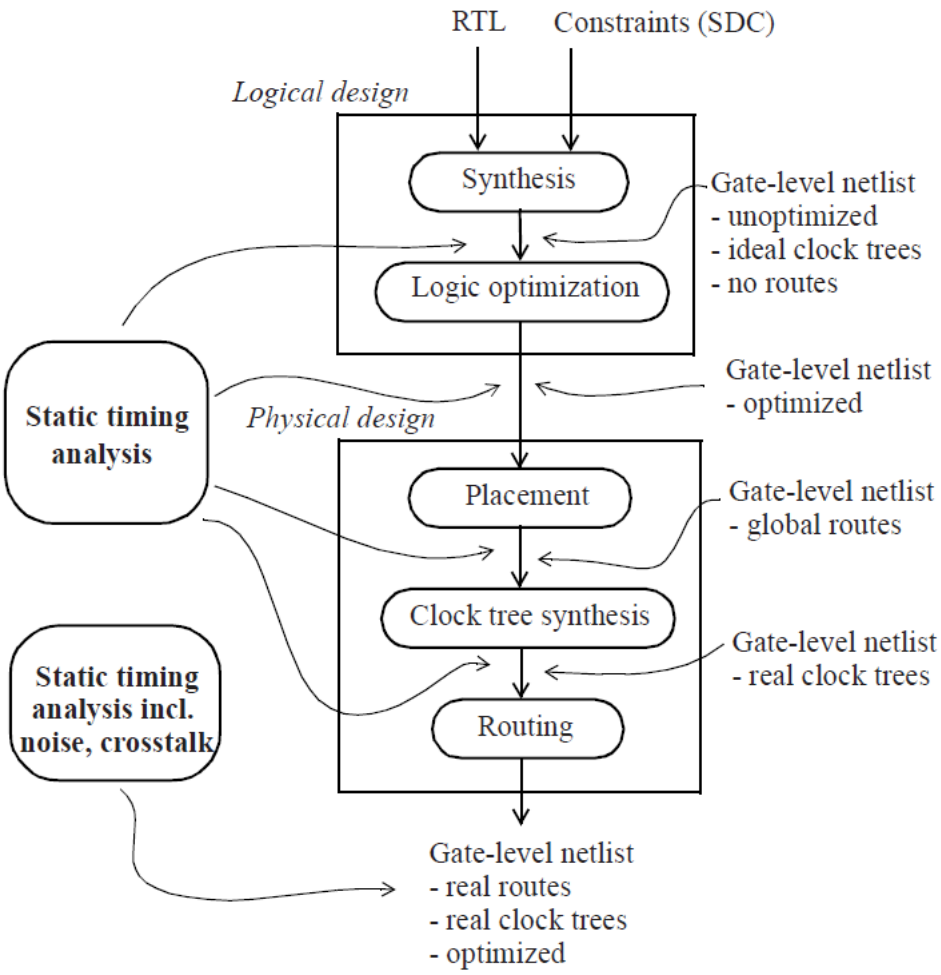


Fig 19. Diagrama de flux dels processos de síntesi lògica i física.
J. Bhasker and R. Chadha, "Figure 1-2 CMOS digital flow," in Static Timing Analysis for Nanometer Designs, Springer Science & Business Media, 2009.

- Setup synthesis environament/ read libraries/ read constraints: Es tracta d'una etapa prèvia a la síntesi que realitza la mateixes funcions que les primeres etapes del flux de síntesi lògica mostrat al capítol 3. S'indica a l'eina on es troben els directoris de les biblioteques de la tecnologia i dels fitxers SDC i MMMC per llegir-los i assigna un directori per els fitxers generats per l'eina un cop finalitza el procés de síntesi [12].

- Placement: Realitza les funcions de crear el *floorplan*, és a dir, la base sobre la que es modela el *layout*, a més d'inserir els elements d'alimentació (anells, línies i interconnexió) i, per últim, inserir la *netlist* mitjançant *standard cells* [12][17].
- Clock Tree Synthesis (CTS): És de les més importants del procés, ja que en aquesta es genera el *clock tree* que connecta tots els rellotges i registres a més de gestionar els temps del disseny tal com es va explicar a la secció 3.3.1.3 [12].
- Routing: . Es coneix com a *routing* al procés d'interconnexió dels diferents elements físics del disseny mitjançant diferents capes de metall amb diferents paràmetres, com ara la resistència elèctrica. Aquesta interconnexió es realitza en diverses fases [12]:
    - La fase de *global routing*, on l'eina de síntesi interconnecta els blocs més simples, com ara les *standard cells*. per mitjà d'un algoritme.
    - La fase de *detailed routing*, dins la qual es realitza la interconnexió de blocs RF de forma més acurada d'acord a les necessitats específiques del bloc en qüestió.

  El resultat és una xarxa interconnectada amb múltiples capes de metall, unes sobre altres, connectades en vertical per cables a través de vies [12]. El motiu d'aquesta interconnexió vertical és per evitar la congestió física del *layout*. Tot i així, és comú trobar desviacions elèctriques produïdes per efectes paràsits de capes de metall que es troben massa properes. Aquest fenomen es coneix com *crosstalk* i es tractarà a la secció 4.2.7.

Addicionalment, l'eina de síntesi realitza un conjunt d'anàlisis temporals estàtics entre les diferents etapes per detectar i corregir possibles *timing violations*. Un cop finalitza l'etapa de routing, l'eina de síntesi duu a terme una última STA per detectar i corregir problemes temporals o relacionats amb la presència de soroll o *crosstalk* [12][18].

L'execució del procés de síntesi física es pot dur a terme de dues formes: fent servir la interfície gràfica Innovus Stylus o per mitjà de scripts amb l'objectiu de realitzar *batch execution*. Inicialment es va considerar que l'alumne fes servir la interfície gràfica per familiaritzar-se amb el software degut a la naturalesa formativa del projecte. Malauradament, eventualment es va deixar de banda aquest mètode degut a que la complexitat de la síntesi física posava en risc que l'alumne pogués finalitzar el

projecte dins del termini previst. Així doncs, la síntesi s'ha realitzat per *batch execution* sota la supervisió del director del projecte.

De forma similar al capítol 3, els scripts de síntesi es troben estructurats de manera que hi ha un script de control (*run_all.tcl*) encarregat de dur a terme la *batch execution* i 5 scripts que contenen el conjunt d'instruccions per executar les etapes del procés de síntesi física. Addicionalment, s'ha inclòs un fitxer (*variables.tcl*) encarregat de carregar les variables d'entorn i paràmetres que serveixen de suport per alguns dels scripts d'instruccions. La Figura 20 mostra l'estructura de la *bash execution* que duu a terme el script run_all.

```
1 source $env(DESIGN_PATH)/common/variables.tcl          ;
2 source $env(DESIGN_PATH)/pnr/scripts/0_init_design.tcl  ;# Imports synthesized design and tech data.
3 source $env(DESIGN_PATH)/pnr/scripts/1_floorplan.tcl    ;# Floorplan
4 source $env(DESIGN_PATH)/pnr/scripts/2_place.tcl        ;# Places the design.
5 source $env(DESIGN_PATH)/pnr/scripts/3_cts.tcl          ;# Performs Clock Tree Synthesis.
6 #source $env(DESIGN_PATH)/pnr/scripts/4_route.tcl       ;# Routes the design, performs DRC and signoff checks, generates output data and export the design.
7
```

Fig 20. Contingut del script run_all.tcl.

Els scripts de síntesi es troben annexats al final del document (Annex 2).

A continuació es descriu com s'ha desenvolupat la tasca de dur a terme el procés de síntesi física. S'ha escollit fer servir la tecnologia de porta GP a una freqüència de treball baixa (100MHz) ja que aquestes seran unes condicions de treball òptimes per poder realitzar una primera síntesi per verificar que els scripts utilitzats funcionen correctament. Una vegada es finalitzi amb èxit la primera síntesi, es realitzarà una segona síntesi fent servir la màxima de treball assolida durant el procés del capítol 3 (400MHz) per verificar que es pot obtenir un *layout* funcional amb aquestes condicions de treball. Durant el procés es farà servir la interfície gràfica Innovus Stylus com a suport visual per mostrar l'evolució del *layout* a mesura que s'avança en el flux de síntesi.

4.2.1. Inicialització

L'etapa de de Setup environament/ read libraries/ read constraints es duu a terme entre dos scripts: el script *variables.tcl*, que s'encarrega de llegir els *paths* dels directoris (*paths.tcl*) i els paràmetres de diversos elements del disseny (*config.tcl*) per poder tenir accés a aquestes dades durant totes etapes del flux, i el script *0_init_design.tcl*, que configura els paràmetres globals de l'eina de síntesi per adequar-los al disseny que es vol sintetitzar. En aquesta etapa també es realitzen les connexions a PWR i GND.

Així mateix, el script 0_init_design.tcl també és l'encarregat d'assignar els directoris de sortida i de realitzar la importació de diversos fitxers que serveixen com a input per al flux de síntesi física: la *netlist* del disseny, el fitxer MMMC i els fitxers LEF de la tecnologia. Tot i que és possible llegir el fitxer SDC en aquesta etapa, s'ha pres la decisió de desplaçar aquesta acció a l'etapa de CTS ja que és on comencen a tenir impacte els *constraints*. D'aquesta manera, en cas que sigui necessari realitzar canvis en el fitxer SDC, no s'hauran de refer les etapes prèvies.

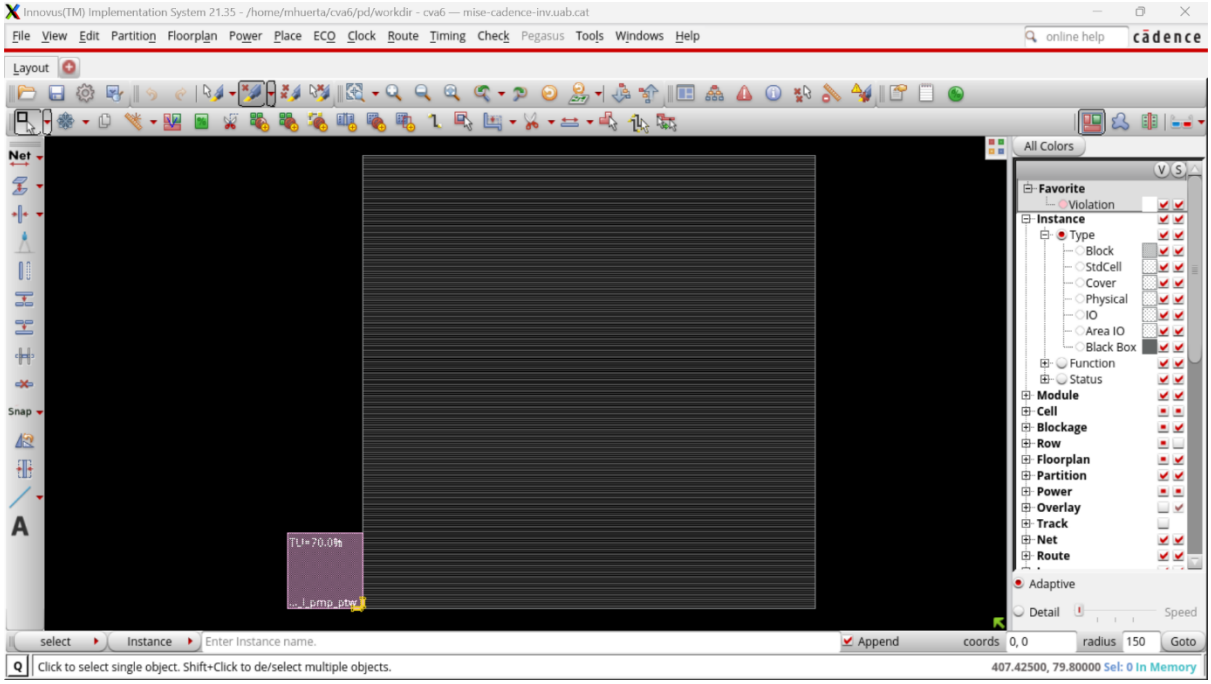


Fig 21. Captura de la graella del layout.

4.2.2. Floorplan

L'etapa de Placement s'ha separat en dos scripts per poder executar el *floorplan* i el *placement* de forma separada en cas que convingui realitzar canvis en només una de les parts.

Durant el procés de *floorplanning*, el script *1_floorplan.tcl* fa servir els paràmetres tant del propi *floorplan* com dels elements d'alimentació que es troben disponibles a *variables.tcl*. Aquests paràmetres s'ha configurat de forma tenen els següents valors per defecte:

- La matriu del *floorplan* presenta una amplada, alçada i *core size* de 500nm, 550nm i 30nm respectivament.
- Els anells d'alimentació (*power rings*) presenten capes horitzontals (superior i inferior) inserides en la capa M9 i capes verticals (dreta i esquerra) inserides en la capa M8. Tant les

capes horitzontals com les verticals presenten una amplada de 10nm i es col·loquen a una distància de 2nm respecte dels límits de la matriu del *floorplan*.

- Les línies d'alimentació (*power stripes*) es troben situades a la capa M6 en posició vertical i amb una distància 90nm entre línies. Presenten una amplada de 10nm.
- La interconnexió de les estructures d'alimentació (*power routing*) es realitza en les capes M6-M9, on la capa de metall M9 és la més elevada de la tecnologia TSMC utilitzada.

Considerant la importància que el *floorplan* té en posteriors etapes (en termes de consum, congestió, densitats de corrent, caigudes IR...) és probable que sigui necessari modificar les dimensions dels seus elements a posteriori [17].

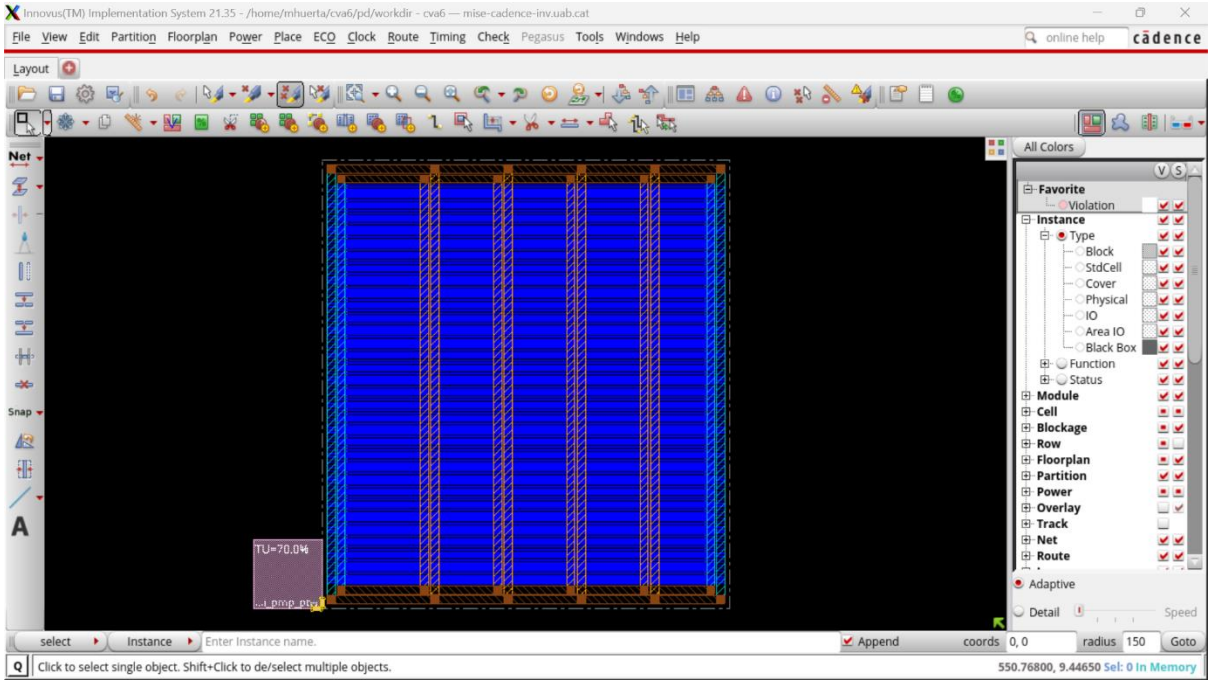


Fig 22. Captura del layout amb el floorplan i els elements d'alimentació.

4.2.3. Placement

El script *2_place.tcl* implementa la netlist amb standard cells i hi afegeix una sèrie de cel·les que milloren les característiques del circuit [12]:

- W*ell taps*: Són cel·les utilitzades per reforçar els nivells de VDD i VSS en el *bulk* dels transistors MOS i en els pous N amb l'objectiu d'evitar que pugui aparèixer *latch-up*, creant camins de baixa impedància entre VDD i VSS [19][20].
- Cel·les *Tie*: Protegeixen les portes dels transistors contra efectes de *Power* o *Ground bounce* que, en circuits d'alta velocitat amb tecnologia sub-micromètrica, poden provocar *false switching*. Les cel·les *TieHi* connecten a VDD les cel·les que ho requereixin mentre que les *TieLo* fan el mateix per VSS [19].

Per últim, es duu a terme un STA previ al *Clock Tree Synthesis* (*pre-CTS*) on es fa un anàlisi temporal i de consum juntament amb un *Design Rule Check* (DRC) per detectar i corregir errors abans de començar a generar l'estructura *clock tree* [12].

Precisament, ha sorgit un problema durant l'execució del script *2_place.tcl*:

```
*ERROR: (IMPSP-190):      Design has util 100.5% > 100%, placer cannot proceed.
The problem may be caused by density screens (softBlockages and partial blockages)
or user-specified cell padding. Please correct this problem first.
```

L'àrea del *floorplan* no era prou gran com per poder inserir el circuit, de forma que s'ha incrementat l'amplada i alçada del de la matriu fins a 550nm, 600nm respectivament.

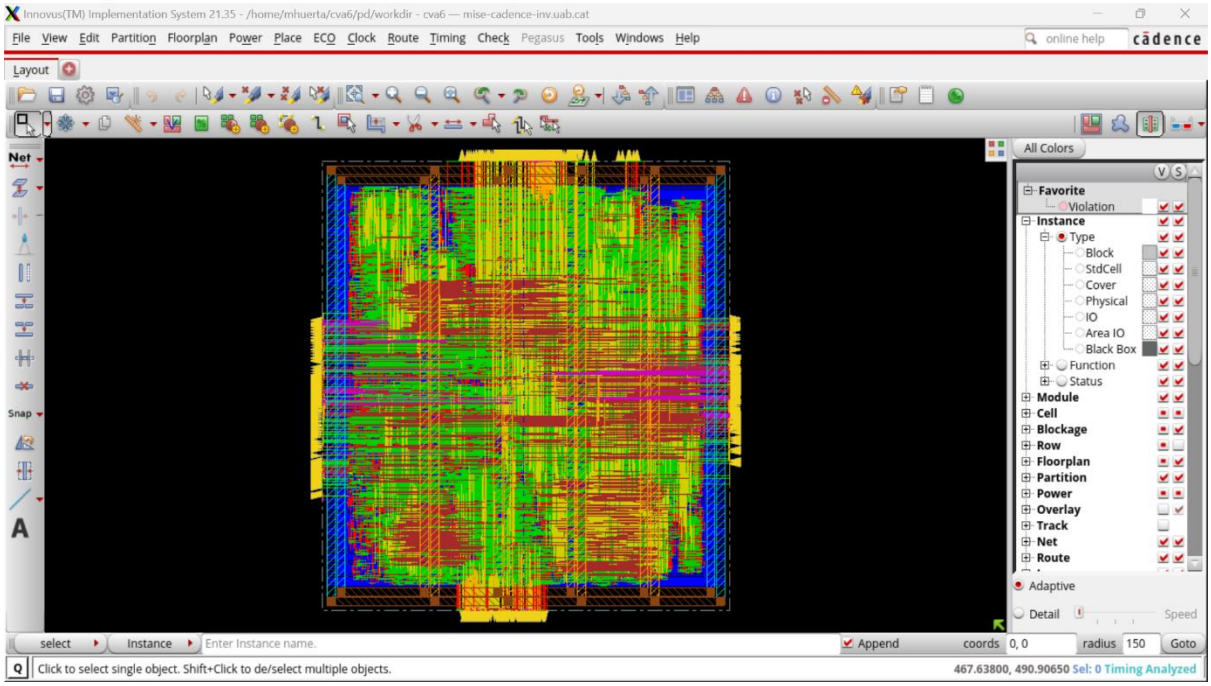


Fig 23. Captura del layout amb el placement.

4.2.4. Clock Tree Synthesis (CTS)

A diferència de la gestió dels temps que es va fer servir durant la síntesi lògica del capítol 3, aquest *clock tree* no és simulat i, per tant, hi ha paràmetres temporals que, en comptes de estar definits en el fitxer de *constraints*, es generen com a resultat de la creació del *clock tree*. Això implica que, un cop generada l'estructura *clock tree*, serà precís revisar que els paràmetres temporals tinguin uns valors raonables per no perjudicar el rendiment del *layout.*

El script *3_cts.tcl* s'encarrega de fer els preparatius previs a la generació de l'estructura *clock tree*. Entre aquests preparatius es troba llegir el fitxer SDC per accedir als *constraints* temporals, configurar altres paràmetres temporals com el factor de *timing derate* i optimitzar la configuració tant de les eines d'anàlisi temporal com de l'eina d'anàlisi de l'estructura *clock tree* anomenada CCOpt i que s'explicarà més en detall durant l'anàlisi del *layout* a la secció 4.3. Addicionalment, el script també configura l'algorisme que gestiona la lògica dins del *clock tree* per fer servir buffers i inversors de les biblioteques de la tecnologia al moment de crear les cadenes d'inversors dels *timing paths*. S'ha decidit prioritzar l'ús d'inversors per sobre de buffers ja que els inversors generalment presenten menor consum i menor retard d'inserció respecte dels buffers.

Posteriorment es realitza la interconnexió del *clock tree* entre les capes M2-M5, on s'hi *inclouen Non-Default Rules* (NDR) creades per inserir els elements de tipus *leaf*, *trunk* i *top* en capes específiques (*leaf* en M2 i M3; *trunk* en M3 i M4; *top* en M4 i M5). Seguidament, es crea una xarxa de *clock trees* amb els respectius *skew groups* i s'executa el *clock tree generation*.

Per últim, s'optimitzen els temps de *hold* i es realitza un STA post-CTS per detectar possibles errors en el *timing* o en la interconnexió de l'estructura *clock tree*.

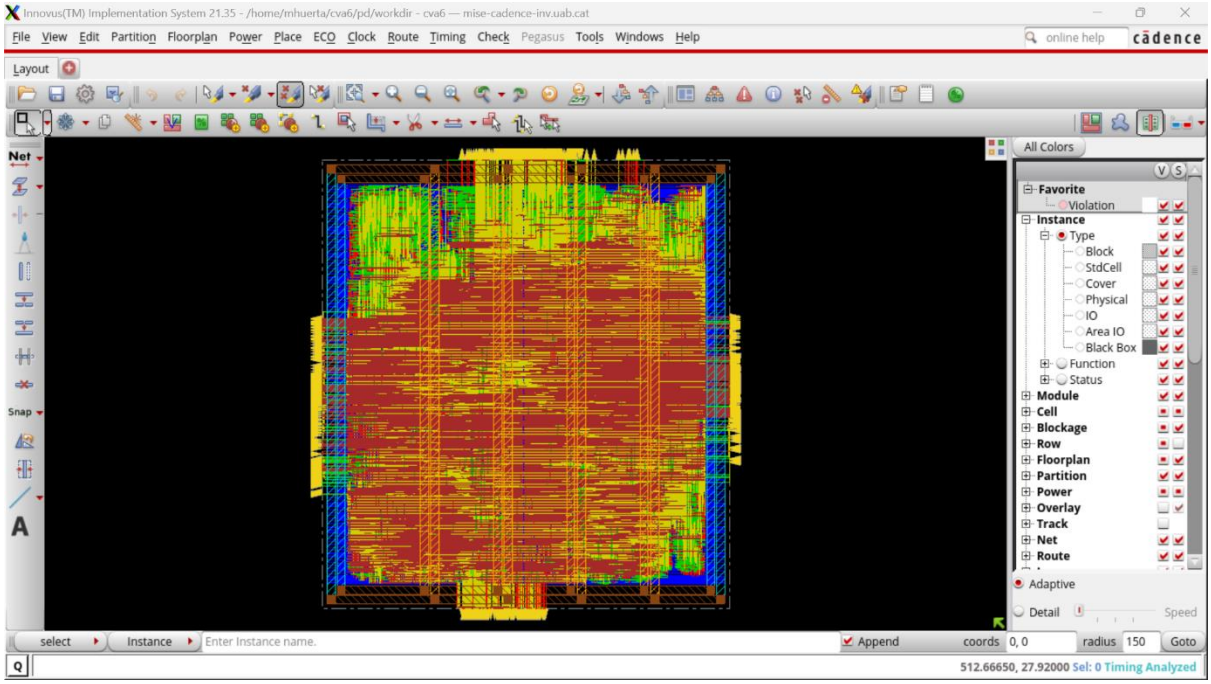


Fig 24. Captura del layout amb la Clock Tree Synthesis.

L'anàlisi post-CTS, mostrat a les Figures 23 i 24, indica que l'etapa s'ha dut a terme amb èxit sense *setup* ni *hold timing violations*.

```
** Profile ** Overall slacks :  cpu=0:00:08.5, mem=8858.5M
Generating machine readable timing report  /home/mhuerta/cva6/pd/workdir/pnr_report//timing/postCTS.mtarpt.gz
** Profile ** Total reports :  cpu=0:00:45.5, mem=8790.7M
** Profile ** DRVs :  cpu=0:00:04.5, mem=8803.9M

------------------------------------------------------------------
         time_design Summary
------------------------------------------------------------------

Setup views included:
 av_normal_typ av_normal_max

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.202  |  0.202  |  0.911  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|  19351  |  10534  |  12846  |
+--------------------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +-----------------+-------------+------------------|
|                |  Nr nets(terms) | Worst Vio   |  Nr nets(terms)  |
+----------------+-----------------+-------------+------------------+
|   max_cap      |      1 (1)      |   -0.006    |      1 (1)       |
|   max_tran     |      1 (2)      |   -0.422    |      1 (2)       |
|   max_fanout   |      0 (0)      |     0       |      0 (0)       |
|   max_length   |      0 (0)      |     0       |      0 (0)       |
+----------------+-----------------+-------------+------------------+

Density: 89.497%
Routing Overflow: 0.00% H and 0.07% V
------------------------------------------------------------------
** Profile ** Report data :  cpu=0:00:00.5, mem=8805.4M
Reported timing to dir /home/mhuerta/cva6/pd/workdir/pnr_report//timing
Total CPU time: 62.7 sec
Total Real time: 13.0 sec
Total Memory Usage: 8613.402344 Mbytes
*** time_design #2 [finish] : cpu/real = 0:01:02.7/0:00:13.0 (4.8), totSession cpu/real = 1:16:46.5/0:35:42.5 (2.2), mem = 8613.4M
```

Fig 25. Captura de l'anàlisi temporal post-CTS per el temps de setup de la síntesi a 100MHz.

```
35306 Start delay calculation (fullDC) (8 T). (MEM=8545.06)
35307 Total number of fetched objects 54019
35308 AAE_INFO: Total number of nets for which stage creation was skipped for all views 0
35309 Total number of fetched objects 54019
35310 AAE_INFO: Total number of nets for which stage creation was skipped for all views 0
35311 End delay calculation. (MEM=8899.2 CPU=0:00:39.7 REAL=0:00:05.0)
35312 End delay calculation (fullDC). (MEM=8899.2 CPU=0:00:46.9 REAL=0:00:07.0)
35313 *** Done Building Timing Graph (cpu=0:00:58.2 real=0:00:10.0 totSessionCpu=1:17:52 mem=9072.2M)
35314 ** Profile ** Overall slacks :  cpu=0:01:00, mem=9080.2M
35315 Generating machine readable timing report  /home/mhuerta/cva6/pd/workdir/pnr_report//timing/postCTS_hold.mtarpt.gz
35316 ** Profile ** Total reports :  cpu=0:00:39.2, mem=8888.4M
35317
35318 ------------------------------------------------------------------
35319          time_design Summary
35320 ------------------------------------------------------------------
35321
35322 Hold views included:
35323  av_normal_min_lt av_normal_max
35324
35325 +--------------------+---------+---------+---------+
35326 |     Hold mode      |   all   | reg2reg | default |
35327 +--------------------+---------+---------+---------+
35328 |           WNS (ns):|  0.159  |  0.159  |  0.212  |
35329 |           TNS (ns):|  0.000  |  0.000  |  0.000  |
35330 |    Violating Paths:|    0    |    0    |    0    |
35331 |          All Paths:|  19351  |  10534  |  12846  |
35332 +--------------------+---------+---------+---------+
35333
35334 Density: 89.497%
35335 Routing Overflow: 0.00% H and 0.07% V
35336 ------------------------------------------------------------------
35337 ** Profile ** Report data :  cpu=0:00:00.5, mem=8935.4M
35338 *** Enable all active views. ***
35339 Reported timing to dir /home/mhuerta/cva6/pd/workdir/pnr_report//timing
35340 Total CPU time: 108.12 sec
35341 Total Real time: 25.0 sec
35342 Total Memory Usage: 8603.816406 Mbytes
35343 *** time_design #3 [finish] : cpu/real = 0:01:48.1/0:00:24.8 (4.4), totSession cpu/real = 1:18:34.7/0:36:07.4 (2.2), mem = 8603.8M
35344 @file 209:
35345 @file 210: ### Report on clock trees to check area and other statistics
35346 @@file 211: report_clock_trees -out_file ${PNR_REPORT}/ClockTreeReports/clock_trees.rpt
```

Fig 26. Captura de l'anàlisi temporal post-CTS per el temps de hold de la síntesi a 100MHz.

4.2.5. Routing

El script *4_route.tcl* és el script més complex de tots els presentats ja que, a part de dur a terme el *routing*, també realitza una sèrie de passos addicionals d'optimització per millorar la qualitat del *layout* resultant. Per aquest motiu, les explicacions de les funcions que el script realitza es divideixen en diverses seccions (4.2.5 – 4.2.10).

La part corresponent a la interconnexió del script *4_route.tcl* realitza el *routing* global i detallat del disseny i, posteriorment, analitza la congestió del *layout*. La congestió acostuma a succeir a les capes més baixes degut a que la majoria de cel·les són inserides a les capes M1 i M2, mentre que les capes més altes es reserven per estructures d'alimentació de forma que es pugui construir malles d'alimentació formades per estructures d'alimentació verticals i horitzontals situades a capes de metall diferents [12].

Si la congestió del disseny supera el 90%, és necessari reajustar les dimensions del *floorplan* de forma que hi hagi més espai per realitzar la interconnexió. Precisament, això ha succeït en el primer intent de síntesi física. A les Figures 25 i 26 ja es pot veure que el nivell de densitat de metall comença a ser una mica gran tenint en compte que encara no s'ha arribat a l'etapa de *routing*. La Figura 27 mostra l'anàlisi *post-route* d'aquesta síntesi, on el percentatge de densitat de metall supera el 90% [12].

```
46489 ** Profile ** Total reports :  cpu=0:00:00.9, mem=13983.6M
46490 ** Profile ** DRVs :  cpu=0:00:08.6, mem=13996.9M
46491
46492 ------------------------------------------------------------------
46493      opt_design Final SI Timing Summary
46494 ------------------------------------------------------------------
46495
46496 Setup views included:
46497  av_normal_typ av_normal_max av_normal_min av_normal_min_lt
46498
46499 +--------------------+---------+---------+---------+
46500 |     Setup mode     |   all   | reg2reg | default |
46501 +--------------------+---------+---------+---------+
46502 |           WNS (ns):|  0.121  |  0.121  |  0.225  |
46503 |           TNS (ns):|  0.000  |  0.000  |  0.000  |
46504 |    Violating Paths:|    0    |    0    |    0    |
46505 |          All Paths:|  19351  |  10534  |  12846  |
46506 +--------------------+---------+---------+---------+
46507
46508 +----------------+-------------------+------------+------------------+
46509 |                |              Real              |       Total      |
46510 |    DRVs        +-------------------+------------+------------------+
46511 |                |  Nr nets(terms)   | Worst Vio  |  Nr nets(terms)  |
46512 +----------------+-------------------+------------+------------------+
46513 |   max_cap      |      1 (1)        |   -0.002   |      1 (1)       |
46514 |   max_tran     |     12 (29)       |   -0.149   |     12 (29)      |
46515 |   max_fanout   |      0 (0)        |     0      |      0 (0)       |
46516 |   max_length   |      0 (0)        |     0      |      0 (0)       |
46517 +----------------+-------------------+------------+------------------+
46518
46519 Density: 90.110%
46520 Total number of glitch violations: 1
46521 ------------------------------------------------------------------
46522 ** Profile ** Report data :  cpu=0:00:00.5, mem=13998.4M
46523 **opt_design ... cpu = 0:20:49, real = 0:04:19, mem = 10403.3M, totSessionCpu=5:02:01 **
46524  ReSet Options after AAE Based Opt flow
46525 *** Finished opt_design ***
46526 cleaningup cpe interface
46527       flow.cputime  flow.realtime  timing.setup.tns  timing.setup.wns  snapshot
46528 UM:*                                      0.000 ns          0.121 ns  final
46529 UM: Running design category ...
```

Fig 27. Captura de l'anàlisi temporal post-route del temps de setup de la síntesi a 100MHz.

Per solucionar-ho, ha sigut necessari augmentar novament les dimensions de la matriu del *floorplan*, de forma que ara la matriu presenta una l'amplada i alçada de 600nm i 650nm respectivament.

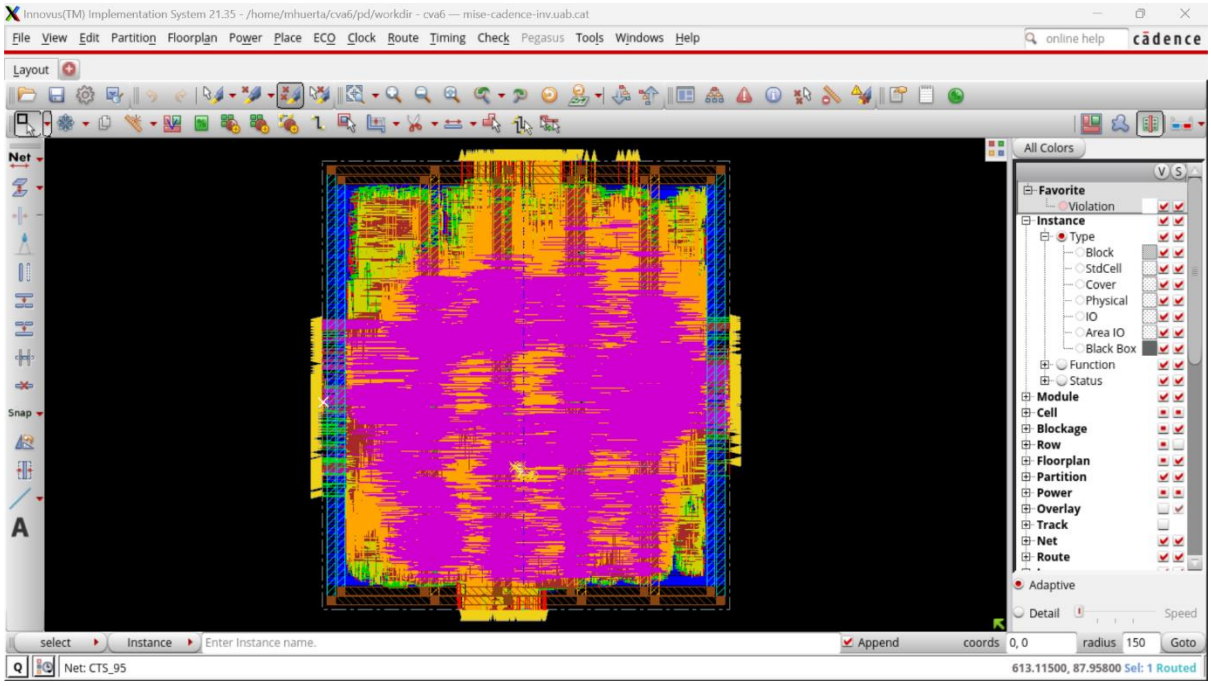


Fig 28. Captura del layout amb el global routing.

4.2.6. Optimització

Tal com es va explicar al llarg del capítol 3, la optimització del temps és la principal prioritat a l'hora d'optimitzar aquest disseny. Un cop feta la interconnexió, el disseny es troba en un escenari semblant al del capítol 3: tenim un circuit format pels elements físics que conformen el disseny i tenim elements de gestió del temps necessaris per dur a terme un STA. La diferència és que en aquest cas no cal fer l'anàlisi de forma manual ja que l'eina de síntesi física és capaç de dur-la a terme de forma automàtica per mitjà d'un conjunt d'instruccions. D'aquesta forma, el script *4_route.tcl* realitza un anàlisi temporal del *layout* i optimitza *timing violations* de tot tipus, ja siguin corresponents al temps de *setup*, *hold* o fins i tot *Design Rule Violation*s (DRV). Com en aquest cas no tenim problemes temporals, no suposa un part rellevant del procés.

4.2.7. Crosstalk

La següent part del script *4_route.tcl* tracta el soroll i els efectes paràsits provocats per la interconnexió. El *crosstalk* és un fenomen que ha aparegut sempre en VLSI però va començar a fer-se més important quan la tecnologia amb que es treballava era a escala sub-micromètrica [21]. Actualment és un dels factors que més afecten al rendiment d'un disseny degut a que les dimensions dels circuits integrats són tant petites que és comú que s'hi introdueixi soroll al disseny a través d'interconnexions que són massa properes fins al punt de crear-se connexions paràsites capacitives. Aquest soroll introduït pot provocar retards temporals (*crosstalk delays*) que alenteixen el funcionament del circuit o variacions en l'amplitud de les senyals que travessen el *path* afectat (*crosstalk glitches*) generant errors en el funcionament del circuit [12][21].

És per aquest motiu que resulta important optimitzar el *layout* per evitar la presència d'aquests efectes paràsits. A l'anàlisi temporal de la secció anterior es van detectar *crosstalk glitches* però, com l'eina de STA no és capaç de corregir-los, la part de *signoff* del script tracta la correcció i optimització del *crosstalk.*

4.2.8. Altres optimitzacions

La següent part del script correspon a petits scripts d'optimització de post-interconnexió, de via i de correcció de les regles de disseny (DRC) que milloren una mica el rendiment del *layout* però no són gaire rellevants a nivell teòric.

4.2.9. Inserció de filler cells, decap cells i metal fill

Un cop s'ha realitzat la CTS, el STA i s'han corregit errors, és una pràctica habitual omplir els espais buits entre cel·les amb el que es coneixen com *filler cells* i *decap cells* [19].

Les *filler cells* són utilitzades per mantenir la continuïtat dels pous N (*N-wells*) i evitar problemes de DRC per l'espai entre pous N. Per altra banda, les *decap cells* són capacitats extrínseques que es connecten a les malles d'alimentació i actuen com a *buffers* per reduir efectes de soroll com els *glitches* [19].

Posteriorment a la inserció de les *filler* i *decap cells*, s'omple la resta d'espais buits del *layout* amb metall per complir amb les regles de densitat de metall [12][19].

4.2.10. Signoff

Un cop finalitzada l'etapa de *routing* i realitzades les optimitzacions temporals, resta dur a terme un últim anàlisi per detectar i corregir una sèrie de problemes en preparació al procés de fabricació [12][15]:

- Problemes espacials relacionats amb les capes de metall on es verifica per mitjà de DRC que es compleixen les regles de disseny i que les DRV han sigut corregides.
- Problemes elèctrics verificant mitjançant *Electric Rule Checks* (ERC) que la connectivitat elèctrica del *layout* és correcta.

Aquest procés es coneix com a *design signoff* i és l'etapa d'anàlisi final prèvia a l'exportació de la base de dades amb la descripció estructural del *layout* en format *Graphic Data System* (GDSII).

Aquest procés d'exportació del fitxer GDSII es coneix com a *design tapeout* i proporciona les dades necessàries per començar el procés de fabricació [12].

La part del script *4_route.tcl* corresponent al *signoff* no ha pogut corregir els *crosstalk glitches*. És probable que el motiu sigui l'elevada densitat de metall (81.841%), que congestiona el *layout* i fa inevitable que apareguin efectes paràsits d'interconnexió. Per falta de temps es considerarà que els scripts de síntesi funcionen correctament i es continuarà amb el projecte. En cas que també apareguin problemes d'interconnexió en la síntesi a 400MHz, es procedirà a buscar una solució.

4.3. Layout final a 400MHz i observacions

Com que s'ha verificat que els scripts de síntesi funcionen correctament, es procedeix a realitzar la segona síntesi incrementant la freqüència de treball a 400MHz. Abans de començar el procés de síntesi, considerant els problemes previs amb la densitat de metall, s'ha pres la decisió d'augmentar les dimensions de la matriu del *floorplan* fins a una amplada i alçada de 700nm i 750nm respectivament.

Durant l'anàlisi *post-CTS* s'ha observat que apareixen *setup timing violations* tal com es mostra a les Figures 29 i 30. Tot i així, s'ha decidit continuar amb l'etapa de *routing* ja que al final d'aquesta es realitza una optimització dels temps de *setup* que pot ser que solucioni els problemes temporals de la mateixa manera que el script *3_cts.tcl* optimitza els temps de *hold*. De no ser així, serà necessari fer un *debug* temporal per revisar què ha anat malament.

```
** Profile ** Overall slacks :  cpu=0:00:18.9, mem=15054.7M
** Profile ** Total reports :  cpu=0:00:00.7, mem=14993.9M
** Profile ** DRVs :  cpu=0:00:13.0, mem=15007.3M

------------------------------------------------------------------
     opt_design Final SI Timing Summary
------------------------------------------------------------------

Setup views included:
 av_normal_typ av_normal_max av_normal_min av_normal_min_lt

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):| -0.569  |  0.033  | -0.569  |
|           TNS (ns):| -28.456 |  0.000  | -28.456 |
|    Violating Paths:|   134   |    0    |   134   |
|          All Paths:|  18145  |  9328   |  13070  |
+--------------------+---------+---------+---------+

+----------------+-------------------+------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +-------------------+------------+------------------+
|                |  Nr nets(terms)   | Worst Vio  |  Nr nets(terms)  |
+----------------+-------------------+------------+------------------+
|   max_cap      |      0 (0)        |   0.000    |      0 (0)       |
|   max_tran     |     79 (180)      |  -0.637    |     79 (180)     |
|   max_fanout   |      0 (0)        |     0      |      0 (0)       |
|   max_length   |      0 (0)        |     0      |      0 (0)       |
+----------------+-------------------+------------+------------------+

Density: 60.744%
Total number of glitch violations: 0
------------------------------------------------------------------
** Profile ** Report data :  cpu=0:00:00.5, mem=15008.8M
**opt_design ... cpu = 0:19:02, real = 0:04:10, mem = 11101.5M, totSessionCpu=7:26:42 **
 ReSet Options after AAE Based Opt flow
*** Finished opt_design ***
cleaningup cpe interface
      flow.cputime  flow.realtime  timing.setup.tns  timing.setup.wns  snapshot
UM:*                                  -28.456 ns         -0.569 ns  final
```

Fig 29. Captura de l'anàlisi temporal post-CTS per el temps de setup amb timing violations (síntesi a 400MHz).

```
1 ############################################################
2 #  Generated by:      Cadence Innovus 21.35-s114_1
3 #  OS:                Linux x86_64(Host ID mise-cadence-inv.uab.cat)
4 #  Generated on:      Tue Jan 30 08:53:54 2024
5 #  Design:            cva6
6 #  Command:           opt_design -post_cts -hold
7 ############################################################
8
9 ------------------------------------------------------------------
10     opt_design Final Summary
11 ------------------------------------------------------------------
12
13 +--------------------+---------+---------+---------+
14 |     Hold mode      |   all   | reg2reg | default |
15 +--------------------+---------+---------+---------+
16 |           WNS (ns):|  0.093  |  0.093  |  0.323  |
17 |           TNS (ns):|  0.000  |  0.000  |  0.000  |
18 |    Violating Paths:|    0    |    0    |    0    |
19 |          All Paths:|  18145  |  9328   |  13070  |
20 +--------------------+---------+---------+---------+
21
22 +----------------+-------------------------------+------------------+
23 |                |             Real              |       Total      |
24 |    DRVs        +------------------+------------+------------------|
25 |                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
26 +----------------+------------------+------------+------------------+
27 |   max_cap      |      1 (1)       |   -0.006   |      1 (1)       |
28 |   max_tran     |      9 (20)      |   -0.247   |      9 (20)      |
29 |   max_fanout   |      0 (0)       |     0      |      0 (0)       |
30 |   max_length   |      0 (0)       |     0      |      0 (0)       |
31 +----------------+------------------+------------+------------------+
32
33 Density: 58.097%
34 Routing Overflow: 0.56% H and 0.55% V
35 ------------------------------------------------------------------
36
```

Fig 30. Captura de l'anàlisi temporal post-CTS per el temps de hold (síntesi a 400MHz).

Un cop realitzada la interconnexió, l'anàlisi *post-route* continua indicant que hi han un gran nombre de *setup timing violations*.

```
1 ############################################################
2 #  Generated by:      Cadence Innovus 21.35-s114_1
3 #  OS:                Linux x86_64(Host ID mise-cadence-inv.uab.cat)
4 #  Generated on:      Sun Feb  4 22:57:48 2024
5 #  Design:            cva6
6 #  Command:           time_design -post_route -path_report -drv_report -slack_report -num_paths 50 -report_prefix cva6_postRoute -report_dir timingReports
7 ############################################################
8
9 ------------------------------------------------------------------
10         time_design Summary
11 ------------------------------------------------------------------
12
13 +--------------------+---------+---------+---------+
14 |     Setup mode     |   all   | reg2reg | default |
15 +--------------------+---------+---------+---------+
16 |           WNS (ns):| -0.570  |  0.025  | -0.570  |
17 |           TNS (ns):| -28.573 |  0.000  | -28.573 |
18 |    Violating Paths:|   121   |    0    |   121   |
19 |          All Paths:|  18145  |  9328   |  13070  |
20 +--------------------+---------+---------+---------+
21
22 +----------------+-------------------------------+------------------+
23 |                |             Real              |       Total      |
24 |    DRVs        +------------------+------------+------------------|
25 |                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
26 +----------------+------------------+------------+------------------+
27 |   max_cap      |      0 (0)       |   0.000    |      0 (0)       |
28 |   max_tran     |     80 (179)     |   -0.638   |     80 (179)     |
29 |   max_fanout   |      0 (0)       |     0      |      0 (0)       |
30 |   max_length   |      0 (0)       |     0      |      0 (0)       |
31 +----------------+------------------+------------+------------------+
32
33 Density: 60.744%
34        (100.000% with Fillers)
35 Total number of glitch violations: 0
36 ------------------------------------------------------------------
37
```

Fig 31. Captura de l'anàlisi temporal post-route del temps de setup amb timing violations (síntesi a 400MHz).

Cal tenir en compte que els problemes temporals de *setup* i de *hold* generalment són generats per motius diferents. Les *setup timing violations* depenen de la freqüència del rellotge, mentre que les *hold timing violations* indiquen problemes de funcionalitat en el *layout* [22]. Resulta estrany que apareguin errors relacionats amb la freqüència quan s'està fent servir una freqüència de treball que ha

donat bons resultats durant la síntesi lògica. És per aquest motiu que es procedeix a revisar manualment els *timing paths* mitjançant un *timing debug*.

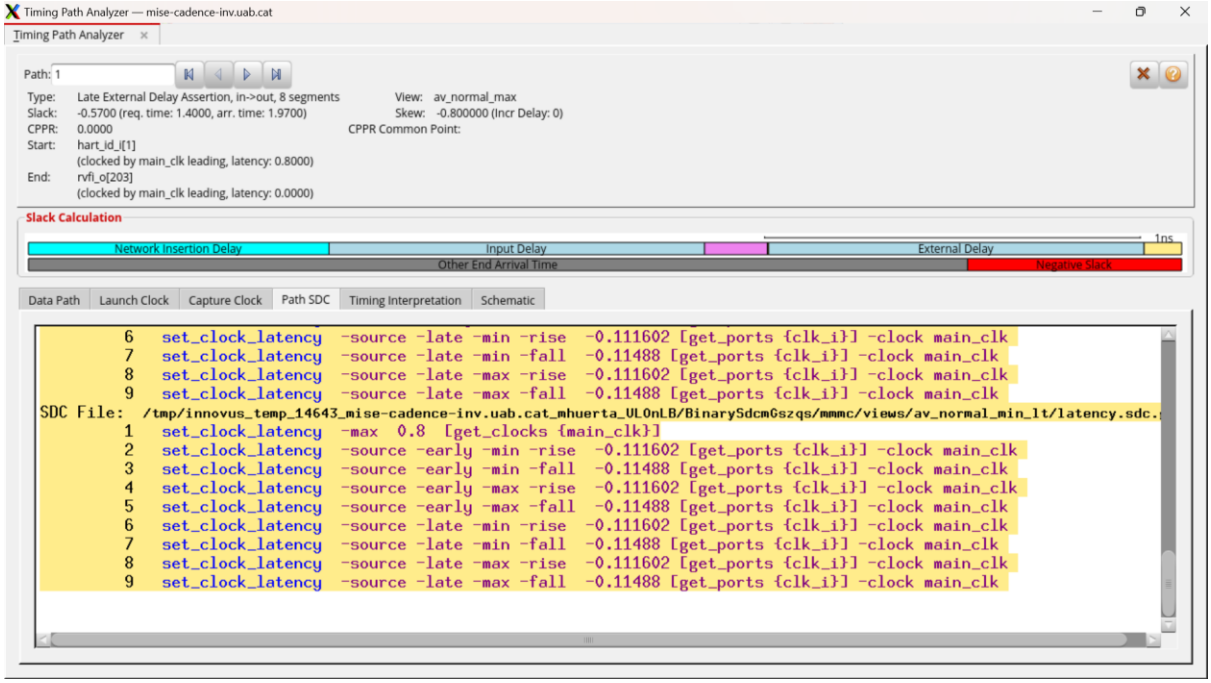


Fig 32. Anàlisi dels retards d'un timing path a una freqüència de 400MHz.

En la Figura 32 mostra que els *paths* presenten uns valors notables de *slack* negatiu, és a dir, que el temps de propagació supera considerablement el període del rellotge. Revisant els diferents tipus de retards s'ha trobat que la latència de xarxa és inusualment elevada i el *skew* és idèntic al que es va introduir manualment durant la síntesi lògica. Això es deu a que l'alumne no va modificar els paràmetres temporals del fitxer SDC, fent ús dels mateixos *constraints* temporals utilitzats durant la síntesi lògica on s'hi introduïen uns valors fixes de latència de xarxa i *skew* de *setup*/*hold* per simular els efectes d'un *clock tree*.

Fig 33. Constraints causants dels problemes temporals de setup.

Un cop eliminades la latència de xarxa i el *skew* (100ps en *setup* i 50ps en *hold*) i tornat a fer el procés de síntesi física, ja no apareixen *timing violations*.

```
#################################################################
#  Generated by:      Cadence Innovus 21.35-s114_1
#  OS:                Linux x86_64(Host ID mise-cadence-inv.uab.cat)
#  Generated on:      Sat Feb  3 17:00:50 2024
#  Design:            cva6
#  Command:           opt_design -post_route -drv
#################################################################

------------------------------------------------------------------
     opt_design Final SI Timing Summary
------------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.073  |  0.077  |  0.073  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|  18145  |  9328   |  13070  |
+--------------------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |      0 (0)       |
|   max_tran     |     10 (30)      |   -0.394   |     10 (30)      |
|   max_fanout   |      0 (0)       |     0      |      0 (0)       |
|   max_length   |      0 (0)       |     0      |      0 (0)       |
+----------------+------------------+------------+------------------+

Density: 54.030%
Total number of glitch violations: 1
------------------------------------------------------------------
```

Fig 34. Captura de l'anàlisi temporal post-route del temps de setup (síntesi a 400MHz).

Tot i haver solucionat els problemes temporals, encara queda pendent intentar corregir el *glitch*. L'objectiu del capítol és obtenir un *layout* funcional i la presència de *crosstalk glitches* compromet la seva fiabilitat. Finalment, el *glitch* és corregit i l'anàlisi *signoff* del *layout* no presenta cap tipus de problema temporal ni de DRV. El slack positiu indica que hi ha marge per optimitzar el layout però per falta de temps s'ha decidit deixar-ho com està.

```
#################################################################
#  Generated by:      Cadence Innovus 21.35-s114_1
#  OS:                Linux x86_64(Host ID mise-cadence-inv.uab.cat)
#  Generated on:      Sat Feb  3 18:48:12 2024
#  Design:            cva6
#  Command:           time_design -sign_off -path_report -drv_report -slack_report -num_paths 50 -report_prefix cva6_signOff -report_dir timingReports
#################################################################

------------------------------------------------------------------
         time_design Summary
------------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.131  |  0.212  |  0.131  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|  18145  |  9328   |  13070  |
+--------------------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |      0 (0)       |
|   max_tran     |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout   |      0 (0)       |     0      |      0 (0)       |
|   max_length   |      0 (0)       |     0      |      0 (0)       |
+----------------+------------------+------------+------------------+

Density: 54.030%
       (100.000% with Fillers)
Total number of glitch violations: 0
------------------------------------------------------------------
```

Fig 35. Captura de l'anàlisi temporal signoff del temps de setup (síntesi a 400MHz).

Un cop finalitzat el procés de síntesi física, s'ha procedit a analitzar el *layout* resultant. Les Figures 36, 37 i 38 mostren el *layout* des de perspectives física, *amoeba* i de *floorplan* respectivament:

- La vista física mostra elements físics del disseny com la interconnexió de les cel·les, de l'alimentació i les *standard cells*.
- La vista *amoeba* mostra principalment la forma dels mòduls que constitueixen el *layout*, tot i que en el nostre cas es tracta d'un únic mòdul, i la ubicació de les *standard cells*.
- La vista *floorplan* mostra objectes relacionats amb el *floorplan*: *constraints* del *floorplan*, *hard macros*, la interconnexió de l'alimentació, etc.

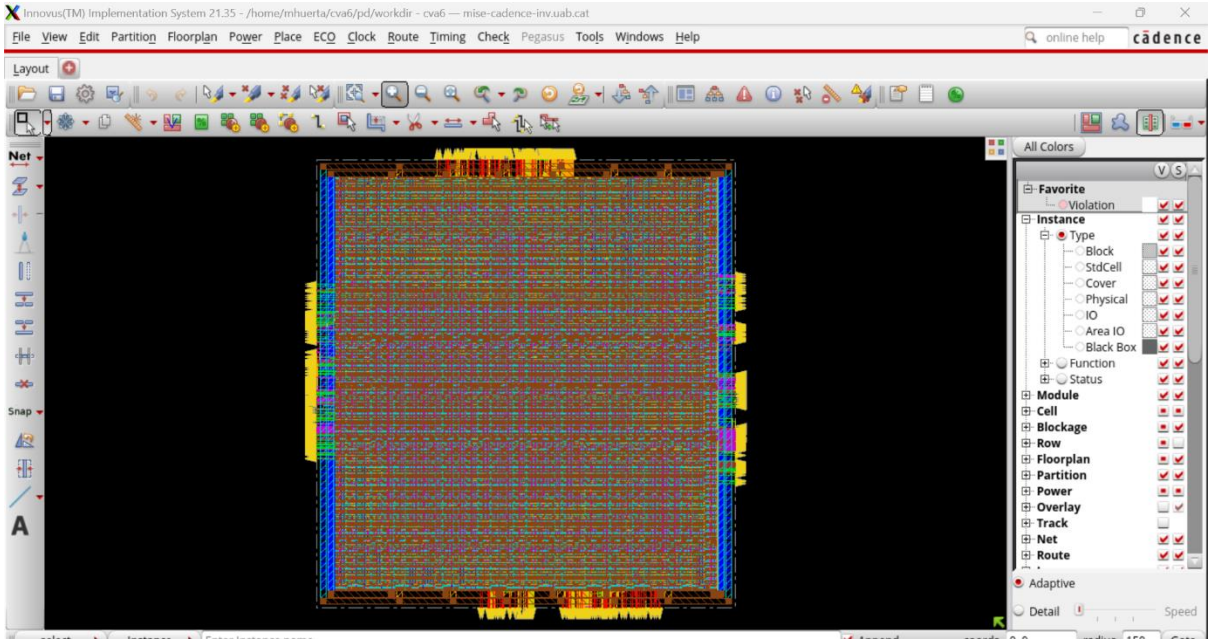


Fig 36. Vista física del layout finalitzat.

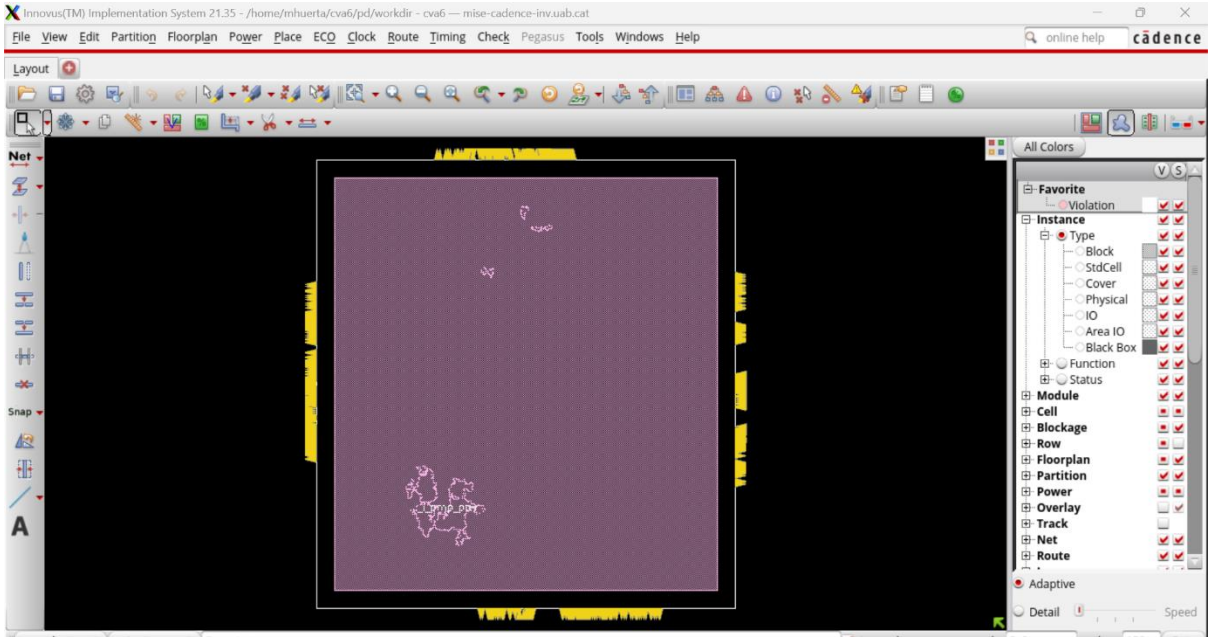


Fig 37. Vista amoeba del layout finalitzat.

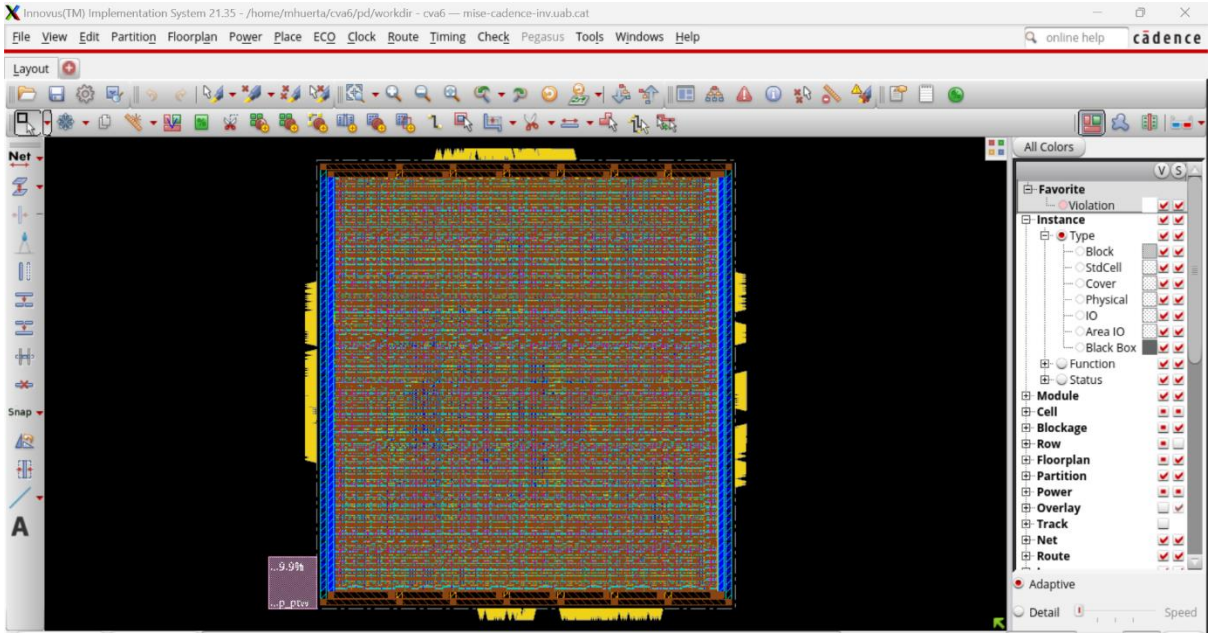


Fig 38. Vista floorplan del layout finalitzat.

Es procedeix a revisar el *layout* mitjançant l'eina CCOpt *Clock Tree Debugger* (CTD), una eina gràfica per analitzar i debugar l'estructura *clock tree*. A la Figura 39 es visualitzen tots els *paths* temporals del *clock tree* on, en verd, es veu l'origen del *paths* i, en vermell, es poden veure les fulles.

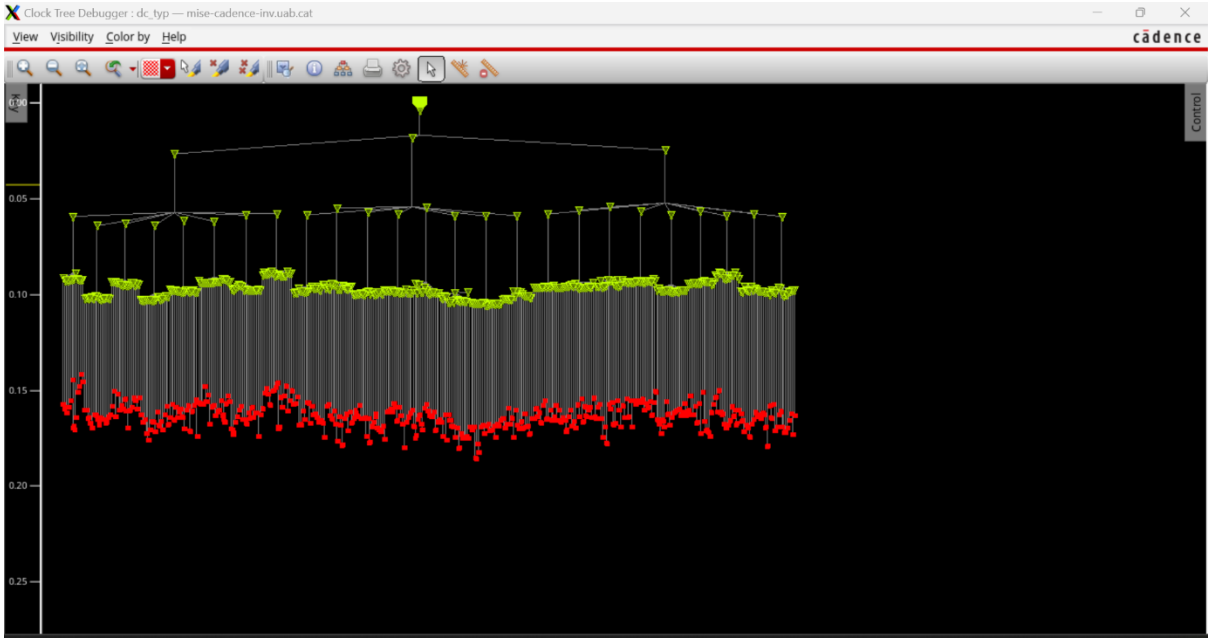


Fig 39. Captura dels timing paths amb l'eina CCOpt.

Dins l'eina de CTD s'ha avaluat el temps de transició del diferents *paths* temporals i els resultats indiquen que els valors de temps de transició es troben dins un rang de valors molt raonable considerant que el temps màxim de transició definit als *constraints* és de 0.125ns.

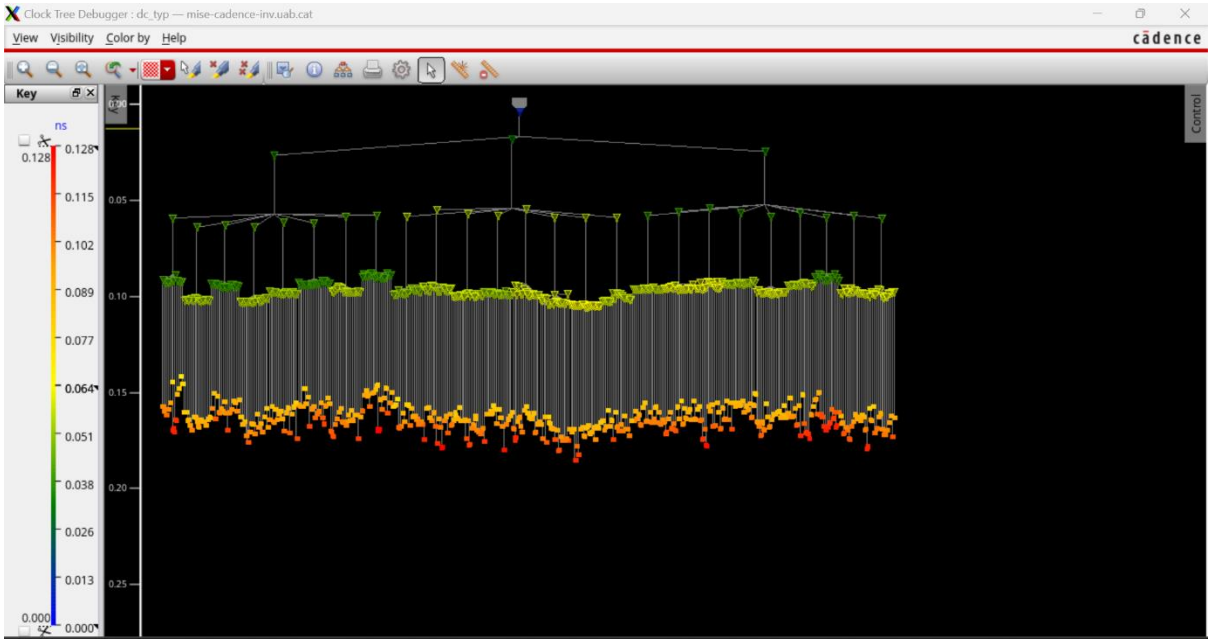


Fig 40. Temps de transició dels timing paths amb l'eina CCOpt.

Per últim, s'ha obert el Clock Path Browser per revisar els paràmetres temporals en al pitjor cas (dc_max:both.late) amb un *skew* de 0.070ns. La Figura 41 mostra gràficament quins són els paths mínim i màxim mentre que les Figures 42 i 43 indiquen els seus paràmetres temporals.

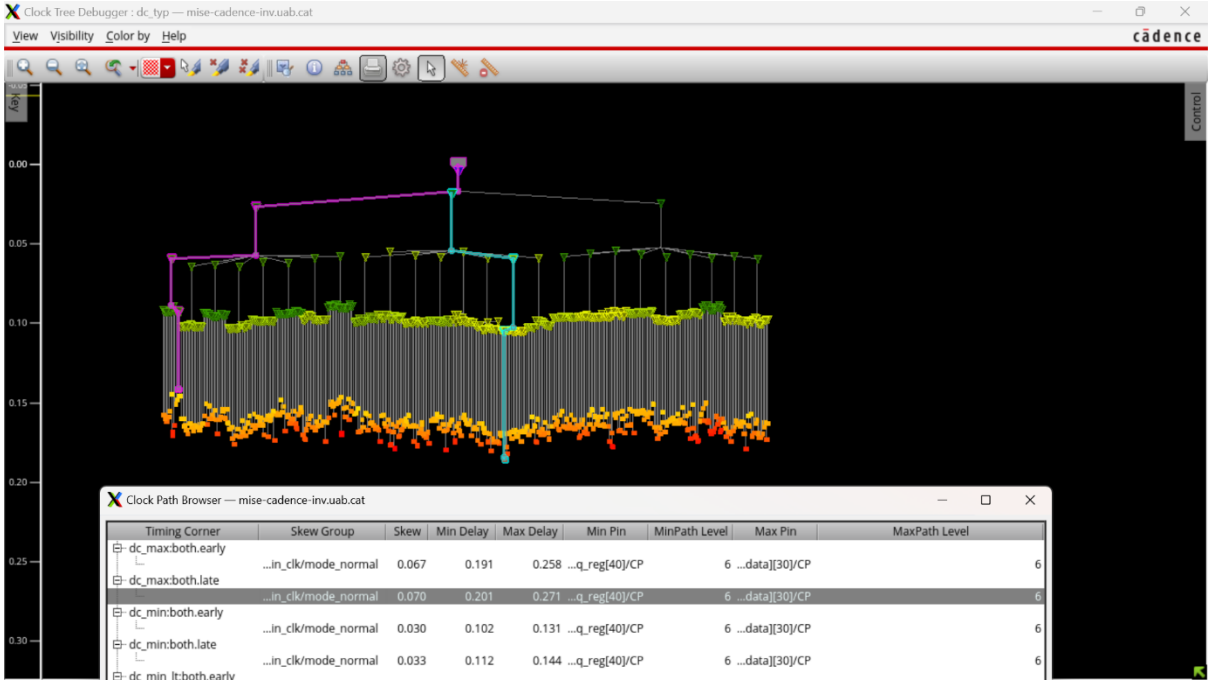
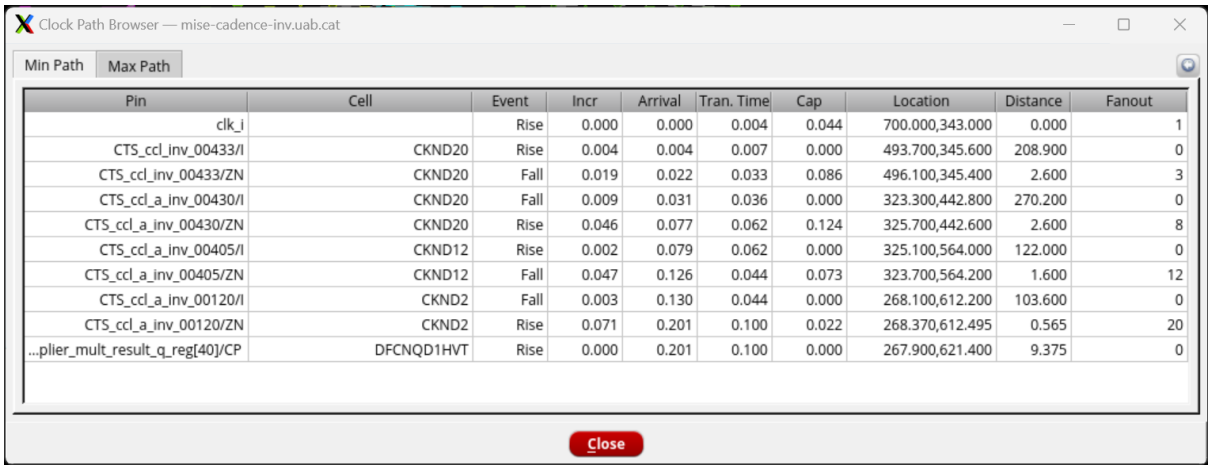


Fig 41. Timing paths per al cas amb major incertesa. Blau: max path. Violeta: min path.



Pin	Cell	Event	Incr	Arrival	Tran. Time	Cap	Location	Distance	Fanout
clk_i		Rise	0.000	0.000	0.004	0.044	700.000,343.000	0.000	1
CTS_ccl_inv_00433/I	CKND20	Rise	0.004	0.004	0.007	0.000	493.700,345.600	208.900	0
CTS_ccl_inv_00433/ZN	CKND20	Fall	0.019	0.022	0.033	0.086	496.100,345.400	2.600	3
CTS_ccl_a_inv_00430/I	CKND20	Fall	0.009	0.031	0.036	0.000	323.300,442.800	270.200	0
CTS_ccl_a_inv_00430/ZN	CKND20	Rise	0.046	0.077	0.062	0.124	325.700,442.600	2.600	8
CTS_ccl_a_inv_00405/I	CKND12	Rise	0.002	0.079	0.062	0.000	325.100,564.000	122.000	0
CTS_ccl_a_inv_00405/ZN	CKND12	Fall	0.047	0.126	0.044	0.073	323.700,564.200	1.600	12
CTS_ccl_a_inv_00120/I	CKND2	Fall	0.003	0.130	0.044	0.000	268.100,612.200	103.600	0
CTS_ccl_a_inv_00120/ZN	CKND2	Rise	0.071	0.201	0.100	0.022	268.370,612.495	0.565	20
...plier_mult_result_q_reg[40]/CP	DFCNQD1HVT	Rise	0.000	0.201	0.100	0.000	267.900,621.400	9.375	0

Fig 42. Paràmetres temporals del min path.

Clock Path Browser — mise-cadence-inv.uab.cat

Min Path | Max Path

Pin	Cell	Event	Incr	Arrival	Tran. Time	Cap	Location	Distance	Fanout
clk_i		Rise	0.000	0.000	0.004	0.044	700.000,343.000	0.000	1
CTS_ccl_inv_00433/I	CKND20	Rise	0.004	0.004	0.007	0.000	493.700,345.600	208.900	0
CTS_ccl_inv_00433/ZN	CKND20	Fall	0.019	0.023	0.033	0.086	496.100,345.400	2.600	3
CTS_ccl_a_inv_00431/I	CKND16	Fall	0.001	0.024	0.034	0.000	477.500,345.600	18.800	0
CTS_ccl_a_inv_00431/ZN	CKND16	Rise	0.055	0.079	0.079	0.131	479.500,345.400	2.200	8
CTS_ccl_a_inv_00423/I	CKND8	Rise	0.005	0.083	0.079	0.000	569.900,456.000	201.000	0
CTS_ccl_a_inv_00423/ZN	CKND8	Fall	0.067	0.150	0.067	0.078	568.900,456.200	1.200	17
CTS_ccl_a_inv_00170/I	CKND2	Fall	0.002	0.152	0.067	0.000	603.300,491.800	70.000	0
CTS_ccl_a_inv_00170/ZN	CKND2	Rise	0.119	0.271	0.176	0.039	603.030,491.505	0.565	20
..._reg[3][sbe][lsu_wdata][30]/CP	DFCNQD1LVT	Rise	0.001	0.272	0.176	0.000	593.700,545.800	63.625	0

Close

Fig 43. Paràmetres temporals del max path.

5. CONCLUSIONS

Aquest treball tenia com objectius principals la formació de l'alumne dins la especialitat i l'aplicació dels coneixements adquirits en un projecte d'implementació física d'un *core* CVA6. Tot i les dificultats durant el procés de síntesi física, s'ha pogut generar un *layout* funcional amb un rendiment temporal prou raonable com per poder considerar-lo un èxit. De totes formes, el termini de temps per realitzar el projecte ha impossibilitat dur a terme alguns dels objectius addicionals que inicialment s'esperava poder assolir, com optimitzar el rendiment temporal del *layout* a la freqüència màxima de treball fins obtenir *slack* zero o implementar les memòries SRAM per avaluar quina seria la freqüència màxima de treball que podria assolir el disseny.

Referències

[1] V. S. Chakravarthi, "Chapter 1: Introduction," in *A Practical Approach to VLSI System on Chip (SoC) Design*, Springer Nature Switzerland, 2020.

[2] F. Ahmed Choudhary, "Logic Synthesis," *www.vlsi-backend-adventure.com*, 2021. https://www.vlsi-backend-adventure.com/logic_synthesis.html (accessed Nov. 01, 2023).

[3] OpenHW Contributors, "CVA6 User Manual," 2023. Accessed: Dec. 01, 2023. [Online]. Available: https://github.com/openhwgroup/cva6/tree/master

[4] S. Sharma, "Understanding RISC-V: The Open Standard Instruction Set Architecture," *www.wevolver.com*, Nov. 14, 2023. https://www.wevolver.com/article/understanding-risc-v-the-open-standard-instruction-set-architecture (accessed Jan. 01, 2024).

[5] S. Sharma, "RISC-V vs ARM: A Comprehensive Comparison of Processor Architectures," *www.wevolver.com*, Aug. 23, 2023. https://www.wevolver.com/article/risc-v-vs-arm-a-comprehensive-comparison-of-processor-architectures (accessed Jan. 01, 2024).

[6] "List of x86 manufacturers," *Wikipedia*, Oct. 21, 2023. https://en.wikipedia.org/wiki/List_of_x86_manufacturers (accessed Jan. 01, 2024).

[7] "Zhaoxin," *Wikipedia*, Dec. 15, 2023. https://en.wikipedia.org/wiki/Zhaoxin (accessed Jan. 01, 2024).

[8] E. Engheim, "Advantages of RISC-V vector processing over x86 SIMD," *Medium*, Apr. 29, 2022. https://itnext.io/advantages-of-risc-v-vector-processing-over-x86-simd-c1b72f3a3e82 (accessed Jan. 01, 2024).

[9] J. Bhasker and R. Chadha, "Chapter 1: Introduction," in *Static Timing Analysis for Nanometer Designs*, Springer Science & Business Media, 2009.

[10] V. S. Chakravarthi, "Chapter 6: Static Timing Analysis," in *A Practical Approach to VLSI System on Chip (SoC) Design*, Springer Nature Switzerland, 2020.

[11] V. S. Chakravarthi, "Chapter 5: SOC Synthesis," in *A Practical Approach to VLSI System on Chip (SoC) Design*, Springer Nature Switzerland, 2020.

[12] V. S. Chakravarthi, "Chapter 9: SOC Physical Design," in *A Practical Approach to VLSI System on Chip (SoC) Design*, Springer Nature Switzerland, 2020.

[13] J. Bhasker and R. Chadha, "Chapter 2: STA Concepts," in *Static Timing Analysis for Nanometer Designs*, Springer Science & Business Media, 2009.

[14] F. Ahmed Choudhary, "Static Timing Analysis," *www.vlsi-backend-adventure.com*, 2021. https://www.vlsi-backend-adventure.com/sta.html?#17 (accessed Nov. 01, 2023).

[15] J. Bhasker and R. Chadha, "Chapter 10: Robust Verification," in *Static Timing Analysis for Nanometer Designs*, Springer Science & Business Media, 2009.

[16] F. Ahmed Choudhary, "MC/MM/OCV," *www.vlsi-backend-adventure.com*, 2021. https://www.vlsi-backend-adventure.com/mc_mm_ocv.html (accessed Jan. 01, 2024).

[17] F. Ahmed Choudhary, “FloorPlan,” *www.vlsi-backend-adventure.com*, 2021. https://www.vlsi-backend-adventure.com/floorplan.html (accessed Dec. 01, 2023).

[18] Cadence Design Systems, Inc., “Innovus User Guide,” 2022.

[19] F. Ahmed Choudhary, “Cells in Physical Design,” *www.vlsi-backend-adventure.com*, 2021. https://www.vlsi-backend-adventure.com/pd_cells.html#4 (accessed Jan. 01, 2024).

[20] F. Ahmed Choudhary, “CMOS Fundamentals,” *www.vlsi-backend-adventure.com*, 2021. https://www.vlsi-backend-adventure.com/cmos_fundamentals.html (accessed Jan. 01, 2024).

[21] V. S. Chakravarthi, “Chapter 10: SOC Physical Design Verification,” in *A Practical Approach to VLSI System on Chip (SoC) Design*, Springer Nature Switzerland, 2020.

[22] F. Ahmed Choudhary, “Physical Design Q&A,” *www.vlsi-backend-adventure.com*, 2021. https://www.vlsi-backend-adventure.com/pd_qa_42.html (accessed Jan. 01, 2024).

Annexos

Annex 1: Scripts de síntesi lògica

- run_all.tcl:

```
set times(start) [clock seconds]
puts "Hostname : [info hostname]"

source $env(DESIGN_PATH)/common/variables.tcl
source $env(DESIGN_PATH)/synth/scripts/genus_setup.tcl
source $env(DESIGN_PATH)/synth/scripts/elaborate.tcl
source $env(DESIGN_PATH)/synth/scripts/synthesis.tcl
source $env(DESIGN_PATH)/synth/scripts/export.tcl
```

- genus_setup.tcl:

```
##############################################################################
### Genus Setup
##############################################################################
set_db max_cpus_per_server 8
set_db super_thread_servers {localhost localhost localhost localhost localhost}

#### Setup the library and RTL search paths
set_db hdl_language sv
set_db init_lib_search_path "."
#set_db / .script_search_path ". ${SYN_PATH}/scripts"
set_db init_hdl_search_path "$CVA6_REPO_DIR/vendor/pulp-
platform/common_cells/include"
set b " "
#### Set the leakage power effort: suggested high
set_db / .leakage_power_effort $LEAKAGE_PWR_EFFORT

#### Create SV wrapper using interfaces - not compatible with verilog port style
TB
#set_db / .write_sv_port_wrapper true

#### Connects any undriven signal in a module to the specified value [none | 0 | 1
| X], default 'none'
set_db / .hdl_unconnected_value none

#### Logging information level - suggested 7
set_db / .information_level $INFO_LEVEL

#### Reads the libraries.
read_libs -max_libs [subst $DIGLIBS$b$MACROLIBS]

## PLE
## set_db / .lef_library $LEFLIBS
##set_db / .qrc_tech_file $QRC_TECH_FILE
set_db / .cap_table_file $CAPTABLE

#### clock gating
```

```
set_db / .lp_insert_clock_gating $CLOCK_GATING
set_db / .lp_clock_gating_infer_enable $CLOCK_GATING_INFER
#set_db / .lp_clock_gating_prefix "LP_Gating_"

#### If needed specifies the path name of the cell to be used for clock-gating
insertion.
# set_db / .lp_clock_gating_cell [find / -libcell CKLNQD*] $DESIGN

#### Controls whether to propagate the switching activities in the design. The
attribute can have the following values:
set_db / .lp_power_analysis_effort medium

#### Verilog file row and column information tracking
set_db / .hdl_track_filename_row_col $TRACK_HDL_LINE

#### Determines the maximum number of iterations for unfolding a loop construct of
any type.
set_db / .hdl_max_loop_limit 4096

#### Set the naming style to simplyfy scripting
set_db / .hdl_reg_naming_style %s_reg%s
#set_db / .hdl_array_naming_style %s__%d__  ; (DO NOT USE)

#### Forces the optimization of gates with inputs tied to constants independent of
the QoR.
set_db / .iopt_force_constant_removal true

#### Controls how complex ports in the RTL are represented in the generated
netlist.
#### Useful if you write in SystemVerilog and you expect to open your design in
Virtuoso
set_db / .hdl_flatten_complex_port true

#### Geometries in a LEF and DEF can be defined much larger than what will
actually be scribed on the silicon. In such cases, a process shrink factor is
used. Default 1
set_db / .shrink_factor 1

set_db hdl_max_memory_address_range 100000
```

- elaborate.tcl:

```
#### Loads HDL files in the order given into memory.
source $FLIST/Flist.cva6_synth

#### Creates design from HDL entity
elaborate $TOP

#### Load constraints
read_sdc $SYN_SDC/constraints400MHz.sdc

#### Load constraints
check_design -unresolved
```

synthesis.tcl:

```
#synthesis
```

```
set_db syn_generic_effort medium
syn_generic
set_db syn_map_effort medium
syn_map
set_db syn_opt_effort medium
syn_opt
```

- export.tcl:

```
##############################################################################
## Writing output files and reports
##############################################################################

report_dp > $SYN_REPORT/${TOP}_datapath_incr.rpt

report_messages > $SYN_REPORT/${TOP}_messages.rpt

write_snapshot -outdir $SYN_REPORT -tag final

report_summary -directory $SYN_REPORT

write_db -to_file ${SYN_OUTPUT}/${TOP}.db


write_hdl  > ${SYN_OUTPUT}/${TOP}_netlist.v

write_script > ${SYN_OUTPUT}/${TOP}_m.script

##############################################################################
## Writing SDC
##############################################################################

write_sdc > ${SYN_OUTPUT}/${TOP}_netlist.sdc

##############################################################################
## LEC
##############################################################################

### write_do_lec
#write_do_lec -golden_design ${SYN_OUTPUT}/${TOP}_intermediate.v -revised_design
${SYN_OUTPUT}/${TOP}_m.v -logfile  ${LOG_PATH}/intermediate2final.lec.log >
${SYN_OUTPUT}/intermediate2final.lec.do
write_do_lec -golden_design ${SYN_OUTPUT}/${TOP}_intermediate.v -revised_design
${SYN_OUTPUT}/${TOP}_m.v > ${SYN_OUTPUT}/intermediate2final.lec.do

#write_do_lec -revised_design ${SYN_OUTPUT}/${TOP}_m.v -logfile
${LOG_PATH}/rtl2final.lec.log > ${SYN_OUTPUT}/rtl2final.lec.do
write_do_lec -revised_design ${SYN_OUTPUT}/${TOP}_m.v >
${SYN_OUTPUT}/rtl2final.lec.do


##############################################################################
## Final
##############################################################################

set times(end) [clock seconds]
set syn_time [expr $times(end) - $times(start)]
```

```
puts "Final Runtime & Memory."
time_info FINAL
puts "============================"
puts "Synthesis Finished ........."
puts "  Total time: $syn_time s"
puts "============================"

#### file copy [get_db / .stdout_log] ${LOG_PATH}/.

puts "To load a synthesized design, you can do 'read_db ${SYN_OUTPUT}/${TOP}.db'"
```

Annex 2: Scripts de síntesi física

- run_all.tcl:

```
source $env(DESIGN_PATH)/common/variables.tcl           ;
source $env(DESIGN_PATH)/pnr/scripts/0_init_design.tcl   ;# Imports synthesized
design and tech data.
source $env(DESIGN_PATH)/pnr/scripts/1_floorplan.tcl     ;# Floorplan
source $env(DESIGN_PATH)/pnr/scripts/2_place.tcl         ;# Places the design.
source $env(DESIGN_PATH)/pnr/scripts/3_cts.tcl           ;# Performs Clock Tree
Synthesis.
#source $env(DESIGN_PATH)/pnr/scripts/4_route.tcl        ;# Routes the design,
perform DRC and signoff checks, generates output data and export the design.
```

- 0_init_design.tcl:

```
source $env(DESIGN_PATH)/common/variables.tcl

#### Copying cds.lib in the working directory
catch { exec rm cds.lib }
file copy  $env(DESIGN_PATH)/common/cds.lib .

### Creating report folders directories
if {[file exists ${PNR_REPORT}] == 0} {exec mkdir ${PNR_REPORT}}
if {[file exists ${PNR_OUTPUT}] == 0} {exec mkdir ${PNR_OUTPUT}}
if {[file exists ${POWER_REPORT}] == 0} {exec mkdir ${POWER_REPORT}}
if {[file exists ${LEC_REPORT}] == 0} {exec mkdir ${LEC_REPORT}}
if {[file exists ${DRC_REPORT}] == 0} {exec mkdir ${DRC_REPORT}}
if {[file exists ${PLACE_REPORT}] == 0} {exec mkdir ${PLACE_REPORT}}
if {[file exists ${TIMING_REPORT}] == 0} {exec mkdir ${TIMING_REPORT}}

##############################################################################
# GLOBAL SETTINGS
##############################################################################

set_db design_process_node $vars(tech,node)
set_multi_cpu_usage -no_cpu_auto_adjustment true -local_cpu $numCpu -keep_license
true

set_db timing_report_fields {hpin cell delay required arrival edge user_derate
transition load}
```

```
#### Set this global to allow set_input_delay assertion to have an effect on clock
source paths beginning at the clock root
set_db timing_allow_input_delay_on_clock_source false

getenv ENCOUNTER_CONFIG_RELATIVE_CWD
set_db write_def_hierarchy_delimiter {/}
### NOTE the libraries loaded here through $OALib must correspond to the ones
called in the mmmc file
set_db init_power_nets $vars(PWR,net_name)
set_db init_ground_nets $vars(GND,net_name)
set_db init_oa_ref_libs $OALib
set_db init_oa_design_view {floorplan}
set_db init_netlist_files ${SYN_OUTPUT}/${TOP}_netlist.v
set_db init_mmmc_files $env(DESIGN_PATH)/common/mmmc.tcl
set_db delaycal_input_transition_delay {0.1ps}
set_db init_oa_layout_views {layout}
set_db init_oa_abstract_views {abstract}

if { $STDCLIB == "GP"} {
read_physical -lef {(CONFIDENCIAL).lef \
(CONFIDENCIAL).lef \
(CONFIDENCIAL).lef}
}

if { $STDCLIB == "LP"} {
read_physical -lef {(CONFIDENCIAL).lef \
(CONFIDENCIAL).lef \
(CONFIDENCIAL).lef}
}

read_mmmc $env(DESIGN_PATH)/common/mmmc.tcl
##############################################################################
# INIT DESIGN
##############################################################################
#### Initializes a design using the Tcl globals.
#read_physical -oa_ref_libs $OALib
set_db init_design_uniquify true
read_netlist $env(DESIGN_PATH)/workdir/syn_output/$env(TOP)_netlist.v
uniquify $env(TOP) -verbose

init_design
#### Fits the entire design in the viewable window. You can use this command at
any stage of the design flow to view the design in entirety
gui_fit

### Sets the view to floorplan view. This view displays the hierarchical module
and block guides, connection flightlines, and floorplan objects, including block
placement, and power and ground nets.
gui_set_draw_view fplan

##############################################################################
# POWER GROUND CONNECT
##############################################################################

connect_global_net $vars(PWR,net_name) -type pg_pin -pin_base_name
$vars(PWR,pin_name) -inst *
connect_global_net $vars(GND,net_name) -type pg_pin -pin_base_name
$vars(GND,pin_name) -inst *
connect_global_net $vars(PWR,net_name) -type tiehi
```

```
connect_global_net $vars(GND,net_name) -type tielo
```

- 1_floorplan.tcl:

```
create_floorplan \
    -site $vars(core,name) \
    -die_size [list $vars(fp,width) $vars(fp,height) $vars(fp,io_core_space)
$vars(fp,io_core_space) $vars(fp,io_core_space) $vars(fp,io_core_space)]

gui_fit

if {$vars(cell,tieHiLo) != ""} {
    set_db add_tieoffs_max_distance 20 ;
    set_db add_tieoffs_cells $vars(cell,tieHiLo)
}

# ##############################################################################
# ADDING POWER RAILS and STRIPES
# ##############################################################################

#### Create options for the ring
set_db add_rings_avoid_short 0
set_db add_rings_extend_over_row 0
set_db add_rings_ignore_rows 0
set_db add_rings_orthogonal_only true
set_db add_rings_target default
set_db add_rings_skip_shared_inner_ring none
set_db add_rings_skip_via_on_pin {standardcell}
set_db add_rings_skip_via_on_wire_shape {noshape}
set_db add_rings_stacked_via_bottom_layer M1
set_db add_rings_stacked_via_top_layer AP
set_db add_rings_via_using_exact_crossover_size 1

set_db add_rings_avoid_short 0
set_db add_rings_extend_over_row 0
set_db add_rings_ignore_rows 0
set_db add_rings_orthogonal_only true
set_db add_rings_target default
set_db add_rings_skip_shared_inner_ring none
set_db add_rings_skip_via_on_pin {standardcell}
set_db add_rings_skip_via_on_wire_shape {noshape}
set_db add_rings_stacked_via_bottom_layer M1
set_db add_rings_stacked_via_top_layer AP
set_db add_rings_via_using_exact_crossover_size 1


# Create the ring, Top and Bottom in M5, Left and Right in M4
add_rings -nets "VDD VSS" \
    -type core_rings -follow core \
    -layer   "top $vars(ring,top_layer)  bottom $vars(ring,bottom_layer)  left
$vars(ring,left_layer)  right $vars(ring,right_layer)"  \
    -width   "top $vars(ring,top_width)  bottom $vars(ring,bottom_width)  left
$vars(ring,left_width)  right $vars(ring,right_width)"  \
    -spacing "top $vars(ring,top_space)  bottom $vars(ring,bottom_space)  left
$vars(ring,left_space)  right $vars(ring,right_space)"  \
    -offset  "top $vars(ring,top_offset) bottom $vars(ring,bottom_offset) left
$vars(ring,left_offset) right $vars(ring,right_offset)" \
```

```
    -center 0 -extend_corner {} -threshold 0 \
    -jog_distance 0 -snap_wire_center_to_grid None

add_stripes \
    -direction $vars(stripe,dir) -spacing $vars(stripe,space) -layer
$vars(stripe,metal) -width $vars(stripe,width) \
    -nets "VDD VSS" \
    -create_pins 1 \
    -start_from left \
    -start_offset 90 \
    -set_to_set_distance 90

# ##############################################################################
# POWER ROUTING and CONNECTION
# ##############################################################################

### sroute: Routes power structures.
route_special -connect {core_pin}  \
    -allow_layer_change 0  \
    -allow_jogging 1 \
    -floating_stripe_target ring

# ##############################################################################
# SAVE DESIGN
# ##############################################################################
write_db -current_top_cell_only ./pnr_output/floorplan.dat
```

- 2_place.tcl:

```
reset_db -category place

set_db place_global_cong_effort high
set_db place_global_clock_gate_aware 1
set_db place_global_clock_power_driven 0

set_db place_global_cong_effort low
set_db place_global_clock_gate_aware false
set_db place_global_clock_power_driven false
set_db place_global_reorder_scan true
set_db place_global_ignore_spare true
set_db place_global_place_io_pins true
set_db place_global_module_aware_spare false
set_db place_detail_preserve_routing false
set_db place_detail_remove_affected_routing false
set_db place_detail_check_route false
set_db place_detail_swap_eeq_cells false

set_db opt_buffer_assign_nets true

set_db opt_preserve_all_sequential true
gui_set_draw_view place

## timingDriven 1 missing
## modulePlan 1 missing

# set_db place_global_ignore_scan false
```

```
set_db route_early_global_effort_level low
set_db route_early_global_top_routing_layer $top_routing_layer
set_db route_early_global_bottom_routing_layer $bottom_routing_layer

gui_set_draw_view place

# ##############################################################################
#  If needed - Add well taps
# ##############################################################################

if {$vars(cell,welltap) != ""} {
    add_well_taps \
      -cell $vars(cell,welltap) \
      -cell_interval $vars(cell,welltap,gap) \
      -in_row_offset 7 \
      -start_row 1 \
      -prefix WELLTAP \
      -checker_board
}

# if {$MIXED_TRACKS} { add_well_taps -cell TAPCELLBWP7T -cell_interval 14 -
in_row_offset 9 -start_row 1 -prefix WELLTAP -checker_board -power_domain
core7thvt }
# if {$ONLY_7TRACKS} { add_well_taps -cell TAPCELLBWP7T -cell_interval 14 -
in_row_offset 9 -start_row 1 -prefix WELLTAP -checker_board }

# ##############################################################################
#  If needed - Trace scan-chain
# ##############################################################################

#if {[file exists "${SYN_OUTPUT}/scan.def"]} {
#    read_def ${SYN_OUTPUT}/scan.def
    # use specifyScanChain if no def available
#    trace_scan -comp_logic -verbose
#    report_scan_chain_partitions
#}

# ##############################################################################
#  If needed - set_path_group_options
# ##############################################################################

if {0} {
    set_path_group_options reg2out -effort_level high
    set_path_group_options in2reg  -effort_level high
    set_path_group_options reg2reg -effort_level high
}

# ##############################################################################
#  PLACE
# ##############################################################################

### Reports the available always-on buffers and inverters per domain.
report_always_on_buffer

### Checks for missing or inconsistent library and design data at any stage of the
design and writes the results to a text and HTML report.
check_legacy_design -all

### For spreading more the cells to simplify the routing
```

```
# set_db place_global_module_padding 1.2
# set_db place_global_uniform_density true

### Executes pre-CTS flow with both placement and pre-CTS optimization.
place_opt_design -expanded_views

### To perform rapid timing optimization for design prototyping, use the following
commands:
#setOptMode -effort
#place_opt_design

# NOTE: preRoute Optimization
### There is a standalone command called congRepair that can be called in any part
of the preRoute flow to attempt to relieve congestion. The command uses
globalRoute + incremental placement. This can have a significantly detrimental
effect on timing and often requires additional optDesign
#congRepair

# ##############################################################################
# ADDING TIEHI and TIELO
# ##############################################################################

set_db add_tieoffs_max_distance 20

add_tieoffs -lib_cell $vars(cell,tieHiLo)

if {[file exists ${PNR_DATA}/place_tiehilo.tcl ]} {
    source ${PNR_DATA}/place_tiehilo.tcl
} else {
    if { $vars(cell,tieHiLo) != "" } {
        add_tieoffs -lib_cell $vars(cell,tieHiLo)
    }
}

# ##############################################################################
# POWER ANALYSIS and DRC CHECK
# ##############################################################################

delete_routes -type signal

delete_drc_markers
check_drc -out_file ${DRC_REPORT}/verifyGeometry_place.rpt

time_design -pre_cts -report_prefix preCTS -report_dir ${TIMING_REPORT} -
timing_debug_report

set_power_output_dir ${POWER_REPORT}
report_power -out_file ${POWER_REPORT}/preCTS.rpt
report_power -insts * -out_file ${POWER_REPORT}/preCTS_instances.rpt

# ##############################################################################
# Logical Equivalence Check (LEC)
# ##############################################################################

#if {[file exists "${SYN_OUTPUT}/scan.def"]} {

#    run_lec -golden ${SYN_OUTPUT}/r2g_withscan.v -setup_only ../../lec/pnr -
user_commands "add pin constraints 0 SCAN_* -both"
#    # lec -lpgxl -64 -nogui -dofile ${LEC_REPORT}/place/runLEC.do
```

```
#} else {

#    run_lec -golden ${SYN_OUTPUT}/r2g.v -setup_only ${LEC_REPORT}/place
#    # lec -lpgxl -64 -nogui -dofile ${LEC_REPORT}/place/runLEC.do
#}


# ##############################################################################
# SAVE DESIGN
# ##############################################################################

#write_db -oa_lib_cell_view "$env(oaLibDir) $env(oaLibName) place"
write_db -current_top_cell_only ./pnr_output/place.dat
```

- 3_cts.tcl:

```
source $env(DESIGN_PATH)/common/variables.tcl

# ##############################################################################
# CTS SETTINGS
# ##############################################################################
### Ensure that sufficient analysis view are active.
### Clock specification generation is fully multi-mode and it is important that
views using both functional and scan modes are active.

# ##############################################################################
# Sets delay scaling or derating factors
# ##############################################################################

### Sets delay scaling or derating factors for early and late paths in the design.
#   You use this command to perform scaled on-chip variation around best-case and
worst-case corners

source $env(DESIGN_PATH)/common/timing_derate.tcl

# ##############################################################################
# CTS and Optimization settings
# ##############################################################################

### Sets global analysis modes for timing analysis.
### onChipVariation --- Calculates the delay for one path based on maximum
operating condition while calculating the delay for another path based on minimum
operating condition for setup or hold checks
set_db timing_analysis_type ocv

### Removes pessimism from clock paths that have a portion of the clock network in
common between the clock source and clock destination paths
## Enables removal of clock reconvergence pessimism for both setup and hold modes
when both is specified
set_db timing_analysis_cppr both

###Reports violations for hold and setup checks
set_db timing_analysis_check_type setup

### If this property is set, the CTS algorithm will move logic that appears in the
clock tree. "Logic" does not include clock gates, buffers, and inverters in a
```

```
clock tree, which are always moved unless they are locked, or clock generators
that are above the root of the clock tree
set_db cts_move_logic true

set_db cts_consider_power_intent true

set leakage_power_effort none
set dynamic_power_effort none

### Specifies whether CCOpt will balance this clock tree. If set to true, CCOpt
will not balance or optimize this clock tree. The default is false.
# set_db cts_opt_ignore false

#### Specifies whether to run diagnostic checks to ensure that clock nets routed
by NanoRoute follow route guides. The diagnostic checks compare lengths of
estimated routes to the final detailed wiring.
set_db cts_check_route_follows_guide true

### Tells CCOpt to rename all clock tree nets for easy identification later in the
flow.
set_db cts_rename_clock_tree_nets true

### This property specifies the current waveform calculation methods when
considering EM constraints.No or incorrect method means NOT considering EM
constraints.
# set_ccopt_property consider_em_constraints avg

### Configure library cells for CTS to use. CTS will use matching cells which
arem't dont_use
set_db cts_buffer_cells        $vars(cell,clk_bufs)
set_db cts_inverter_cells      $vars(cell,clk_invs)
#set_db cts_clock_gating_cells  $vars(cell,clk_gating)

### Include this setting to use inverters in preference to buffers.
### For many, but not all, low geometry processes inverters result in lower
insertion delay and power than buffers.
set_db cts_use_inverters true

### The clock input pins of MACROS must usually be earlier than other sinks, which
means they will have a lesser clock arrival time to take account of the internal
clock path inside the macro.
### This can be specified in SDC timing constraints through set_clock_latency if
this is represented by a pin specific network latency, otherwise:
# set_ccopt_property -pin mem1/CLK insertion_delay 1.2ns

set_db opt_area_recovery false
set_db opt_remove_redundant_insts true

set_db opt_add_insts true
set_db opt_delete_insts true
set_db opt_move_insts true
set_db opt_down_size_insts true

set_db opt_detail_drv_failure_reason true
set_db opt_fix_hold_allow_overlap auto

set_db opt_post_route_fix_glitch true
set_db opt_post_route_fix_si_transitions true
```

```
set_db opt_fix_hold_allow_setup_tns_degradation true
set_db opt_fix_hold_allow_resize  true
set_db opt_fix_hold_on_excluded_clock_nets false

set_db opt_fix_hold_verbose true

set_db opt_setup_target_slack 0.2
set_db opt_hold_target_slack 0.2

set_db opt_max_density 0.90
set_db opt_drv_margin 0

set_db opt_leakage_to_dynamic_ratio 1.0

#setOptMode
# -effort high  TO FIND
# -yieldEffort none TO FIND

#if {$use_useful_skew} {
    ### enable usefull skew at CTS and optimization
#    set_db timing_analysis_useful_skew  true
#    set_db timing_analysis_check_type  setup
#    set_db opt_useful_skew_delay_pre_cts true
#    set_db opt_useful_skew_pre_cts true
#    set_db opt_useful_skew_post_route true
#    set_db opt_useful_skew true
#    set_db opt_useful_skew_ccopt standard
    ## To limit usage to where usefull
#    set_db opt_useful_skew_min_allowed_delay 0.2
#} else {
    ### disable usefull skew
    set_db opt_useful_skew_delay_pre_cts false
    set_db opt_useful_skew_pre_cts false
    set_db opt_useful_skew_post_route false
    set_db opt_useful_skew false
    set_db opt_useful_skew_ccopt none
#}

##############################################################################
### Clock tree routing layers
##############################################################################

set clk_tree_top_layer     $clk_top_routing_layer
set clk_tree_bottom_layer  $clk_bottom_routing_layer

set route_clock_nets true
set ccopt_effort high

set_db route_design_bottom_routing_layer        $clk_tree_bottom_layer
set_db route_design_top_routing_layer           $clk_tree_top_layer
set_db route_early_global_top_routing_layer     $clk_tree_top_layer
set_db route_early_global_bottom_routing_layer  $clk_tree_bottom_layer

##############################################################################
### Creating NON DEFAULT RULES for clock routing
##############################################################################

source $PNR_SETUP/ndr.tcl
```

```
set_db  cts_route_type_leaf   rt_leaf
set_db  cts_route_type_trunk  rt_trunk
set_db  cts_route_type_top    rt_top


### The clock input pins of MACROS must usually be earlier than other sinks, which
means they will have a lesser clock arrival time to take account of the internal
clock path inside the macro.
### This can be specified in SDC timing constraints through set_clock_latency if
this is represented by a pin specific network latency, otherwise:
# set_ccopt_property \
#  -pin mem1/CLK insertion_delay 1.2ns

update_constraint_mode -name mode_normal -sdc_files
$env(syn_output)/cva6_netlist_postCTS.sdc
# ##############################################################################
# CLOCK TREE SPECS
# ##############################################################################

### Creates a clock tree network with associated skew groups and other clock tree
synthesis (CTS) configuration settings based on a multi-mode timing configuration.

create_clock_tree_spec     \
    -views { av_normal_typ }      \
    -keep_all_sdc_clocks   \
    -out_file ./ctsSpec_ccopt.tcl

source ./ctsSpec_ccopt.tcl

### add custom CTS specs and skew groups
source $PNR_DATA/CTS_custom_rules_and_skew_groups.tcl

 ### This command can be used to perform setup, library and design validation
checks WITHOUT running CTS
ccopt_design  -check_cts_config

# ##############################################################################
# CLOCK TREE GENERATION
# ##############################################################################

ccopt_design

# ##############################################################################
# POST CTS HOLD OPTIMIZATION
# ##############################################################################

### HOLD optimization
opt_design -post_cts -hold

# ##############################################################################
# POST CTS CLOCK SKEW OPTIMIZATION
# ##############################################################################

#if {$use_useful_skew} {
    ### Modifies the clock arrival time on sequential elements in order to improve
the data path timing between two sequential elements
#    opt_clock_skew -post_cts
#}
```

```
# ##############################################################################
# CTS REPORTS
# ##############################################################################

### timing Reports
time_design \
    -post_cts \
    -report_prefix postCTS  \
    -report_dir ${TIMING_REPORT}   \
    -timing_debug_report    \
    -num_paths 10000

time_design -hold   \
    -post_cts   \
    -report_prefix postCTS   \
    -report_dir ${TIMING_REPORT}   \
    -timing_debug_report   \
    -num_paths 10000

### Report on clock trees to check area and other statistics
report_clock_trees -out_file ${PNR_REPORT}/ClockTreeReports/clock_trees.rpt

### Report on skew groups to check insertion delay and skew
report_skew_groups -out_file ${PNR_REPORT}/ClockTreeReports/skew_groups.rpt

### Reports statistics for the entire design
report_summary -no_html -out_dir ${PNR_REPORT}/SummaryReport

# ##############################################################################
# SAVE DESIGN
# ##############################################################################

write_db -current_top_cell_only ./pnr_output/postCTS.dat
```

- 4_route.tcl:

```
# ##############################################################################
# Sets delay scaling or derating factors
# ##############################################################################

source $env(DESIGN_PATH)/common/timing_derate.tcl

# ##############################################################################
# ROUTING SETTINGS
# ##############################################################################

set_interactive_constraint_modes [ all_constraint_modes -active ]

### Controls certain aspects of how the NanoRoute router routes the design.
set_db route_design_bottom_routing_layer $bottom_routing_layer
set_db route_design_top_routing_layer $top_routing_layer
set_db route_design_detail_post_route_wire_widen_rule NA
set_db route_design_with_timing_driven false
set_db route_design_with_si_driven false
set_db route_design_detail_on_grid_only None
set_db route_design_strict_honor_route_rule true
```

```
#the tool will only do post-route via optimization and all normal routing will be
SKIPPED
set_db route_design_detail_use_multi_cut_via_effort high
set_db route_design_concurrent_minimize_via_count_effort high
set_db route_design_detail_fix_antenna true
### The router runs search-and-repair routing during detailed routing. During
search and repair, it locates shorts and spacing violations and reroutes the
affected areas to eliminate as many of the violations as possible.
### Runs a search-and-repair step after the initial detailed routing.
set_db route_design_detail_search_and_repair true
### search and repair number of iterations
set_db route_design_detail_end_iteration 20

### Antenna fixing settings
set_db route_design_detail_fix_antenna true
set_db route_design_antenna_diode_insertion true

### Inserts diodes to repair process antenna violations on clock nets that are in
the regular net section of the DEF file.
set_db route_design_diode_insertion_for_clock_nets true
set_db route_design_antenna_cell_name ANTENNA

### Skip routing for power nets
set_route_attributes -skip_routing true -nets $vars(PWR,net_name)
set_route_attributes -skip_routing true -nets $vars(GND,net_name)

# ##############################################################################
# GLOBAL & DETAILED ROUTING
# ##############################################################################

### -globalDetail : Runs timing-driven and SI-driven global and detailed routing.
### When specified, it unfixes the clock nets and then routes all nets, including
clocks that still need to be routed.
### It does not ECO route the clock nets prior to other signals so clock nets are
not given priority as they are with routeDesign.

route_design -global_detail

### This command checks for routing congestion
#congRepair

write_db -current_top_cell_only ./pnr_output/route_thin.dat

# ##############################################################################
# TIMING ANALYSIS
# ##############################################################################

if {$DETAILED_PNR} {

    time_design  \
        -post_route                             \
        -path_report                            \
        -drv_report                             \
        -slack_report                           \
        -report_prefix route_thin               \
        -report_dir ${TIMING_REPORT}            \
        -timing_debug_report                    \
        -num_paths 1000
```

```
    time_design -hold  \
        -post_route                             \
        -path_report                            \
        -slack_report                           \
        -report_prefix route_thin               \
        -report_dir  ${TIMING_REPORT}           \
        -timing_debug_report                    \
        -num_paths 1000
}

# ##############################################################################
# OPTIMIZATION I
# ##############################################################################

set_db opt_area_recovery true

set_db opt_remove_redundant_insts true
set_db opt_move_insts true
set_db opt_delete_insts true

set_db opt_setup_target_slack 0.2
set_db opt_hold_target_slack 0.2

set_db opt_fix_hold_allow_setup_tns_degradation true

set_db opt_fix_hold_allow_resize true

set_db opt_max_density 0.90
set_db opt_drv_margin 0.1

set_db opt_leakage_to_dynamic_ratio 1.0

set_db  opt_fix_hold_verbose true
set_db  opt_post_route_fix_si_transitions true
set_db  opt_post_route_fix_glitch true
set_db  opt_fix_drv true

set_db si_glitch_enable_report 1
set_db si_delay_enable_report 1

if {$use_useful_skew} {
    set_db  opt_useful_skew true
    set_db  opt_useful_skew_post_route true
}

### SETUP optimization
opt_design -post_route -setup

### HOLD optimization
opt_design -post_route -hold

### DRV optimization
opt_design -post_route -drv

write_db -current_top_cell_only ./pnr_output/route_opt1.dat

if {$DETAILED_PNR} {

    time_design \
```

```
        -post_route \
        -path_report \
        -drv_report \
        -slack_report \
        -report_prefix opt_1\
         -report_dir ${TIMING_REPORT} \
         -timing_debug_report \
         -num_paths 1000

    time_design -hold \
        -post_route \
        -path_report \
        -slack_report \
        -report_prefix opt_1 \
        -report_dir ${TIMING_REPORT} \
        -timing_debug_report \
        -num_paths 1000
}


# ##############################################################################
#  NOISE REPORT
# ##############################################################################

set_db si_glitch_enable_report 1
set_db si_delay_enable_report 1

report_timing
report_noise -out_file ${PNR_REPORT}/noise.txt


# ##############################################################################
# ADDITIONAL POST-ROUTE OPTIMIZATION THAT CAN BE USEFULL
# ##############################################################################

if { $opt_signoff } {
    source ${PNR_DATA}/opt_postroute_signoff_settings.tcl
    proc_opt_signoff  true true
}

if { $postroute_optimization_with_signoff_settings } {
    source ${PNR_DATA}/opt_postroute_signoff_settings.tcl
    proc_postroute_optimization_with_signoff_settings  true true true
}


if { $postroute_opt_with_tempus_eco_flow } {
    source ${PNR_DATA}/opt_with_tempus_eco_flow.tcl
    proc_opt_postroute_with_tempus_eco_flow;
}

write_db -current_top_cell_only ./pnr_output/route_opt2.dat

# ##############################################################################
#  VIA OPTIMIZATION
# ##############################################################################

set_db route_design_detail_fix_antenna true
set_db route_design_antenna_diode_insertion true
```

```
### Swaps single-cut vias for multiple-cut vias or reverses swapping on critical
nets in a fully routed design, so that multiple-cut vias are swapped for single-
cut vias on those nets.
set_db route_design_detail_post_route_swap_via multiCut
set_db route_design_detail_use_multi_cut_via_effort high
set_db route_design_reserve_space_for_multi_cut false
set_db route_design_with_via_in_pin false

route_design -via_opt

# ##############################################################################
# DRC FIX
# ##############################################################################

### This command is part of an ECO flow process and is based on the NanoRoute
router
check_drc -out_file ${DRC_REPORT}/verifyGeometry_route.rpt -limit 100000
delete_routes_with_violations
route_eco
delete_drc_markers
check_drc -out_file ${DRC_REPORT}/verifyGeometry_route_eco.rpt -limit 100000
write_db -current_top_cell_only ./pnr_output/route_drc_eco.dat

# ##############################################################################
# POST ROUTING POWER & MULTI VIA REPORT
# ##############################################################################

set_db power_handle_glitch true
report_power -out_dir ${POWER_REPORT}
report_power -out_file ${POWER_REPORT}/postRoute.rpt
report_power -insts * -out_file ${POWER_REPORT}/postRoute_instances.rpt

report_route -multi_cut

report_route -summary

# ##############################################################################
# ADDING FILLER CELLS WITH DECAP and FILLERS (removed during optimization)
# ##############################################################################

add_fillers -base_cells $vars(cell,decaps) -prefix FILLER_DECAP
add_fillers -base_cells $vars(cell,filler) -prefix FILLER

# addFiller command is running on a postRoute database. It is recommended to be
followed by ecoRoute -target command to make the DRC clean.


route_eco -fix_drc

delete_drc_markers
check_drc -out_file ${PNR_REPORT}/drc/verifyGeometry4p2_postFill.rpt -limit 100000

write_db -current_top_cell_only ./pnr_output/route_cell_fill.dat

# ##############################################################################
# ADDING METAL FILL WITH DECAP
# ##############################################################################

source ${PNR_DATA}/addMetalFill.tcl
```

```
write_db -current_top_cell_only ./pnr_output/route_metal_fill.dat

#setMetalFill -windowStep x_step y_step -windowSize x y
#trimMetalFillNearNet -slackThreshold $slack1 -spacing value -spacingAbove value -
spacingBelow value -minTrimDensity value

delete_floating_nets
delete_dangling_ports
delete_empty_hinsts

delete_place_halo -all_blocks
delete_route_blockages -type all
delete_route_halos -all_blocks


# ##############################################################################
# Route Final Timing chack with signoff settings and QRC extraction
# ##############################################################################

source ${PNR_DATA}/signoff_timing_analysis.tcl

signoff_extract_qrc "route_final" "_route_final"
signoff_time_design "route_final" "_route_final"

catch [ gunzip ./pnr_report/timingReports/Signoff_Final.tran.gz ]
catch [ gunzip ./pnr_report/timingReports/Signoff_Final.SI_Glitches.rpt.gz ]


# ##############################################################################
# SAVE DESIGN
# ##############################################################################

write_netlist ${PNR_OUTPUT}/route.v

write_db -current_top_cell_only ./pnr_output/route_final.dat
```