ASIC PHYSICAL DESIGN OF THE RISC-V BASED OPENPULP CORE

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRONICS

AND ELECTRICAL COMMUNICATIONS ENGINEERING

OF CAIRO UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

BACHELOR OF SCIENCE

By

Mariam Mohammed Mahmoud Mahmoud

Menna Magdy Saad

Monica Nashaat Botros Hakim

Roaa Ahmed Sabek

Walid Fady Akoum

Yara Mamdouh Gohneim

July 2021

ASIC PHYSICAL DESIGN OF THE RISC-V BASED OPENPULP CORE

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRONICS

AND ELECTRICAL COMMUNICATIONS ENGINEERING

OF CAIRO UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

BACHELOR OF SCIENCE

By

Mariam Mohammed Mohammed
Monica Nashaat Botros
Walid Fady Akoum
Menna Magdy Saad
Roaa Ahmed Sabek
Yara Mamdoh Gohneim

Under the supervision of

Dr. Hassan Mostafa
Associate professor

Eng. Hoda Thabet
Senior ASIC team leader

Sponsored by

Si-VISION and ONE Lab

July 2021

Table of Contents

List of figures

List of tables

List of symbols

List of abbreviations

Ri5cy	another name of OpenPULP core
FPGA	Field Programmable Gate Arrays
ASIC	Application specific integrated circuit
ISA	instruction set architecture
PULP	Parallel Ultra Low Power
LSU	Load Store Unit
PMP	Physical Memory Protection
ALU	Athematic Logic Unit
FPU	Floating Point Unit
RTL	Register Transfer Logic

GDSII	data exchange of integrated circuit or IC layout artwork
DC	Design Compiler
CMOS	Complementary metal–oxide–semiconductor
PDK	process design kit
DRC	Design Rule Check
PnR	Place and Route
CTS	clock Tree Synthesis
STA	Static Timing Analysis
LVT	Low Voltage Threshold
ECO	Engineering Change Order
MCMM	Multi-corner multi-mode
HDL	Hardware Description Language
VHDL	very high-speed integrated circuit Hardware Description Language
EDA	Electronic design automation
SDC	Synopsys Design Constraints

Abstract

ASIC design or application specific integrated circuits is the physical implementation of logic designs, which simply put is creating chips that are specialized in a certain function, like the electronic chips found in televisions or DVD players. This branch of engineering has originated more than 20 years ago and grew exponentially with the growth and demand of ICs and electronics. ASIC design can be separated into two stages. The first stage is the frontend stage in which the designer writes RTL code, which is a code that describes the functionality of the hardware. The second stage or backend is the physical implementation of the RTL in which the designer turns the RTL into actual gates. The thesis will be mainly focused on the latter stage.

The design that we will be implementing is a RISC-V ISA based core. An ISA is a group of instructions that helps interface with the hardware which makes it much easier than dealing with the hardware directly, and a core is simply a processor that can either be used for general or specific purposes. RISC-V is an open-source ISA, and because it's open source the RISC-V community is very big in both the open source and the commercial market. It's mainly used in storage, edge computing and AI applications.

The RTL provided to us was written by OpenPULP which is an open-source multi-core computation platform. It is an open-source RTL that describes a RISC-V core. They were chosen because their reputation is good in both literature reviews and websites that promote open-source RTL. Adding as well that the RTL given to us has been verified by our sponsor and supervisor engineers from Si-Vision, and is therefore ready to go through the physical implementation process.

The process of implementation is done through two stages, synthesis and PNR. Synthesis is the stage in which we read the RTL and convert it into logic gates that perform the same logical function. Secondly, PNR is the stage in which we take those gates and place them on the chip finally connecting them together and to the clock that feeds them with metal wires.

However there exists multiple flows in which a designer can use in order finish his design, such as the hierarchal, flat and topographical flow. In this thesis we describe what each of the flows deal with when using the tool provided as well as the final result for each of the flows with respect to timing closure or frequency

used, area and power consumption comparing them in order to know what is the advantage and disadvantage of each of the flows.

The following thesis is a comparative analysis of three different ASIC design flows for a RISC-V core provided by openPULP.

Chapter 1

Introduction

1.1. RISC-V

RISC-V is an open instruction set architecture (ISA) which is standard and based on the principles of established reduced instruction set computer (RISC). Unlike most other open instruction set architecture designs, the RISC-V open instruction set architecture is provided by licenses that are open source and do not require fees to be used. Many companies have announced RISC-V hardware or offered it.

The instruction set is supported in several popular software tool chains and open-source operating systems are available.

1.1.1. Why Instruction Set Architecture Matters?

First of all, we need to know what the instruction set architecture (ISA) is:

ISA defines the software interface for hardware; the single ISA can make many hardware implementations.

The ISA defines:

- Set of instructions and how they behave
- Data types
- Registers
- Addressing models
- How inputs/outputs are done
- Memory models
- Virtual memory
- Protection levels

The ISA somehow decides the application. It is common sense that the world's most famous 2 ISA are x86 and ARM. Their application fields are totally different.

Over 99% of laptops, desktops, and servers are based on X86 or AMD64 ISA. Their IPs belong to Intel and AMD, over 99% of mobile phones, tablets are based on ARM ISA, the IPs are divided into A series, R series, and M series.

While RISC-V is very suitable for use in some specific application fields such as storage, edge computing, and AI applications.

The different applications field makes it possible for RISC-V to compete with ARM and x86.

1.1.2. Kinds of ISA

Table 1: ISA kinds

CISC	RISC
Instruction can take several clock cycles	Single-cycle instructions
Hardware-centric design The ISA does as much as possible using hardware circuitry	Software-centric design High-level compilers take on most of the burden of coding many software steps from the programmer
More efficient use of RAM than RISC	Heavy use of RAM (cam cause bottlenecks if RAM is limited)
Complex and variable length instruction	Simple, standardized instructions
May support microcode (micro-programming where instructions are treated like small programs)	Only one layer of instructions
Large number of instructions	Small number of fixed-length instructions
Compound addressing modes	Limited addressing modes

1.1.2.1 CISC – Complex Instruction Set Computer

The CISC contains not only the common instructions of the processor but also a lot of uncommon instructions (the 28th principle, 80% of the instructions used in the program operation, only 20% of all instructions.

There are many instructions; the typical representative is Intel's x86 architecture. x86-64 is a 64-bit extension of the x86 architecture, designed by AMD and also known as AMD64.

1.1.2.2 RISC – Reduced Instruction Set Computer

Reduced instruction set architecture contains only the instructions commonly used by the processor. For less commonly used operations, the same effect can be achieved by executing multiple commonly used instructions. The reduced instruction set represents more MIPS, RISC-V, Power, Alpha, etc.

1.1.3. What's Different about RISC-V?

Comparing to ARM and x86, RISC-V has below advantages:

- **Free:** RISC-V is open-source; there is no need to pay for the IP.
- **Simple:** RISC-V is far smaller than other commercial ISAs.
- **Modular:** RISC-V has a small standard base ISA, with multiple standard extensions.
- **Stable:** Base and first standard extensions are already frozen. There is no need to worry about major updates.
- **Extensibility:** Specific functions can be added based on extensions. There are many more extensions are under development, such as Vector.
- **Software architects/firmware engineers/software developers:** RISC-V is much more than an open ISA; it is also a frozen ISA. The base instructions are frozen and optional extensions which have been approved are also frozen. Because of the stability of the ISA, software development can confidently be applied to RISC-V knowing that your investment will be preserved. Software written for RISC-V will run on all similar RISC-V cores forever. The frozen ISA provides a solid foundation that software managers can depend on to preserve their software investments. Because the RISC-V ISA is open, this translates to hardware engineers having more flexibility over the processor implementation. With this power, software architects can become more influential in the final hardware implementation. They can provide input to hardware designers to make the RISC-V core more software centric.
- **CTOs/Chip designers/System Architects:** Innovation is the key enabler of RISC-V. Because the ISA is open, it is the equivalent of everyone having a micro architecture license. One can optimize designs for lower power, performance, security, etc. while keeping full compatibility with other designs. Because there is significantly more control over the hardware implementation, all technical recipients of the architecture can make suggestions at a much earlier point than previously was possible.

The result is a solution with significantly fewer compromises. RISC-V also supports custom instructions for designs which require particular acceleration or specialty functions.

- **Board designers:** In addition to the frozen ISA benefits, RISC-V's open ISA can provide several additional benefits. For example, if engineers are implementing a soft RISC-V core in an FPGA, often the RTL source code is available. Since RISC-V is royalty, free this creates significant flexibility to port a RISC-V based design from an FPGA to an ASIC or another FPGA without any software modifications. Designers who are concerned with security from a trust perspective will also appreciate RISC-V. When the RTL source code is available, this enables deep inspection. With the ability to inspect the RTL, one can establish trust.

1.1.4. ISA Base and Extensions of RISC-V

As mentioned earlier, the RISC-V instruction set has modular characteristics. The instruction set is organized in a modular manner. Each module is represented by an English letter.

The instruction set includes the standard part and the extension part. The standard part must be implemented.

For example:

If you want to implement a 32-bit architecture RISC-V processor, the RV32I instruction set must be implemented on the hardware (machine mode must also be implemented in privileged mode).

The basic integer ISA and the machine privilege ISA provide the functions required by the basic general-purpose CPU. Developers can also enhance the processor's functionality by adding extensions to ISA. There are already many standard extensions, such as the approved MAFDGQ.

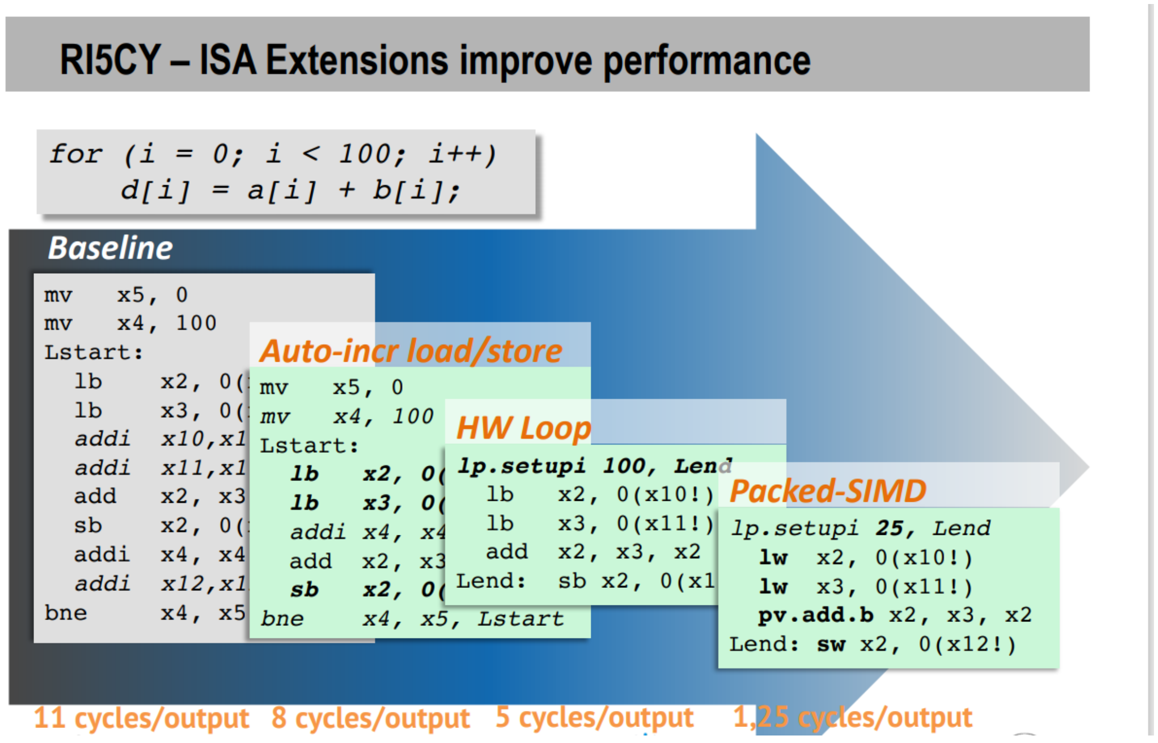
1.2. What is OpenPULP

PULP stands for Parallel Ultra-Low-Power. It is an open-source multi-core computing platform started in 2013 in collaboration between ETH Zurich and the University of Bologna.

1.2.1. PULP platform cores

Ri5cy: 4-stage pipeline, optimized for energy efficiency, it implements RV32-ICMX.

- Enhance performance.
- Reduce the code size.
- Increase the energy efficiency of signal processing algorithms (DSP).
- General Purpose extensions to Ri5cy include.
- Post–incrementing load/store instructions.
- Hardware Loops.
- ALU instructions (Bit-manipulation instructions).
- Packed-SIMD instructions.
- The core has been designed and optimized to work in a multi-core cluster.


RI5CY – ISA Extensions improve performance

```
for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];
```

```
mv    x5, 0
mv    x4, 100
Lstart:
  lb    x2, 0(
  lb    x3, 0(
  addi  x10,x1
  addi  x11,x1
  add   x2, x3
  sb    x2, 0(
  addi  x4, x4
  addi  x12,x1
bne     x4, x5
```

```
mv    x5, 0
mv    x4, 100
Lstart:
  lb    x2, 0(
  lb    x3, 0(
  addi  x4, x4
  add   x2, x3
  sb    x2, 0(
bne     x4, x5, Lstart
```

```
lp.setupi 100, Lend
  lb    x2, 0(x10!)
  lb    x3, 0(x11!)
  add   x2, x3, x2
Lend:   sb x2, 0(x1
```

```
lp.setupi 25, Lend
  lw    x2, 0(x10!)
  lw    x3, 0(x11!)
  pv.add.b x2, x3, x2
Lend: sw x2, 0(x12!)
```

Figure 1: improved performance for RI5CY

Zero-ri5cy: is 2-stage pipeline, an area-optimized RISC-V core implementing the RVC32IM for control applications, implements the HW multiplication instruction.

Micro-ri5cy: it is more optimized for area, it is possible to further reduce area by using 16 registers instead of 32, and the multiplier can be removed saving a bit more area.

1.2.2. Performance

When running DSP applications, Ri5cy is 6.1× faster than Zero-ri5cy and 53.4× faster than Microri5cy.

Coremark does not contain much DSP code, therefore the difference in performance does not come from the DSP extensions of Riscy, but mainly from its deeper pipeline that enables single-cycle data memory access, and from the single-cycle multiplication-accumulation and general-purpose instruction extensions as e.g., bit-manipulation.

In the Runtime kernel, the three cores show negligible differences in performance: the code implementing these routines do not benefit from multiplications nor from DSP extensions.

The IPC of Riscy is the highest for all the micro-benchmarks. As the PULPino data and instruction memories have 1 cycle access, the IPC is 0.8 due to misaligned memory accesses (that require two cycles), data-hazard after load operations and branches/jump.

For Zero-riscy, the IPC in the DSP micro-benchmark is only 0.5 as the most common instructions are multi-cycle operations as load, store and multiplications.

Table 2: number of cycles, IPC and code size for each KERNEL

KERNEL	**Riscy**			**Zero-riscy**			**Micro -riscy**		
	Cycles	IPC	Cod.size	Cycles	IPC	Cod. size	Cycles	IPC	Cod.size
2D-Convolution	1.0 (43.1K)	0.82	1.0 (1.80B)	6.1	0.52	1.0	53.4	0.66	1.0
Core mark	1.0 (313.5K)	0.79	1.0 (15.3KB)	1.3	0.67	1.1	3.5	0.67	1.3
Runtime	1.0 (37)	0.76	1.0 (232B)	1.0	0.68	1.1	1.0 (36)	0.67	1.1

1.2.3. Area

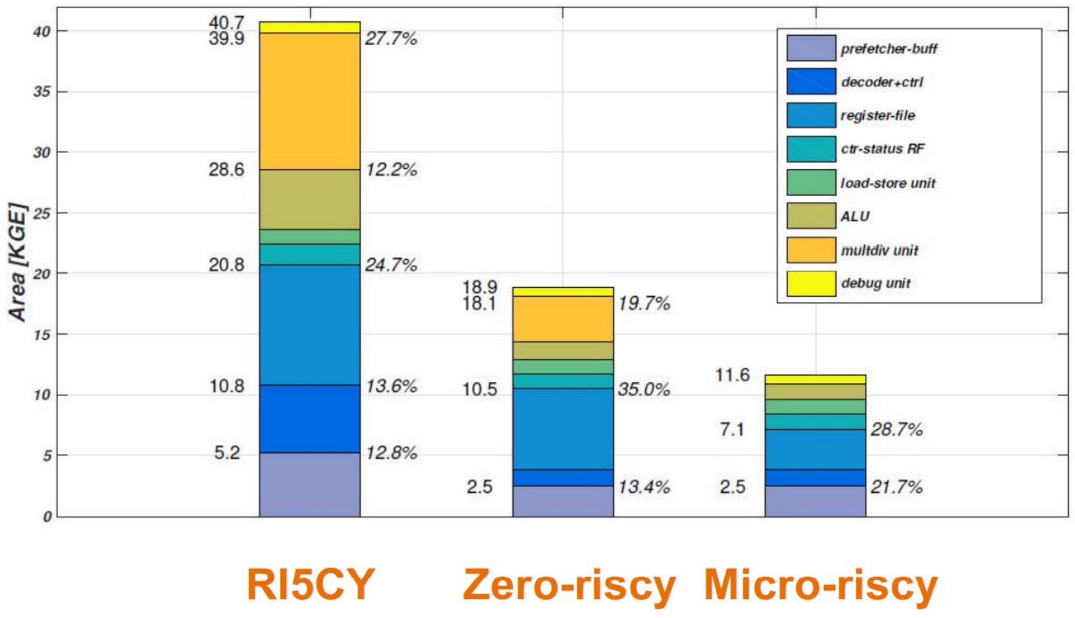


Figure 2:Area comparison

1.2.4. Power

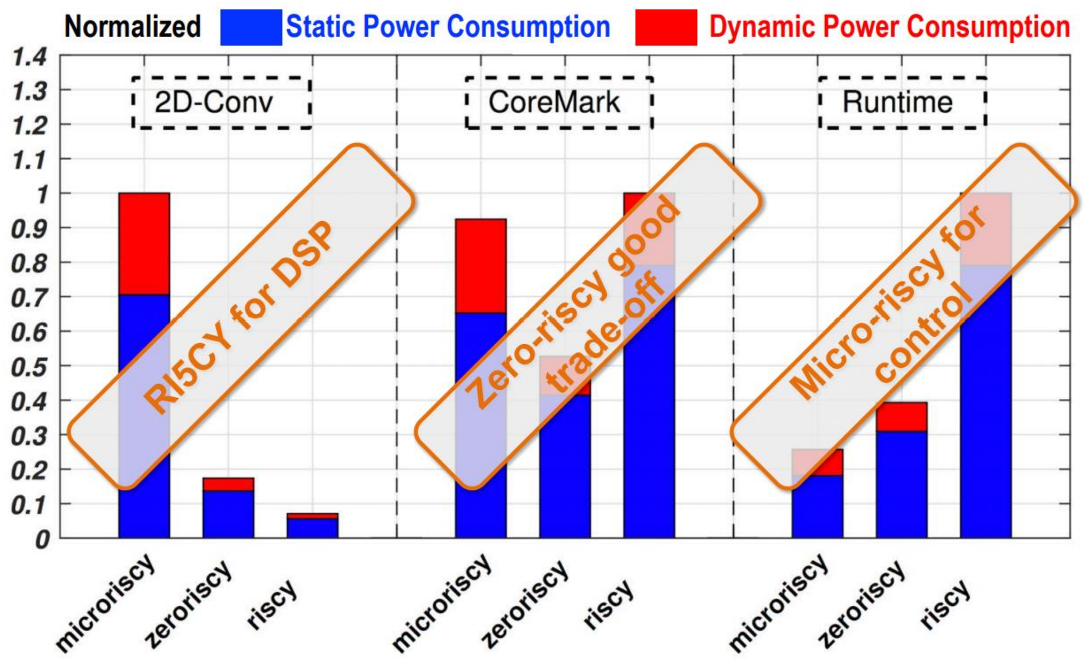


Figure 3:static and dynamic power comparison

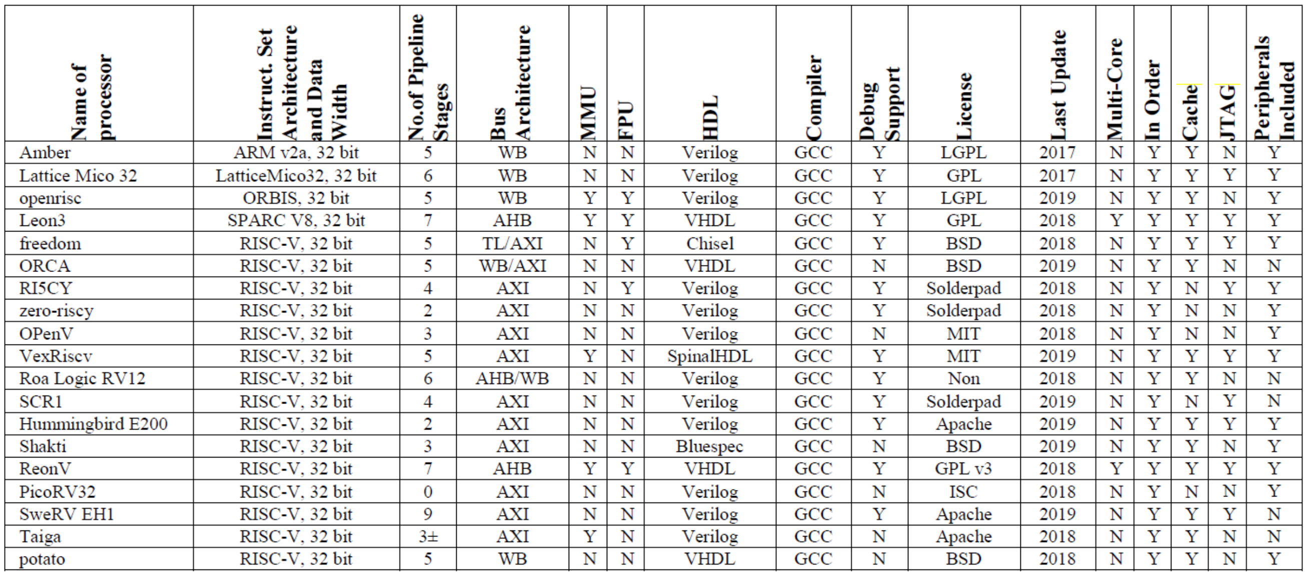
Ariane:

- supports single and double precision (F&D).
- Its area 2*area of 32-bit.
- Linux capable.

As shown from previous comparison that Ri5cy has best performance and power for DSP applications and it has less area than Ariane that is why we choose to work with it.

1.3. Why RISC-V in Open Pulp

Since RISC-V is an open-source ISA, the availability of this standard creates an opportunity to find many different suppliers, which results in both commercial and open-source cores.



Name of processor	Instruct. Set Architecture and Data Width	No.of Pipeline Stages	Bus Architecture	MMU	FPU	HDL	Compiler	Debug Support	License	Last Update	Multi-Core	In Order	Cache	JTAG	Peripherals Included
Amber	ARM v2a, 32 bit	5	WB	N	N	Verilog	GCC	Y	LGPL	2017	N	Y	Y	N	Y
Lattice Mico 32	LatticeMico32, 32 bit	6	WB	N	N	Verilog	GCC	Y	GPL	2017	N	Y	Y	Y	Y
openrisc	ORBIS, 32 bit	5	WB	Y	Y	Verilog	GCC	Y	LGPL	2019	N	Y	Y	N	Y
Leon3	SPARC V8, 32 bit	7	AHB	Y	Y	VHDL	GCC	Y	GPL	2018	Y	Y	Y	Y	Y
freedom	RISC-V, 32 bit	5	TL/AXI	N	Y	Chisel	GCC	Y	BSD	2018	N	Y	Y	Y	Y
ORCA	RISC-V, 32 bit	5	WB/AXI	N	N	VHDL	GCC	N	BSD	2019	N	Y	Y	N	N
RI5CY	RISC-V, 32 bit	4	AXI	N	Y	Verilog	GCC	Y	Solderpad	2018	N	Y	N	Y	Y
zero-riscy	RISC-V, 32 bit	2	AXI	N	N	Verilog	GCC	Y	Solderpad	2018	N	Y	N	N	Y
OPenV	RISC-V, 32 bit	3	AXI	N	N	Verilog	GCC	N	MIT	2018	N	Y	N	N	Y
VexRiscv	RISC-V, 32 bit	5	AXI	Y	N	SpinalHDL	GCC	Y	MIT	2019	N	Y	Y	Y	Y
Roa Logic RV12	RISC-V, 32 bit	6	AHB/WB	N	N	Verilog	GCC	Y	Non	2018	N	Y	Y	N	N
SCR1	RISC-V, 32 bit	4	AXI	N	N	Verilog	GCC	Y	Solderpad	2019	N	Y	N	Y	N
Hummingbird E200	RISC-V, 32 bit	2	AXI	N	N	Verilog	GCC	Y	Apache	2019	N	Y	Y	Y	Y
Shakti	RISC-V, 32 bit	3	AXI	N	N	Bluespec	GCC	N	BSD	2019	N	Y	Y	N	Y
ReonV	RISC-V, 32 bit	7	AHB	Y	Y	VHDL	GCC	Y	GPL v3	2018	Y	Y	Y	Y	Y
PicoRV32	RISC-V, 32 bit	0	AXI	N	N	Verilog	GCC	N	ISC	2018	N	Y	N	N	Y
SweRV EH1	RISC-V, 32 bit	9	AXI	N	N	Verilog	GCC	Y	Apache	2019	N	Y	Y	Y	N
Taiga	RISC-V, 32 bit	3±	AXI	Y	N	Verilog	GCC	N	Apache	2018	N	Y	Y	N	N
potato	RISC-V, 32 bit	5	WB	N	N	VHDL	GCC	N	BSD	2018	N	Y	Y	N	Y

Figure 4: example for different open-source RISC-V suppliers

Out of the many open-source cores in the market, OpenHW or pulp-platfrom has been gaining much attention, in certain literature reviews and core open-source websites due to their efficient RTL, their low resource utilization, low power consumption and good performance.

In addition to the fact the OpenPULP provides peripherals, microcontrollers, cluster and accelerators that are built on the RISC-V cores that they provide.

1.4. Thesis objective

The RTL to GDSII Flow is critical in the creation of electrical chips all over the world. Due to intense rivalry and high client demands, technology in this industry is rapidly evolving toward smaller, quicker, and more complicated gadgets, making it difficult for a beginner to keep up without first studying and comprehending the fundamentals. More than one ASIC design style is currently available in CAD tools. Satisfy large customer expectations while improving ASIC designs' QoR and TTR.

The goal of this thesis is to show how different design flows affect an OpenPULP Core (RI5CY) open-source RTL design utilizing 40nm CMOS technology.

This goal is accomplished by following the RTL-to-GDSII path from start to finish. There are three types of logic synthesis flows: hierarchical, flat, and topographical. Following that, the relevant Place and Route steps are performed and STA post-layout to provide good timing performance across a variety of operating environments conditions.

1.5. Thesis Map

At First, we will see the OpenPULP and its architecture. Then about the main blocks in this architecture. Second, we will discuss the ASIC flow and its stages in the next chapter. Then We will know the Flat Flow, the Hierarchical Flow and the Topographical flow and their flow in each stage of the ASIC flow.

The second part of the thesis is to see the results out of each flow and get the best one which can work on the maximum speed. The conclusion views the comparison between each flow according to timing, power and area.

The third part is the appendix part that discusses the constrain file, the clock gate we used and the meaning of the power consumption, DRCs and LVS. Also, it contains the scripts we run in each flow.

Chapter 2

OpenPULP

OpenPULP Core or RI5CY is a 4-stage in-order 32b RISC-V processor core. The ISA of RI5CY was extended to support multiple additional instructions including hardware loops, post-increment load and store instructions and additional ALU instructions that are not part of the standard RISC-V ISA. RI5CY supports the following instructions: RV32IMC or RV32IMFCXpulp.

2.1. OpenPULP architecture

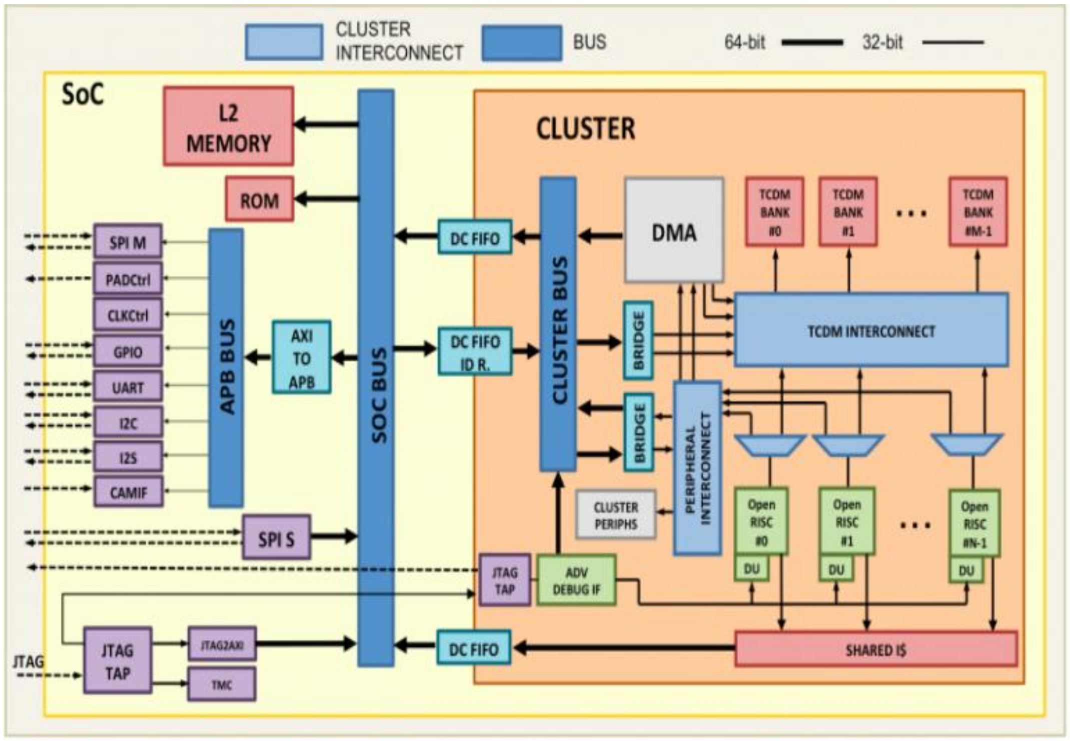


Figure 5: The PULP overview

We are going to focus on the open RISC block which is repeated in green color at the bottom right.

2.2. 32-bit RISC-V in openPULP

This architecture is called RI5Cy, the open-source RTL design core is called CV32E40P.

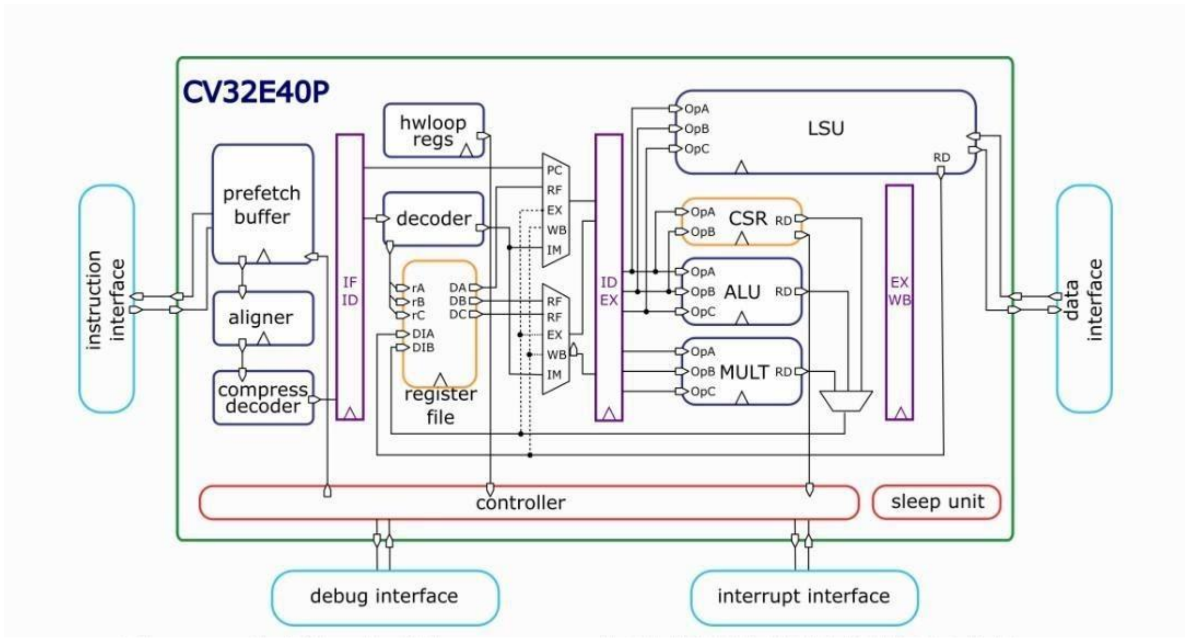


Figure 6:the block diagram of the CV32E40P

2.2.1. RI5CY ISA:

RISC-V is an ISA: it is a well-organized ISA divided into categories and extensions

RI5CY supports the following instructions:

Full support for RV32I Base Integer Instruction Set: We have 32 Registers from (0x to 31x) which means 1 word, but 0x handwritten with 0 value so we have 31 bits to write any data we need.

Ex: add rd, rs1, rs2.

- Full support for RV32C Standard Extension for Compressed Instructions
- Full support for RV32M Integer Multiplication and Division Instruction Set Extension
- Optional full support for RV32F Single Precision Floating Point Extensions
- PULP specific extensions
- Post-Incrementing load and stores
- Multiply-Accumulate extensions
- ALU extensions
- Hardware Loops
- Optional Floating-Point Support

Floating-point support in the form of IEEE-754 single precision can be enabled by setting the parameter FPU of the top-level file "riscv_core" to one. This will instantiate the FPU in the execution stage, and also extends the register file to host floating-point operands and extend the ALU to support the floating-point comparisons and classifications.

We are going to talk about each block in Figure 6.

2.2.1.1. Instruction fetch

The instruction fetcher of the core is able to supply one instruction to the ID stage per cycle, if the instruction cache or the instruction memory is able to serve one instruction per cycle. If the external bus interface can serve one instruction every cycle, the CV32E40P's Instruction Fetch (IF) stage can deliver one instruction to the Instruction Decode (ID) stage per cycle. When executing compressed instructions, the ID stage will require on average less than one 32-bit instruction fetch per instruction.

The instruction address must be half-word-aligned due to the support of compressed instructions. It is not possible to jump to instruction addresses that have the LSB bit set.

A prefetcher is utilized for optimal performance and timing closure, fetching instructions from an externally connected instruction memory or instruction cache via the external bus interface. The prefetch unit executes 32-bit word-aligned prefetches and saves the fetched words in a four-entry FIFO.

CV32E40P can retrieve up to four words outside of the code region as a result of this (speculative) prefetch, and care should be made to ensure that no undesirable read side effects occur for such prefetches outside of the real code region. The signals that are utilized to fetch instructions are described in Table 2-1 below. This interface is a condensed version of the Load-Store-Unit (LSU) interface described in Load-Store-Unit (LSU). The difference is that no writes are permitted, hence fewer signals are required.

Table 3: fetch signals

Signal	Direction	Description
Instr_req_o	output	Request ready, must stay high until instr_gnt_i is high for one cycle
Instr_addr_o[31:0]	output	Address
Instr_rdata_i[31:0]	input	Data read from memory
Instr_rvalid_i	input	Instr_rdata_is holds valid data when instr_rvalid_i is high. This signal will be high for exactly one cycle per request.
Instr_gnt_i	input	The other side accepted the request. instr_addr_o may change in the next cycle

There are two prefetch flavors available:

- 32-Bit word prefetcher. It stores the fetched words in a FIFO with three entries.
- 128-Bit cache line prefetch. It stores one 128-bit wide cache line plus 32-bit to allow for cross-cache line misaligned instructions.

Misaligned Access:

Only word-aligned instruction fetches are performed by the IF interface from the outside. Two distinct word-aligned instruction fetches are used to handle misaligned instruction fetches. To handle compressed instructions, the core can deal with both word- and half-word-aligned instruction addresses internally. Internally, the instruction address's LSB is ignored.

Protocol:

The OBI (Open Bus Interface) protocol is used for the instruction bus interface. The optional OBI signals we, be, wdata, auser, wuser, aid, rready, err, ruser, and rid are not implemented by the CV32E40P instruction fetch interface. These signals can be regarded as being linked off in the way that the OBI specification specifies. Up to two pending transactions can be caused through the CV32E40P instruction fetch interface. The protocol used to communicate with the instruction cache or the instruction memory is the same as the protocol used by the LSU.

2.2.1.2. Load store unit (LSU)

The core's Load-Store Unit (LSU) is in charge of accessing the data memory. Words (32 bit), half words (16 bit), and bytes (8 bit) can all be loaded and stored.

Table 4 describes the signals that are used by the LSU.

Table 4: LSU signals

Signal	Direction	Description
data_req_o	output	Request ready. Must stay high until data_gnt_i is high for one cycle
data_addr_o[31:0]	output	Address
data_we_o	output	Write enable, high for writes, low for reads. Sent together with data_req_o
data_be_o[3:0]	output	Byte enable. Is set for the bytes to write/read. Sent together with data_req_o
data_wdata_o[31:0]	output	Data to be written to memory. Sent together with data_req_o
data_rdata_i[31:0]	input	Data read from memory
data_rvalid_i	input	data_rdata_is holds valid data when data_rvalid_i is high. This signal will be high for exactly one cycle per request.
data_gnt_i	input	The other side accepted the rquest. data_addr_o may change in the next cycle

Misaligned Access:

Exceptions for address misalignment are never raised by the LSU, when the effective address is not naturally aligned with the referenced data type during loads and stores. If the data item exceeds a word boundary, the load/store is done as two bus transactions (i.e., on a four-byte border for word accesses and a two-byte boundary for half word accesses). For the following cases, a single load/store operation is split into two bus transactions:

- Load/store of a word for a non-word-aligned address.
- Load/store of a half word crossing a word address boundary.

The transfer corresponding to the lowest address is completed first in both circumstances. A single bus transaction can handle all other circumstances.

Protocol:

The OBI (Open Bus Interface) protocol is used for the data bus interface. The optional OBI signals auser, wuser, aid, rready, err, ruser, and rid are not implemented by the CV32E40P data interface. These signals can be regarded as being linked off in the way that the OBI specification specifies. Up to two pending transactions can be caused by the CV32E40P data interface.

The OBI protocol, which the LSU uses to interface with a memory, operates like this:

The LSU sets data_req_o to high and supplies a valid address in data_addr_o. As soon as the RAM is ready to handle the request, it responds with a data_gnt_i set high. This may happen right after the request was received, or it could happen a few cycles later. The LSU may change the address when a grant is received in the next round. Furthermore, because it is assumed that the memory has already processed and stored that information, the data_wdata_o, data_we_o, and data_be_o signals may be changed. If data_rdata_i is valid, the memory responds with a data_rvalid_i set high after getting a grant. This could happen after the grant has been received for one or more rounds. When a write is conducted, data_rvalid_i must also be set, even if data_rdata_i has no meaning in this case.

Post-Incrementing Load and Store Instructions:

The load and store instructions that are post-incrementing perform a load/store operation from/to the data memory while also increasing the base address by the provided offset. The base address without offset is used for memory access. In order to run code with regular data access patterns, such as those seen in loops, post-incrementing load and stores reduce the number of instructions necessary. These post-incrementing load/store instructions integrate the address increment into the memory access instructions, eliminating the need for separate pointer handling instructions. These instructions, when combined with hardware loop extension, allow for a significant reduction in loop overhead.

2.2.1.3. Physical Memory Protection (PMP) Unit

The RI5CY core has a PMP module which can be enabled by setting the parameter PULP SECURE=1 which also enables the core to possibly run in USER MODE. Such a unit has a configurable number of entries (up to 16) and supports all the modes as TOR, NAPOT and NA4. Every fetch, load and store access executed in USER MODE are first filtered by the PMP unit which can possibly generate exceptions. For the moment, the MPRV bit in MSTATUS as well as the LOCK mechanism in the PMP is not supported.

2.2.1.4. Optional private floating-point unit (FPU)

It is possible to extend the core with a private FPU, which is capable of performing all RISC-V floating-point operations that are defined in the RV32F ISA extensions.

FP extensions can be enabled by setting the parameter of the top-level file "riscv_core.sv" to one.

The FPU is divided into three parts:

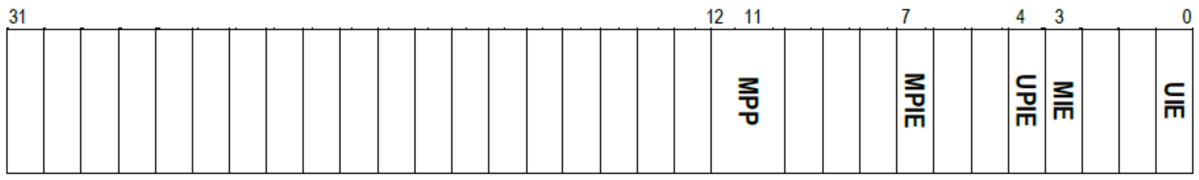
- A simple FPU of ~10kGE complexity, which computes FP-ADD, FP-SUB and FP-casts.
- An iterative FP-DIV/SQRT unit of ~7 kGE complexity, which computes FP-DIV/SQRT operations.
- An FP-FMA unit which takes care of all fused operations (currently only supported through a Synopsys Design Ware instantiation, or a Xilinx block for FPGA targets).

2.2.1.5. Control and status register (CSR)

- RI5CY implements only limited registers needed for the PULP system to avoid any overhead regardless of that specified in the RISC-V privileged specs.
- CSR Address 12-bit,22 registers
- All zeros address is for user status

Let's focus on some of them:

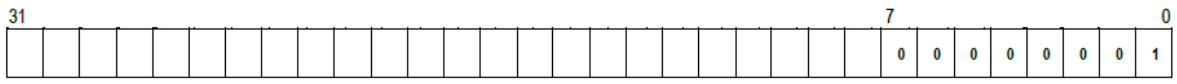
MSTATUS:



For exceptions.

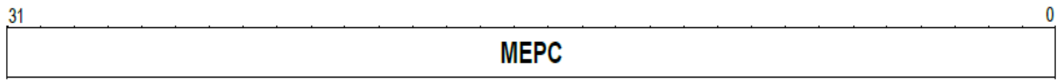
- Reset value at MPP (machine previous privileged mode) = 11, User mode is disabled
- MPIE: previous machine interrupt enables, when an exception occurs, it is set to MIE until MRET instruction is executed (machine mode trap handler return), MPIE will be stored in MIE
- UPIE: Previous user interrupt enable, same case but user mode, pulp doesn't support user interrupt
- MIE: interrupt enable handling is '1' in handler code

MTVEC:



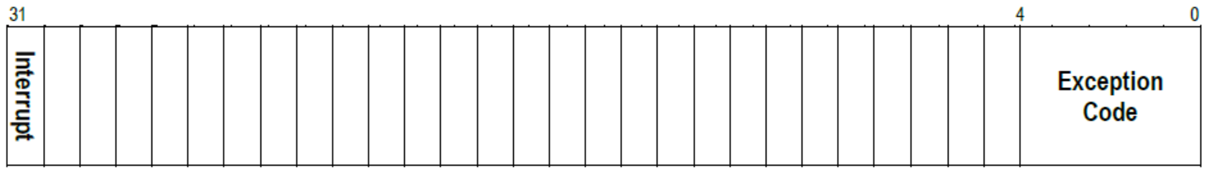
- 01 for vectorized interrupt mode supported.
- When exception occurs, the core jumps to the corresponding handler at address (31:8) as the base address. 8-Byte aligned address only is allowed.

MEPC:



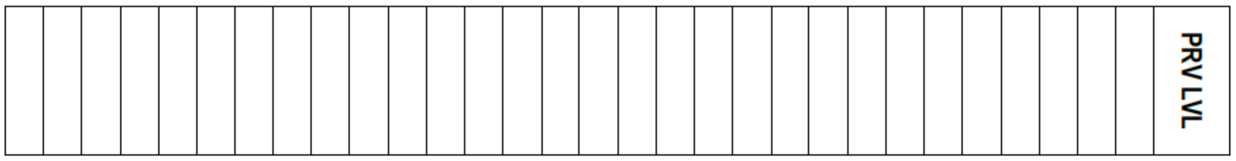
When exception occurs, it saves the current PC and jumps to exception address, when mret instruction is executed, it goes back to the saved PC.

MCAUSE:



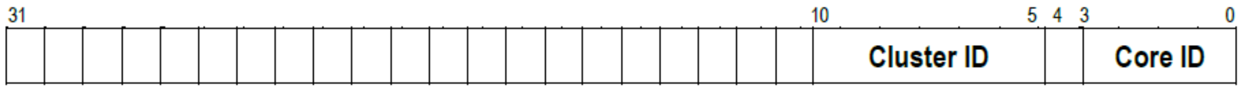
Interrupt is set when the exception was triggered by interrupt.

PRIVLIV:



It contains the current privilege level the core is executing, provide a mechanism to allow portions of the software to operate with differing levels of privilege. The current privilege level is used by the system to control access to resources and execution of certain instructions.

MHARTID/UHARTID

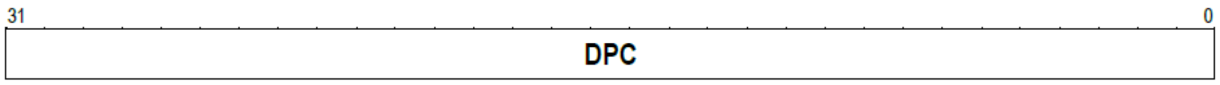


ID of the cluster and the core inside this cluster.

DCSR:

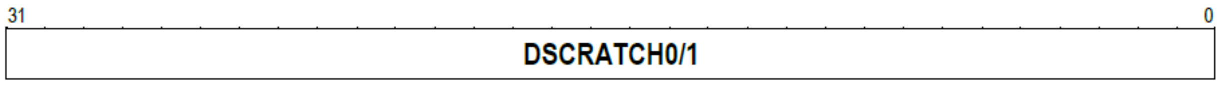
Supports external debugging.

DPC:



Contains the virtual address of the instruction to be executed when the core enters debug mode.

DSCRATCH0/1:

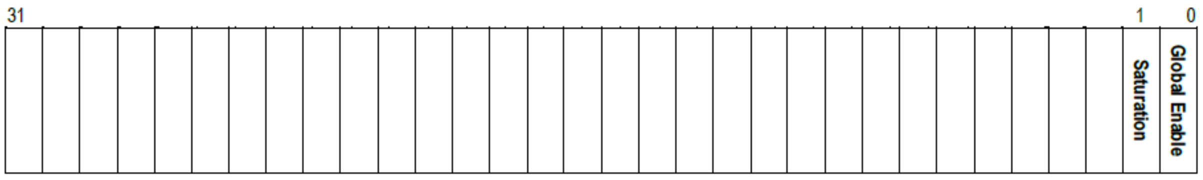


Scratch registers that can be used by implementations that need it. (Temporary registers needed in intermediate stages)

2.2.1.6. Performance counters

Placed inside CSR block and accessed by csrr and csrw instructions.

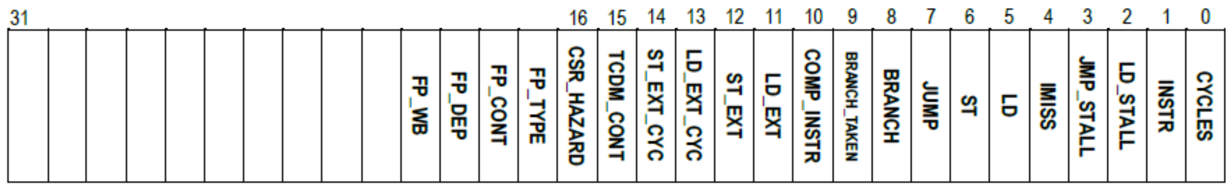
PCMR:



Global enable: activate or deactivates all performance counters.

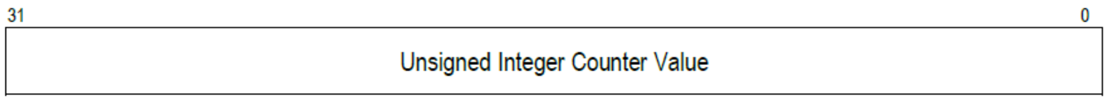
Saturation: enables saturating arithmetic in performance counters.

PCER:



- Each bit controls 1 performance counter enabled by '1'.
- In ASIC there is only one counter, the counting events are masked by PCER and ORed together, if an event is enabling the counter increase and if there are multi unmasked events the counter increases only by one, to count multi event separately the program should be executed inside a loop with different events configured.
- In FPGA or RTL each event has its own counter and works separately.

PCCR



All these registers are accessed separately in FPGA, to save area in ASIC only PCCR31 is used which is equivalent with using PCCR0:30 in FPGA

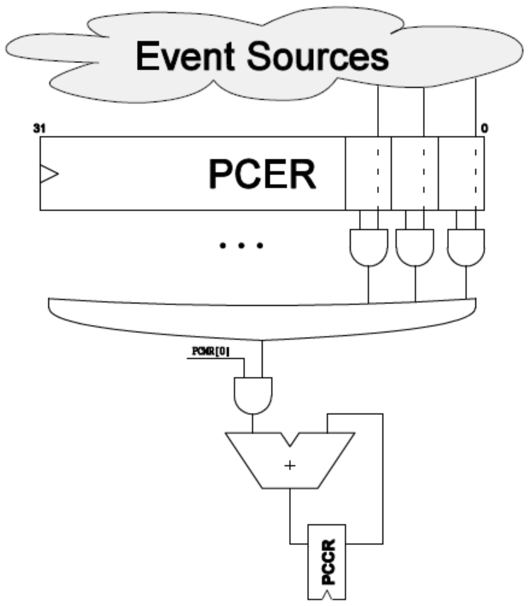


Figure 7: Event of masking with PCER and ORing together to increase one performance counters PCCR

Exceptions and interrupts

- RI5KY supports only interrupts of illegal instructions and PMP filtered requests if enabled on the data and instruction bus.
- Base address of interrupt vector table is given by MTVEC address.
- It supports vectorized interrupts only, interrupt 0 is reserved for illegal instructions, ecall, and instructions or data prohibited access.

Interrupts:

- Interrupts can only be enabled in a global basis from MSTATUS
- It is assumed that there is an event/interrupt controller outside the core that performs masking and buffering of interrupt lines.
- The global interrupt enable is done via CSR register MSTATUS
- Multiple interrupts/events are assumed to be handled by interrupt controller. When interrupt is taken, the core acknowledges it to the controller as well as the interrupt id is taken

Exceptions:

- Illegal instruction and ecall exceptions cannot be disabled and always active.
- For PMP exception, if enabled all data and instructions requests are filtered by PMP which cause load store and fetch exceptions

Handling:

- RI5KY supports SW-assisted nested exception/interrupt handling: the SW has to take care to save the MSTATUS and MEPC before any possible nested exceptions or interrupt.
- Interrupts are disabled during handling but can be explicitly enabled

Debug

- RI5KY supports the RISC-V debug specification 0.13 and it implements the execution based to reuse the existing core pipeline
- RI5KY has a debug_req_i input port that is sent by system debug module. Such request makes the core jumps to the specific address where the debug Rom is mapped.
- This address location is referred to the parameter DM_HALTAddress.
- RI5KY implements the debug sets of registers as DPC, DCSR and DSCRATCH0,1

2.2.1.7. Pipeline

RISCY has a fully independent pipeline containing four stages.

Each pipeline stage has two control inputs:

- **An enable:** activates the pipeline stage and the core moves forward by one instruction.
- **A clear:** clear removes the instruction from the pipeline stage as it is completed.

Every pipeline stage is cleared if the ready coming from the stage to the right is high, and the valid signal of the stage is low.

If the valid signal is high, it is enabled.

Every pipeline stage is independent of its left neighbor, meaning that it can finish its execution no matter if a stage to its left is currently stalled or not. On the other hand, an instruction can only propagate to the next stage if the stage to its right is ready to receive a new instruction. This means that in order to process an instruction in a stage, its own stage needs to be ready and so does its right neighbor.

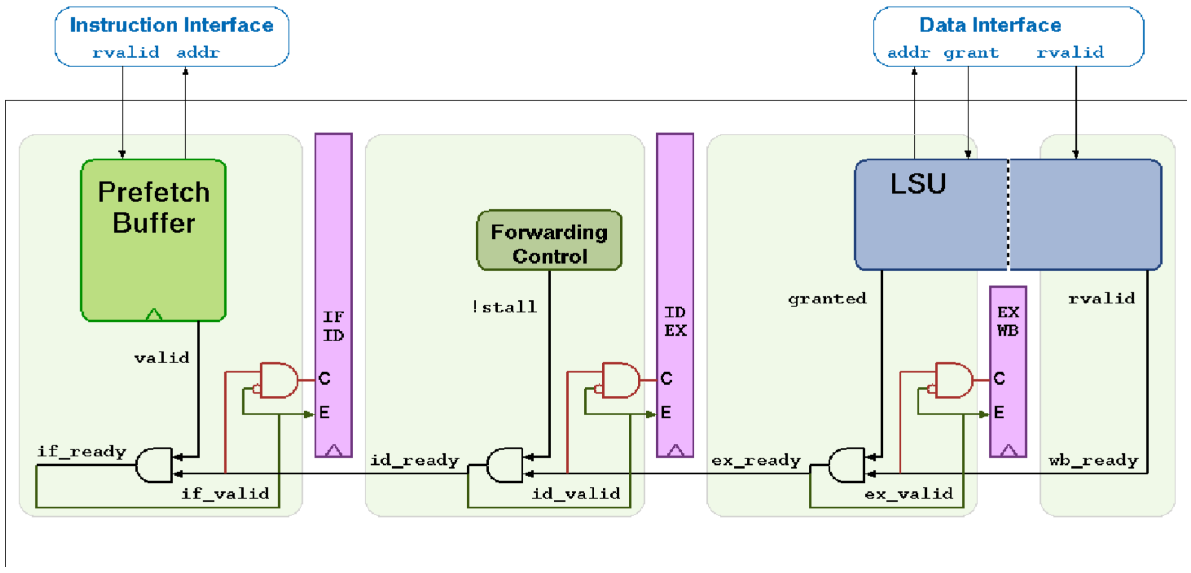


Figure 8: using pipeline

2.2.1.8. PULP Hardware Loop Extensions

To increase the efficiency of small loops, RI5CY supports hardware loops. Hardware loops make it possible to execute a piece of code multiple times, without the overhead of branches or updating a counter.

- Hardware loops involve zero stall cycles for jumping to the first instruction of a loop.
- A hardware loop is defined by its start address (pointing to the first instruction in the loop), its end address (pointing to the instruction that will be executed last in the loop) and a counter that is decremented every time the loop body is executed.

- RI5CY contains two hardware loop register sets to support nested hardware loops, each of them can store these three values in separate flip flops.

2.2.1.9. Register file

There are two flavours of register file available:

a. Latch-based
b. Flip-flop based

- While the latch-based register file is recommended for ASICs, the flip-flop-based register file is recommended for FPGA synthesis.
- Although both are compatible with either synthesis target. Note the flip-flop-based register file is significantly larger than the latch-based register-file for an ASIC implementation.

In case the optional FPU is instantiated, the register file is extended with an additional register bank of 32 registers f0-f31.

2.2.1.10.Instruction set extensions

Post incrementing load and store:

- Perform load and store, besides, it increments the address used for memory access.
- Post increment, the base address is used for the access and the modified address is written back in register file.
- There are versions of instructions that use immediate and registers as offsets.
- The base address always comes from a register.

Hardware loops:

- RI5KY supports 2 levels of nested HW loops.
- The loop has to be setup before entering the loop body.
- There are 2 methods for setup, long commands separately set start, end and number of iterations, or a short command that does all these instruction in a single one.
- Short command has a limited number of instructions in the loop and the loop must start in the next instruction after set up.
- Loop number 0 has a higher priority than loop number 1 in a nested loop configuration, that means loop zero is the inner loop.
- A hardware loop is subjected to the following constraints: minimum of 2 instructions in the body loop, and the loop counter must be greater than 0 as the loop is always entered at least once.

ALU:

- RI5CY supports advanced ALU operations that allow performing multiple instructions that are specified in the base instruction set in one single instruction and thus increases efficiency of the core.
- Examples:
- min/max/avg instructions.
- zero-/sign-extension instructions for 8-bit and 16-bit operands.
- simple bit manipulation/counting instructions.

Multiply-Accumulate:

It is a common step especially in digital signal processing computations; RI5Ky supports its extensions to minimize the number of instructions needed for its implementation.

Vectorial:

Vectorial instructions perform operations in a SIMD-like manner (single instruction multiple data) on multiple sub-word elements at the same time. This is done by segmenting the data path into smaller parts when 8 or 16-bit operations should be performed.

They are available in two flavors:

- 8-bit, to perform operations on 4 bytes inside a 32-bit word at the same time.
- 16bit, to perform operations on 2 bytes inside a 32-bit word at the same time.

Chapter 3

ASIC Flow

ASIC stands for application specific integrated circuit; it is used in digital design to model system on chips with high performance.

There are two types of ASICs which are full custom and standard cell based.

3.1. ASIC Vs FPGA

There are differences between ASIC and FPGA. Both of them are used in industry and each of them has pros and cons.

Table 5: comparison between ASIC and FPGA

ASIC	FPGA
Fabricated circuit only one time and in a Fab	can be programmed many times
High cost	Moderate cost
For mass production to reduce the cost	Small and fast production
Needs long time to market	Programmed in the field
Better utilization as it uses the cells needed only	Waste for resources and less utilization because of fixed number of programmable cells
Higher performance but more complex	Less performance but more flexible

3.2. ASIC design flow

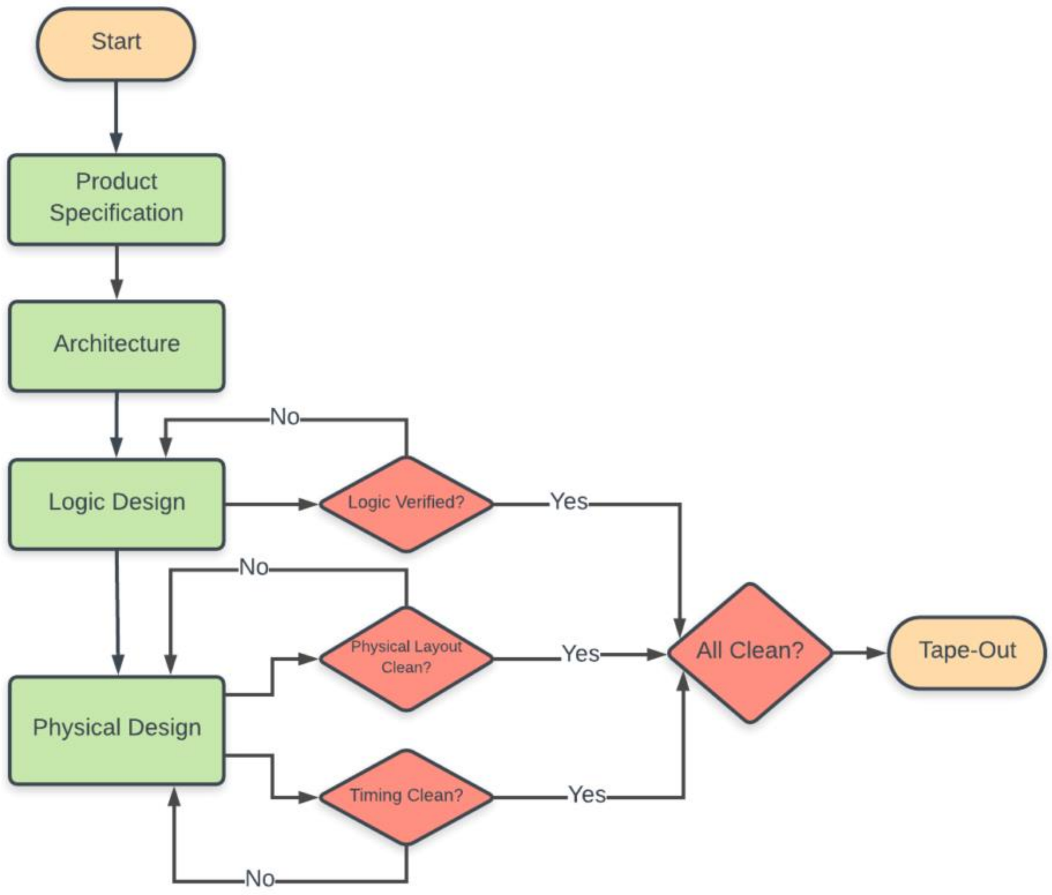


Figure 9: Design flow

In Figure 9 we can find the design flow starting from high level customer needs ending with Tape-out of circuit given to fab for silicon chip production.

In our project, we focused on the synthesis and physical design phase. We worked on the open-source logic design of 32-bit RISC-V in openPULP platform.

We performed synthesis, place and route (PNR), RC extraction and static timing analysis (STA).

3.2.1. Synthesis

Synthesis is a step in the flow in which we convert the RTL along with physical libraries to netlist. Design compiler tool is used.

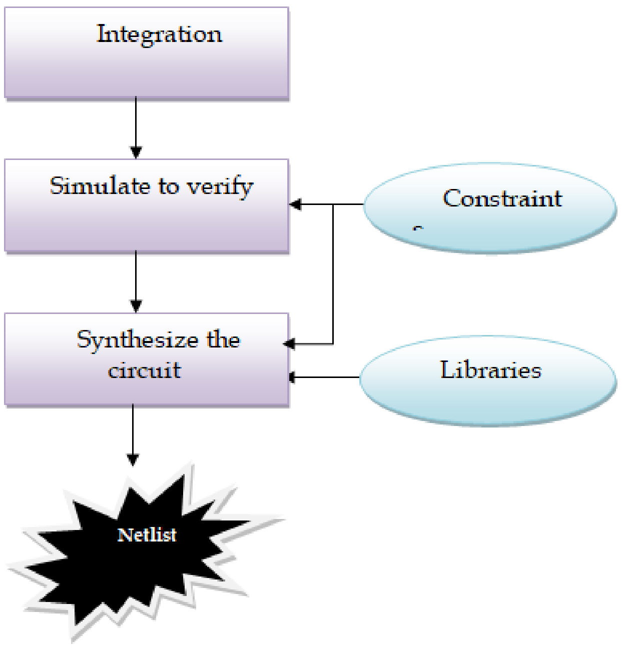


Figure 10: front-end flow

Integration: description of the design as RTL with HDL such as: VHDL, Verilog and system verilog.

Simulation to verify: simulate the RTL using simulation EDAs such as: Modelsim and Xilinix. Synthesis the circuit: for ASIC, we use synthesis tools such as Design compiler which converts RTL to Logic gates netlist.

Synthesis is done in steps:

- **Step 1:** converts HDL to generic logic gates called GTECH stands for generic technology; this is built in synthesis tool before using technology library given to the tool specified by the fab.
- **Step 2:** reading the technology library to map these generic gates to technology-based gates which are called standard cell library for standard cell-based ASIC design. These standard cells are the layout for basic logic gates.
- **Step 3:** optimization according to the constraints given to the tool.

Netlist: it is the output of synthesis tool; it doesn't include wiring or arrangement of cells. It can model timing according to gates only not wiring delay. It represents logic gates in text form given to the PNR tool.

3.2.1.1. Basic synthesis flow:

In order:

- Define libraries.
- Reading design.
- Reading constraints.
- Check design.
- Compile / Technology mapping & optimization.
- Generate gate level netlist.
- Generate reports

We will clarify each step in the following section:

Define libraries:

Standard cells or technology libraries

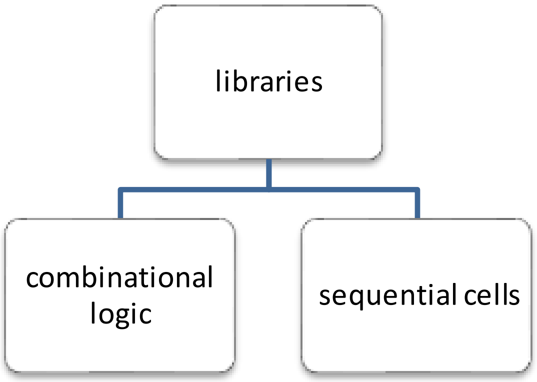


Figure 11: libraries

- **Combinational logic**: basic logic cells only such as: NOT, AND, OR, NAND, NOR, buffers and half adders.
- **Sequential cells**: flip flops and latches; positive and negative edge FF, synchronous and asynchronous reset/set, ATPG cells used in DFT flow and clock gating cells.

Clock gating cells are modeled in Verilog and mapped to technology directly; it has problems inside the cells due to wiring as it doesn't maintain the clock square wave form as expected. Hence internal clock gating is added in libraries and is instantiated directly in synthesis the same as the one used in simulations.

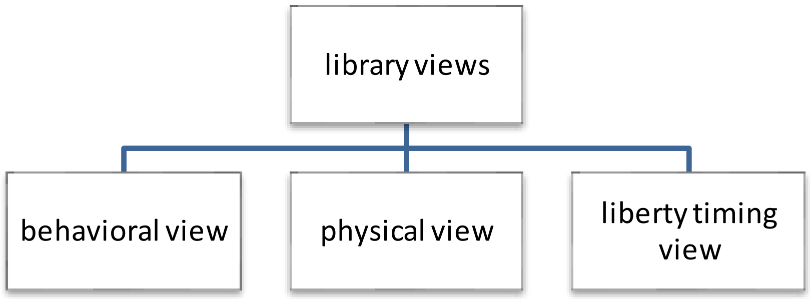


Figure 12: library views

These libraries are given by the fab before design.

- **Behavioral view:** Vital simulation and gate level simulation (GLS). GLS is the netlist simulation which contains the names of cells inside netlist including the reference cells used in netlist. It is not used in synthesis phase.
- **Physical view:** (.gds) and (.lef) files. Representing the layout and it is not used in synthesis but used in PNR.
- (.gds) file: includes the detailed information about the layout of cells such as types of metals and so on.
- (.lef) file: includes abstract information about cells as a black box which are needed in PNR and RC extraction for delay information. It contains pin locations and physical cell sizes.
- **Liberty timing view:** (.lib) file which is measured on a specific PVT. It is used as an input to synthesis tool with timing, area, power information, propagation delay and transition time of the output.

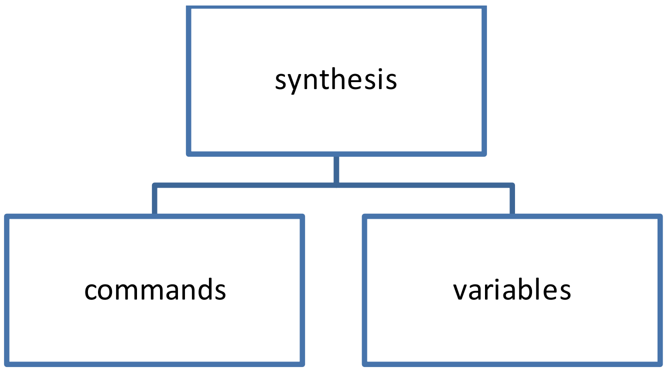


Figure 13: synthesis script

Variables:

- Setting specific variables and continue the synthesis flow with them
- **target_library:** taking timing libraries in (db) format which is the compiled version of (.lib). This (db) format is given by the fab or converted by the tool from (.lib). The tool understands that it uses this library in mapping stage and it can be more than one library.
- **link_library:** responsible for linking all designs together. After converting designs to gates, it links the gates together with the design files. Hard macros are analog blocks; they are put in link libraries. They are linked with the top module as analog blocks are not synthesized.
- **Synthetic_library:** it is defined by the tool automatically no need for definition of the designer. It contains the operators used in HDL such as: adders, shifters and so on.

Reading design files

There are two methods:

- **Analyze/elaborate**

analyze: reads HDL files. It informs the designer if there is any syntax error or non-synthesizable block.

elaborate: takes (.syn) if it passed the analyze command and translates the HDL to GTECH (technology independent cells). Also, it replaces operators and performs link command automatically. Also, we can put link command after it.

read_file

Makes analyze and elaborate in a single step, but it doesn't allow me to get intermediate files or change parameters. Link command must follow read_file command.

Linking files:

Linking instances of cells with their definitions in library after mapping.

If there any hard macros, it will also be linked.

Check design

It is done before compiling therefore before mapping. We should check there that there are no errors in this stage and if there, we should fix these errors.

Constraints

- There are four types for paths:
- Input port to register: needs when the input will be available.
- Register to register: it won't make problems if is not defined, it needs clock period only.
- Register to output: needs set input and output delays.
- Input to output: needs set input and output delays.
- All these constraints must be set in the design.

Compile

Maps GTECH gates to technology gates and optimizes delay. After that, it adjusts the area to the minimum acceptable at the minimum delay.

Generate netlist

- (.ddc): encrypted, can be used in stages after synthesis.
- (.v) or (.vhd): can be read and used in next stages after synthesis.

Reports

For timing, area and power

Timing reports:

Setup analysis: should be met in synthesis and is adjusted by design or constrains.

Hold analysis: adjusted in PNR flow.

3.2.2. PnR

Place and route is the back-end flow in ASIC physical design. Here, we read the netlist generated from synthesis and constraints to arrange the generated cells, manage, fix timing and route them for physical system on chip.

At the start of PnR flow, we define the PDK that we use, and the TLU+ library which is used to estimate the parasitic, hence estimating the delays. IC Compiler tool is used.

3.2.2.1 Basic PnR flow

- Floor planning
- Power planning
- Placement
- Clock tree network
- Routing
- Finishing

Setup or design preparation at the start of PnR flow:

This section will explain in detail the commands and steps used in order to setup the design in order to proceed in the PnR steps correctly.

- First, we make sure that our synthesized design has been set using:

set design cv32e40p_core.

- After setting our design, we set our desired library path using

set_app_var search_path [path directory] .

This is followed by the commands that choose the link and target library as well using,

set_app_var link_library [link library path]

set_app_var target_library [target library]

Both the link and target library contain the library which contains the cells referenced by the net list and the cells used for optimization and mapping respectively.

- The next step in the workflow is creating a design library. A design library contains all the information about the design that change during the PNR process as well as the technology files used and reference libraries. Creating a library is very helpful during the PNR process as it helps create check points that enables returning to them when an error or failure occurs so that we don't need to start the design from the beginning which.

A high-level abstraction of a design library as shown in the figure below:

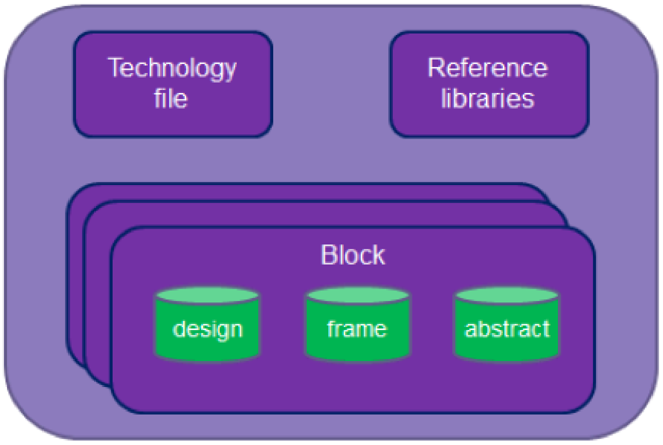


Figure 14: high level view of the design library

In order to create the library, the following command is used:

$$\boldsymbol{create\_mw\_lib}\ \ ./\$\{\boldsymbol{design}\}\ \backslash$$

$$-\boldsymbol{technology}\ \$\boldsymbol{sc\_dir/tech/techfile/milkyway/FreePDK45\_10m.tf}\ \backslash$$

$$-\boldsymbol{mw\_reference\_library}\ \$\boldsymbol{sc\_dir/lib/Back\_End/mdb}\ \backslash$$

$$-\boldsymbol{open}$$

Both the technology files as well as the reference libraries were specified as option in the creation of the design library.

- Following the process of creating a design library, we set information useful for the completion of the PNR, which is are the TLU plus max and min that contain the conditions for calculating resistance and capacitances used in extraction, and the tech2itf file which is used in mapping the names between the technology file and interconnect technology format.

The commands used are:

```
set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"

set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"

set tech2itf $sc_dir/tech/rcxt/FreePDK45_10m.map
```

After setting the paths to the variable, the variable is then used in the next command:

```
set_tlu_plus_files −max_tluplus $tlupmax \
    −min_tluplus $tlupmin \
    −tech2itf_map $tech2itf
```

- The output of the synthesis step is then imported using:

```
import_designs ../syn/output/${design}.v \
    −format verilog \
    −top ${design} \
    −cel ${design}
```

Where the option specifies the format of the HDL files used, as well as the name of the top module and cell in which it’s saved.

- Lastly, we source the same constraints that were used in the synthesis step, which contain all the timing information, and we set the propagated clock in order not to use the default ideal clock.

$$\boldsymbol{source\ ../constraints/[file.sdc]}$$

$$\boldsymbol{set\_propagated\_clock\ [get\_clocks\ [clock\ name]]}$$

We will clarify the basic information in each PnR flow step:

3.2.2.1. Floor planning

The first stage of the PnR flow is the floor planning, floor planning formalizes and refines the floor plan specified for the design. At this stage, we define the area of the chip, and the core area where the cells will be placed. Also in this stage, the macros are placed. Also, Top level, utilization, aspect radio and input/output to core distance are specified here.

Pin and ports are also assigned at rough locations and further can be redefined during the PnR flow.

Inputs for floor plan:

- Netlist (.v)
- Technology file (techlef)
- Timing Library files (.lib)
- Physical library (.lef)
- Synopsys design constraints (.sdc)
- Tlu+

Depending on the utilization value, the tool calculates the core area of the chip based on the relation Utilization = (∑Cell Area)/(Core Area) .Utilization is the percentage of core area that is taken up by standard cells. Increasing the utilization decreases the area of the chip which is good, but it also makes the design more congested. High congestion means many DRCs and makes it harder for the tool to route the cells. Decreasing the utilization increases the area, but makes the routing easier. Utilization value also affects the timing. Increasing it makes the tool trapped between solving DRCs with timing violations or meeting timing with DRCs violations. Also, decreasing it makes the routes between the cells longer as cells will be placed far apart, and this increases the routes' parasitic, hence increases the delay. Therefore, determining the utilization value is a critical decision and has to be made carefully.

We also set the ignored layers in this stage. Ignored layers are the layers at which cells are not placed and routed because we use them for the power grid. We preferred to separate the signal routes from the power routes in order to reduce congestion and noise.

The main command at this stage is **create_floorplan**. Using -**core_utilization** switch, we can specify the desired utilization, and using -**core_aspect_ratio** switch, we can specify the height to width ratio of the chip. Default aspect ratio is 1.

Floor plan basic commands in PnR flow:

The next step in the flow is dedicating a certain area of the chip for the cells of the design. This step is called floor planning and is initiated with the command:

$$\boldsymbol{create\_floorplan}$$

This command defines a rectangular boundary based on a certain aspect ratio (width and height), or number of cells. It also places the I/O pad and corner cells based on pre-defined constraints.

Some of the options used within these commands will be listed below:

$$\boldsymbol{-core\_utilization\ 0.25\ \ -core\_aspect\_ratio\ 2\ \backslash}$$

$$\boldsymbol{-start\_first\_row\ -\ flip\_first\_row\ \backslash}$$

$$\boldsymbol{-left\_io2core\ 12.4\ \ -bottom\_io2core\ 12.4\ \ -right\_io2core\ 12.4\ \ -\ top\_io2core\ 12.4}$$

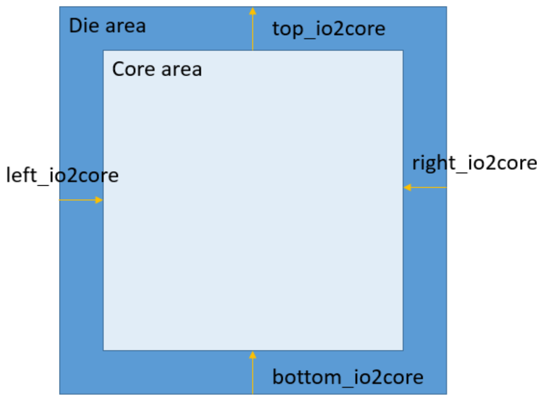


Figure 15: core and die area, with relative io2core distances

The core utilization option is important as it defines how much of design's core's area is utilized.

$$\boldsymbol{Utilization} = \frac{\boldsymbol{total\ area\ of\ the\ cells}}{\boldsymbol{total\ core\ area}}$$

The rest of the core's area will be available for routing.

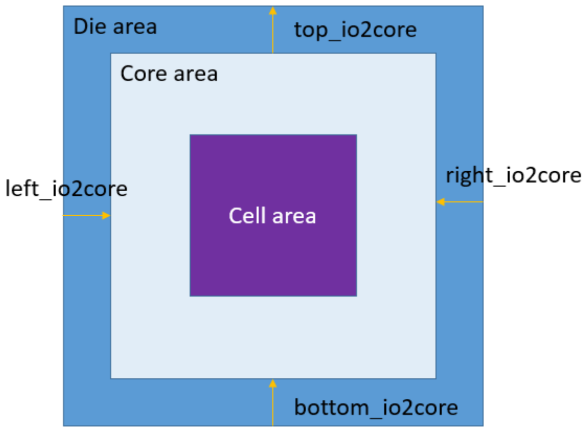


Figure 16: utilization visual, showing cell area compared to core area.

Start first row begins the placement for the bottom left. Flip first row will flip the rows in order to have the correct power line orientation. The io2core is an option defined by the distance between the IO ports and the corners of the specified core, this distance is defined in microns.

.

3.2.2.2. Power planning

Power planning is the second step in PnR.

The power planning stage is very important. At this stage, we determine how our power grid should be, and how we will deliver power to the design's standard cells. We make the power grid in the highest two metal layers; metal 9 and metal 10, and the tape layer is metal 8. We put it in the upper layer because resistance of metal layers decreases as we get to the higher layers, and it is very important to have a power network with low impedance to make sure that the voltage delivered to the cells equals to the ideal voltage, and the IR drop of the metal route is very small. High IR drop also causes electro migration (EM) which can result in catastrophic failure like opens and shorts, or in the best cases it causes performance degradation due to changing the width of the wire, so changes the RC.

Therefore, we need to carefully specify the number of straps, width of straps, width of ring, and the layers of the power grid.

We generally map the IR drop using a color map to highlight "hot spots", where the IR drop is bad. It is also important to check the value of the IR drop itself to see how much percentage of the VDD it represents. It is acceptable to be within 2-3% of VDD.

In order to create and synthesize the power network, there are main steps that we followed:

- First, we create the power and ground network for the design, and logically connect the power pins of standard cells to the specified power nets using **derive_pg_connection** command.
- We define the power ring using command **set_fp_rail_constraints**, as we give it the constraints that will be used to create the power ring like, horizontal ring layer, vertical ring layer, ring width, … etc
- Using **set_fp_rail_constraints**, we constrained the power mesh with max number of straps = 128, minimum number of straps = 20, minimum width = 2.5, and minimum spacing.
- Create virtual pads to be able to synthesize the power network using **create_fp_virtual_pad** command.
- Finally, we synthesize the power network using **synthesize_fp_rail**

- After synthesizing the power network and when the IR (voltage) drop map meets target IR drop constraints. We use **commit_fp_rail** command to generate a real power network (power/ground wires and vias) based on power network synthesis results, Power/ground pins might also be created on the chip boundary.
- **set_preroute_drc_strategy -max_layer** to specify the maximum metal layer used for pre-routing, in our case, it is metal 8.

In this stage, we also define virtual power pads, to have the ability to simulate the IR drop. Also, at the end of this stage, we add tie cells. They are special non logic cells to tie '0' and '1' logic constant values. We connect the by **connect_tie_cells** command.

Power network basic commands in PnR flow:

These are the commands for 2-layer power grid planning.

After initial virtual placing the cells by **create_fp_placement** command (can be done directly after floor planning step), it's time to provide them with power, this is done by creating a network of VDD and GND that spans over several layer in both horizontal and vertical stripes.

First, we tell the tool to connect any net called VDD to the VDD provided in the metals, and the same for the GND nets.

$$\boldsymbol{derive\_pg\_connection\ \ -power\_net\,VDD}\backslash$$

$$\boldsymbol{-ground\_net\,VSS}\backslash$$

$$\boldsymbol{-power\_pin\,VDD}\backslash$$

$$\boldsymbol{-ground\_pin\,VSS}$$

Then we define the constraints which the power rails will follow, this is done using:

$$\boldsymbol{set\_fp\_rail\_constraints\ \ -set\_ring\ -nets\ \{VDD\,VSS\}}\ \backslash$$

$$\boldsymbol{-horizontal\_ring\_layer\,\{\ metal9\ \}}\backslash$$

$$-\boldsymbol{vertical\_ring\_layer}\,\{\,\boldsymbol{metal10}\,\}\,\backslash$$

$$-\boldsymbol{ring\_spacing}\;\mathbf{0.8}\,\backslash$$

$$-\boldsymbol{ring\_width}\;\mathbf{5}\,\backslash$$

$$-\boldsymbol{ring\_offset}\;\mathbf{0.8}\,\backslash$$

$$-\boldsymbol{extend\_strap}\;\boldsymbol{core\_ring}$$

The options define which layers to use for the power connections as well as ring width, spacing and offset from I/O ports.

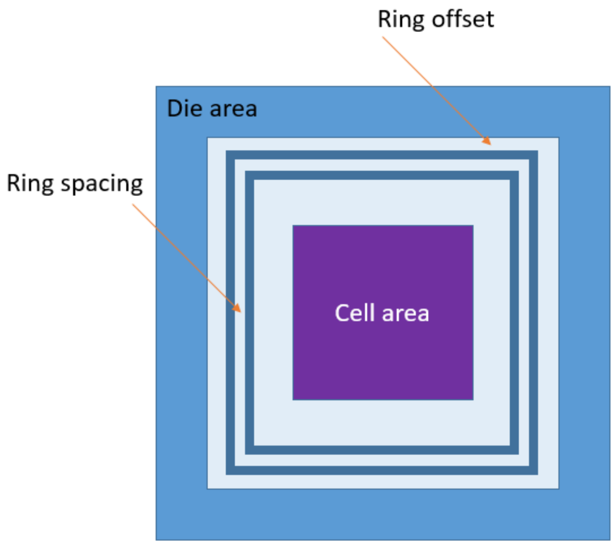


Figure 17: visualization of both spacing and offset

The mesh that provides the power is created using for loops that iterate across the core area placing pads for the power at evenly spaced distances.

```
for {set i "[expr $die_llx + 20]"} {$i
            < "[expr $die_urx − 40]"} {set i [expr $i + 80]} {

create_fp_virtual_pad −net VSS −point "{$i $die_lly}"

create_fp_virtual_pad −net VDD −point "{[expr $i + 40] $die_lly}"

create_fp_virtual_pad −net VSS −point "{$i $die_ury}"

create_fp_virtual_pad −net VDD −point "{[expr $i + 40] $die_ury}"

}

for {set i "[expr $die_lly + 20]"} {$i
            < "[expr $die_ury − 40]"} {set i [expr $i + 80]} {

create_fp_virtual_pad −net VSS −point "{$die_llx $i}"

create_fp_virtual_pad −net VDD −point "{$die_llx [expr $i + 40]}"

create_fp_virtual_pad −net VSS −point "{$die_urx $i}"

create_fp_virtual_pad −net VDD −point "{$die_urx [expr $i + 40] }"

}
```

Each one of the for loops is responsible for the horizontal and vertical placement of the pads respectively.

After placing the pads and constraining which layers the metal are placed in, we synthesize the power rails and rings using:

```
                    synthesize_fp_rail

−nets {VDD VSS} − synthesize_power_plan − target_voltage_drop 22
          − voltage_supply 1.1 − power_budget 500
```

The target voltage drop is an important option that specifies how much IR we can deal with in our chip, and is specified using milli volts

The synthesis is then followed by the actual placement of the power network using:

$$\boldsymbol{comit\_fp\_rail}$$

After creating the network, we can now connect the cells to the receptive power sources using

$$\boldsymbol{preroute\_standard\_cells}$$

We continue by adding tap cells

$$\boldsymbol{add\_tap\_cell\_array\ -master\ \ TAP} \setminus$$

$$\boldsymbol{-distance\ 30} \setminus$$

$$\boldsymbol{-pattern\ \ stagger\_every\_other\_row}$$

3.2.2.3 Placement

The third step in PnR.

In the placement process the ICC tool finds an appropriate position for the leaf cells to be suited on. This process is completed in two different stages.

The first placement stage is the coarse placement, in which the tool places the cells approximately taking into account the timing of the different paths, however in these stages the approximation causes overlaps between the different cells as well as wrong orientation.

The second stage is the legalization stage, wherein the tool starts to legalize the cells by positioning them correctly within the grid, removing overlaps and wrong orientation.

The second stage may result in a degradation for the timing in certain paths which can be further resolved using optimization.

Certain techniques such as blockages and margins can be used during the placement process to ensure that problems like congestion do not occur in certain places.

No blockages were used in our design process.

Placement Density:

This option controls how dense the cells are packed together, and takes as input the percentage of how packed they are where 100% means that there is no empty space between the cells, this is done setting the following variable before actual placement stage or virtual placement if done after floor planning stage:

set_app_var_placer_max_cell_density_threshold X

This X corresponds to a value specified by the designer according to the priority needed. This value gives the high priority to wire length to the placement of cells and its default is 0.7. by decreasing this X the priority of wide placement of cells increases.

Performing placement and optimization:

The placement process is called using the **place_opt** command which consists of 5 different commands including:

- Initial placement (**initial_place**)
- Initial DRC violation fixing (**initial_drc**)
- Initial optimization (**initial_opto**)
- Final placement (**final_place**)
- Final optimization (**final_opto**)
- Legalization

In order to further optimize or incrementally place the cells we can use **refine_opt** which also includes:

- Initial path optimization (**initial_path_opt**)
- Incremental placement (**inc_place**)
- Incremental optimization (**inc_opt**)
- Final placement (**final_path_opt**)
- Legalization

psynopt was also used which performs incremental pre-route or post-route optimization.

There are many options which can be enabled in order to resolve certain issues such as:

- timing driven
- congestion driven

The QoR in the placement process includes reporting placement violations and legalization of non-legalized cells as overlapping cells and so on.

Placement basic commands in PnR flow:

Placement is the process in which we find a suitable physical location for the cells within the core area.

Before starting the placement, we

$$\boldsymbol{check\_physical\_design\ -stage\ pre\_place\_opt}$$

$$\boldsymbol{check\_physical\_constraints}$$

The first of these two commands do checks that depend on the specified stage and makes sure the design contents make sense for the specified stage; otherwise, the report might be meaningless. By choosing preplace opt as the value for the stage, the command requires that the floorplan and netlist data are ready and the design constraints are set.

The second command checks several physical constraints and provide information about possible errors in input. It checks for

- cell areas in hard bound
- Correct layers in the library against those in the floorplan.
- Resistance and capacitance for different route layers.
- Narrow placement areas in the floorplan.
- Legal sites for library cells in floorplan.

The placement process is called using the

place_opt

This stage can also be referred to as coarse placement.

During the coarse placement the IC compiler tool finds an approximate location for the cells based on timing, congestion and power criteria. In this step the cells might overlap, and in the case of IP block usage, those blocks will act as blockages preventing cells from overlapping with them.

Although approximate, the coarse placement is very accurate for initial timing and congestion analysis.

- Initial DRC violation fixing (`initial_drc`)
- Initial optimization (`initial_opto`)
- Final placement (`final_place`)
- Final optimization (`final_opto`)
- Legalization (`legalize_placement`)

Following the coarse placement, legalization moves the cells to legal areas, removing overlap, and finding the right places for the cells within the rows.

These new locations might cause variation in the timing and congestion analysis that was previously achieved, and will usually be more pessimistic as the distances may increase.

psynopt

For further incremental optimization on the synthesis of the placement.

set tie_pins

connect_tie_cells

Tie constant values with VDD and VSS.

3.2.2.4. Clock tree synthesis

CTS or clock tree synthesis is the process of building a buffer/inverter network in order to balance the delays of the flip flops that belong to a clock domain. This delay is caused by the parasitic that appear due to the wire lengths, this delay can also be called insertion delay.

Connection the clock wires causes 2 problems that the CTS helps resolving:

- Skew
- High capacitance and fan-out seen at the clock pin, which causes slow signal transitioning eventually leading to crosstalk issues.

Before we dwell into what CTS does, we first we need some definitions:

- Skew is the difference in clock arrival time at two different registers usually caused by the delay due to buffers and wires placed before clocks pins.
- Capture clock edge: the edge of the clock where the data is read.
- Launch clock edge: the edge of the clock where the data is sent.

Local skew VS global skew:

- **Local skew:** clock arrival between related flops, so each pair of FFs with a data path between them has their own local skew

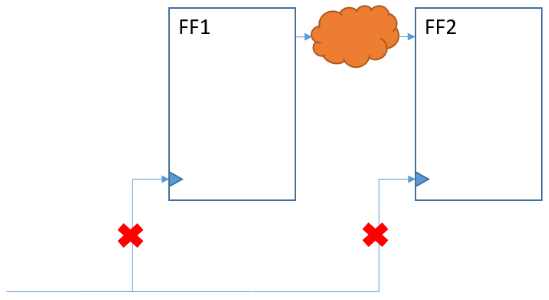


Figure 18: local skew

- **Global skew:** the arrival time difference between the biggest arrival and the smallest arrival, even between unrelated FFs across the entire clock domain.

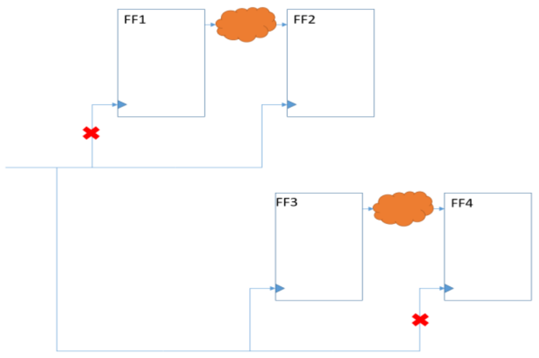


Figure 19: global skew between unrelated registers

CTS is the process in which we try to balance the delay between the different clock paths of the design (sinks or excludes) and the clock source in order to achieve the target global skew, which is done by placing a tree of symmetric buffers so that all the different clock inputs in different flip flops see the same insertion delay. In addition to that, it also helps in decreasing the total capacitance seen by the clock source which reduces transition time as well as crosstalk issues.

Problem with skew: In case the skew is big; if the data flow is in the same direction of clock propagation this will create hold violations, but helps me in setup.

CTS optimizes based on global skew, so the goal is to have a 0 global skew.

By building a buffer network, we ensure that all the flip flops see the same insertion delay while in the time, decrease the total capacitance seen by a single pin.

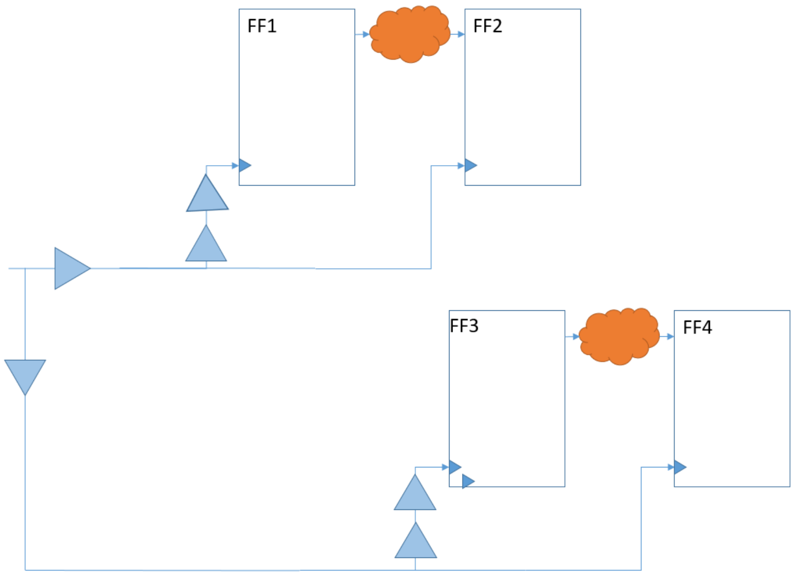


Figure 20: example of buffer insertion to fix the previously mentioned issues

Now that we have an idea about the CTS operation goals, we can delve into the commands used in the CTS script partition.

CTS basic commands in PnR flow:

First, we need to check the design for its readiness the way we did in the placement section,

check_physical_design –stage pre_clock_opt

This stage requires the same items as ***pre_place_opt.*** In addition, the design must be placed and the clock constraints must be set.

check_clock_tree

Is a command preferably used before starting the CTS, this command helps us identify certain issues which might create further problems down the CTS line such as:

- A master clock does not propagate to a generated clock
- Improperly specified master clock
- A master clock terminates at a multi-clock pin
- A clock has no synchronous pins
- A clock loops to itself
- Multiple clocks per register
- Exceptions defined on output pins

Therefore, verification of:

- The clock master.
- Loops in the tree.
- Ignored exceptions.
- Stop pins or float pins on output pins.
- Conflicted exceptions.
- Conflicted balancing settings.

After finishing with the checks, we can now proceed with the CTS

The next command we're going to use is set_driving_cell which sets an attribute for a certain port in our design, where in our case we're specifying the type of buffer used for our clock pin.

Timing characteristics:

In order to model the clock roots correctly we must specify a driving strength for the I/O clock pad cell so that the tool does not assume that the port as an infinite driving strength, this is done using the command:

$$\boldsymbol{set\_driving\_cell\ -lib\_cell\ BUF\_X16\ -pin\ Z\ [get\_ports\ design\_clock]}$$

design_clock is clk_i in our project.

Targets and constraints:

Next is to set the constraints on which the CTS will be optimizing the clock tree on:

```
set_clock_tree_options \
−clock_trees clk_i \
−target_early_delay 0.1 \
−target_skew 0.2 \
−max_capacitance 300 \
−max_fanout 10 \
−max_transition 0.150
```

Most of the issues mentioned before have now been constrained by a certain value.

By default, the tool tries to achieve the best possible skew and latency for the clock tree, however this may lead to degradation in other parameters such as area and power, therefore, **set_clock_tree_options** is used to define the target skew and other constraints.

- clock_trees clk_i
- target_early_delay
- target_skew
- max_capacitance
- max_fanout
- max_transition

Specifying clock tree reference:

The references of a clock tree are the buffer and inverters used to build it, in order to specify them we use: **set_clock_tree_references -references [get_lib_cells */CLKBUF*]**

Further optimization option for the clock tree can be enabled using these commands that allow the CTS to apply different techniques for optimizing the tree:

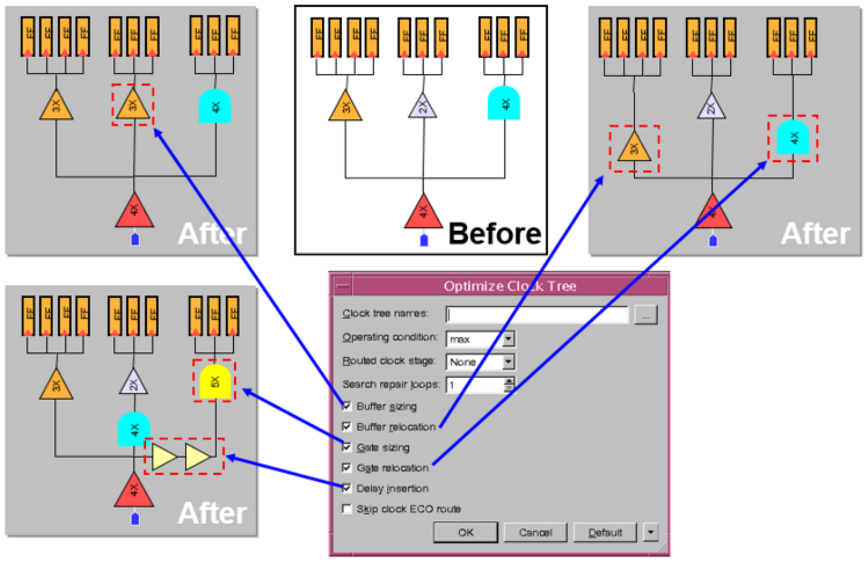
$$\boldsymbol{set\_clock\_tree\_options\ -clock\_trees\ clk\_i} \backslash$$

$$\boldsymbol{-buffer\_relocation\ true} \backslash$$

$$\boldsymbol{-buffer\_sizing\ true} \backslash$$

$$\boldsymbol{-gate\_relocation\ true} \backslash$$

$$\boldsymbol{-gate\_sizing\ true}$$



Clock tree names:
Operating condition: max
Routed clock stage: None
Search repair loops: 1
[✓] Buffer sizing
[✓] Buffer relocation
[✓] Gate sizing
[✓] Gate relocation
[✓] Delay insertion
[ ] Skip clock ECO route

Figure 21: showing the optimization due to the previous commands

Other constraints can also be defined, including specifying the metal layers on which the metal wires that make up the clock tree walk on.

$$\boldsymbol{define\_routing\_rule\ my\_route\_rule}\ \backslash$$

$$\boldsymbol{-widths\ \ \{metal3\ 0.14\ metal4\ 0.28\ metal5\ 0.28\ metal6\ 0.28\ metal7\ 0.8\}}\ \backslash$$

$$\boldsymbol{-spacings\ \{metal3\ 0.14\ metal4\ 0.28\ metal5\ 0.28\ metal6\ 0.28\ metal7\ 0.8\}}$$

This is in case maximum routing layer is metal 8.

As for the width of the wire, non-default wiring is used in order to make the wires wider which helps in decrease crosstalk issues and electro-migration.

$$\boldsymbol{set\_clock\_tree\_options\ -use\_default\_routing\_for\_sinks\ 1}$$

However, we use the default routing for the sinks (a sink is the flip flop seen by the tree, and a tree has many sinks) in order to decrease DRCs.

Implementing the clock tree:

Now for the actual CTS

$$\boldsymbol{clock\_opt}$$

The clock_opt command is a mega command which contains multiple stages that help create the clock tree, such as the synthesis of it, the routing, and lastly the optimization.

The clock_opt command consist of the following three stages:

- The build_clock stage, during which the tool synthesizes and optimizes the clock trees for all clocks in the active modes in all active scenarios. After clock tree synthesis, the tool sets the synthesized clocks as propagated.
- The route_clock stage, during which the tool detail routes the synthesized clock nets.
- The final_opto stage, during which the tool further optimizes the design for timing, logical DRC violations, area, power, and routability.

When you run the clock_opt command, by default, the tool executes all three stages.

These synthesize the clock tree:

```
compile_clock_tree
clock_opt –only_cts –no_clock_route
set_fix_hold [all_clocks]
set_fix_hold_options -prioritize_tns -effort low
set_propagated_clock [all_clocks]
```

set_fix_hold is to consider fixing hold time violation if needed.

```
clock_opt – only_psyn – no_clock_route
```

Pre-route optimization

Clock routing:

```
route_group – all_clock_nets
```

3.2.2.5. Route

Routing is the process of creating physical connections based on logical connectivity. Signal pins are connected by routing metal interconnects. Routed metal paths must meet timing, clock skew, max trans/cap requirements and also physical DRC requirements.

In grid-based routing system each metal layer has its own tracks and preferred routing direction which are defined in a unified cell in the standard cell library.

There are four steps of routing operations:

Global routing

Track assignment

Detail routing

Search and repair

Global Route: assigns nets to specific metal layers and global routing cells. Global route tries to avoid congested global cells while minimizing detours. Global route also avoids pre-routed P/G, placement blockages and routing blockages.

<u>Track Assignment (TA):</u> assigns each net to a specific track and actual metal traces are laid down by it. It tries to make long, straight traces to avoid the number of vias. DRC is not followed in TA stage. TA operates on the entire design at once.

<u>Detail Routing:</u> tries to fix all DRC violations after track assignment using a fixed size small area known as "SBox". Detail route traverses the whole design box by box until entire routing pass is complete.

<u>Search and Repair</u>: fixes remaining DRC violations through multiple iterative loops using progressively larger SBox sizes.

The following commands are a detailed execution of the routing process

routing basic commands in PnR flow:

$$\begin{gathered} \boldsymbol{insert\_spare\_cells} \\ -\boldsymbol{lib\_cell}\,\{\boldsymbol{NOR2\_X4}\ \boldsymbol{NAND2\_X4}\}\,\backslash-\boldsymbol{num\_instances}\ \mathbf{20} \\ \backslash\ -\boldsymbol{cell\_name}\ \boldsymbol{SPARE\_PREFIX\_NAME}\ \backslash-\boldsymbol{tie}: \end{gathered}$$

Inserts spare cells in the legalized design with (unnecessary depending on the purpose):

- -lib_cell {NOR2_X4 NAND2_X4} option which specifies the names of library cells for which the spares cells are to be created. If you specify more than one library cell, the instances of these cells are placed as close as possible
- -num_instances 20 option which Specifies the number of instances of the library cell that are inserted as spare cells
- -cell_name SPARE_PREFIX_NAME option which Specifies the prefix name of the spare cells to be used by the tool. An integer prefix is automatically appended to the prefix name for each instance. If a cell with a generated name already exists, the tool automatically assigns a unique name for the cell name
- -tie option which Ties the input pins of the inserted spare cells to logic zero.

$$\boldsymbol{set\_dont\_touch\ \ [all\_spare\_cells]\ true}$$

This command sets the dont_touch attribute on cells, nets, references,designs in the current design, and on library cells, to prevent modification or replacement of these objects during optimization.

$$\boldsymbol{set\_attribute\ [all\_spare\_cells]\ \ is\_soft\_fixed\ \ true}$$

This command sets the value of an attribute on an object. For a complete list of attributes, see the attributes man page.This command returns a collection of objects that have the specified attribute value set. If the attribute is not set on any objects, the command returns an empty string.

$$\boldsymbol{check\_physical\_design\ -stage\ pre\_route\_opt}$$

This command checks the readiness of the current design for IC Compiler with:

- -stage option specifies which stage to check.
  The checks performed by the **check_physical_design** command depend on the specified stage.

$$\boldsymbol{check\_routeability}$$

Verifies that the current design is routable and Check's pin access points, cell instance wire tracks, pin out of boundaries, min-grid and pin design rules and blockages to ensure they meet the design requirements. It performs a check of the design for optimization in order to substantiate any errors in the design that might need to be fixed or what could help to improve the design. This must currently be run on a placed design.

$$\boldsymbol{set\_delay\_calculation\_options\ -arnoldi\_effort\ low}$$

Defines the delay model used to compute a timing arc delay value for a cell or net. with:

-arnoldi_effort option which Specifies the Arnoldi delay calculation effort level

```
set_route_options
            − groute_timing_driven true
            \−groute_incremental true
            \ −track_assign_timing_driven true
            \−same_net_notch check_and_fix
```

Sets specific options in the internal router control database with:

- **-groute_timing_driven** true option which controls whether global routing is timing driven. The default is false.
- -**groute_incremental** true option which controls whether incremental global routing is enabled. The default is false
- -**track_assign_timing_driven** true option which controls whether track assignment is timing driven.The default is false.
- -same_net_notch check_and_fix option which controls whether the router checks same net notch rule. The default is ignored.

```
                 set_si_options
− route_xtalk_prevention true
\ −delta_delay true \ −min_delta_delay true
\−static_noise true\ −timing_window true :
```

Defines signal integrity options used for analysis or optimization. With:

- **-route_xtalk_prevention** true option which Specifies track assign xtalk prevention. This option is disabled by default.
- **-delta_delay** true option which specifies whether crosstalk delta delay is considered in timing and optimization. When set to false, crosstalk delta delay isnot considered. This option is disabled by default. When set to true, it automatically sets the **-min_delta_delay** option to true as well.
- -**min_delta_delay** true option which Specifies whether min delta delay is considered in timing and optimization. When set to true, min delta delay

enables crosstalk calculation for min timing paths reporting hold. This option also enables min noise propagation for capture clocks in max paths for setup, thereby speeding up the capture clock timing. This option is disabled by default and it is enabled automatically when -**delta_delay** is set to "true".

- **-static_noise** true option which Specifies whether static noise is considered in optimization. When set to false, static noise is not considered. This option is disabled by default
- **-timing_window** true option which Specifies whether timing window overlapping is considered in crosstalk analysis. When set to true, timing window is considered. This options is disabled by default and can only be enabled when -**delta_delay** (or -**static_noise**) option is set to "true".

route_auto

Performs global routing, track assignment, detail routing, and search and repair in one step

Track assignment: in routing step, occurs after global routing and before detailed routing, assigns nets to specific tracks and lays down the actual metal tracks.

Pin assignment: In design planning step, to assign pins at the top level of the design or in macros and plan groups used in bottom-up design flow.

route_opt

Performs simultaneous routing and post route optimization on the design.

psynopt − only_hold_time – congestion

Performs incremental synthesis on the design with **only_hold_time** which Specifies that only hold time violations are fixed and congestion option which enables congestion removal algorithms for improved routability by default, congestion is not enabled.

route_zrt_eco − open_net_driven true

Performs ECO routing on the design with:

-**open_net_driven** true option which Controls whether ECO route fixes DRC violations for the whole design (default) or only in the neighborhood (bounding box) of the open nets. By default (false), ECO route connects the open nets in the design and fixes design rule violations for the entire design. When true, ECO route connects the open nets in the design and fixes DRC violations only in the neighborhood of the open nets.

$$\boldsymbol{verify\_zrt\_route}$$

Verifies and reports routing design rule constraint (DRC) viola tions, net opens, antenna rule violations, and voltage-area rule violations.

$$\boldsymbol{route\_zrt\_detail - incremental\ true}$$
$$\boldsymbol{- initial\_drc\_from\_input\ \ true}$$

Performs detail routing on the design with:

- **-incremental** option which If true (default is false), the router performs incremental mode routing. By default, the router starts from the iteration number that detail routing ends with last time the cell was routed. By default, the router will re-check DRC in the beginning. If -**initial_drc_from_input** is set as true, the router will skip the initial DRC checking and start with the DRC information stored in the cell.
- -initial_drc_from_input option which When true, the detail router uses the DRC information in the cell as the initial DRC information. Use this option very carefully; set it to true only if you are absolutely sure that the DRCs in the cell are up-to-date. The default is false.

$\boldsymbol{Focal\_opt\ \ - setup\_endpoints\ all}$

Performs postroute optimization to fix setup,hold or logical design rule constraint violations, reduce crosstalk , or reduce leakage power with:

-setup_endpoints all option which optimizes the specified endpoints by using setup time as time focal metric.

3.2.2.6. Finishing

To help address some of the process design issues during chip manufacturing, IC provides some commands that provide options which can be applied:

$$\boldsymbol{insert\_stdcell\_filler}$$

To ensure that all the power nets are connected and that all the spacing is filled, these empty spaces in the row can cause power not to be connected throughout the rest of the row which is filler cells are used, using the previous command. You can insert these before routing:

- Insert standard cell fillers
- Insert end cap cells

After routing, you can

- Insert well fillers
- Insert pad fillers

During insertion, filler cells with metal are retained only when they do not cause DRC violations.

$$\boldsymbol{insert\_zrt\_redundant\_vias}$$

Inserting redundant vias is very helpful after routing as during manufacturing, some vias can fail and having an array of them can ensure correct operation.

<u>Reports and checks:</u>

Finally, as we output out PNR into the design library, we run a few check commands to get an idea of how the violations look like:

$$\boldsymbol{verify\_lvs - ignore\_floating\_port - ignore\_floating\_net} \backslash$$

$$\boldsymbol{-check\_open\_locator - check\_short\_locator}$$

$$\boldsymbol{verify\_lvs - max\_error\ 1000}$$

Verify_lvs is an important command that checks for the inconsistencies in the physical layout of the deisign, such as the number of shorted or open wires, this command was used in the previous stages as well, in order to keep track of how each of the commands affect the violations.

$$\boldsymbol{report\_timing\ -delay\_type\ max}$$

$$\boldsymbol{report\_timing\ -delay\_type\ min}$$

Report_timing is another command that was also in almost every stage of the PNR, as it returns the setup and hold slack on which we base our script strategy on.

$$\boldsymbol{analyze\_fp\_rail\ -nets\ \{VDD\ VSS\}\ -power\_budget\ 500\ -voltage\_supply\ 1.1}$$

This command analyzes the created power network in order to check for the IR drop across the entire design as well as the electro-migration for both the power and ground nets.

Used in every stage, we use this command to keep an eye on the IR drop.

The **power_budget** option in this command is in milliwatts and shows us the available power that we distribute among all of the cells of the design.

Before we **write_stream** command, we set options to it use the this first:

$$\boldsymbol{set\_write\_stream\_options\ -map\_layer\ \$sc\_dir/tech/strmout/FreePDK45\_10m\_gdsout.map}\ \backslash$$

$$\boldsymbol{-output\_filling\ fill}\ \backslash$$

$$\boldsymbol{-child\_depth\ 20}\ \backslash$$

$$\boldsymbol{-output\_outdated\_fill}\ \backslash$$

$$\boldsymbol{-output\_pin\ \{text\ geometry\}}$$

map_layer specifies the file that helps in turning the milkyway to GDSII format, the next option is important for the case of hierarchal use, where the **child_depth** specifies the level of child cells in a design. A large number should be used like 20.

output_outdated_fill forces the output of fill data. The last option specifies the type of information that is associated with each pin in the design, the geometry of the pin was chosen in this case.

```
write_stream -lib $design
    -format gds\
    -cells $design\
    ./output/${design}.gds
```

Finally, we turn out design into GDSII format using the **write_stream** command, specifying the output folder and cells.

```
define_name_rules new_verilog -special verilog
    -target_bus_naming_style {%s[%d]}
    -check_internal_net_name -check_bus_indexing

change_names -rule new_verilog -hierarchy
```

These two commands are used for the extractor tool to read the bus names in the right way. It can be replaced by other commands according to the naming style.

```
set verilogout_no_tri true

set verilogout_equation false

write_verilog -pg -no_physical_only_cells ./output/${design}_icc.v

write_verilog -no_physical_only_cells ./output/${design}_icc_nopg.v
```

These are the output files used in StarRC tool for RC extraction and PrimeTime tool for STA.

3.2.2.7. RC extraction

There are two methods:

First one is by using command in ICC tool after finishing which is

$$extract\_rc$$

Then write the output in (.spef) file by the following

$$write\_parasitics\ -output\ \{./output/file.spef\}$$

$$create\_rail\_setup$$

The second method is by using StarRC tool and it is the recommended method.

It has a special style for scripting by setting specific values for maximum capacitance and minimum capacitance. It is run by the tool automatically.

3.2.3. STA

Using PrimeTime tool.

Static timing analysis is a way of assessing a design's timing performance under worst-case scenarios by checking all possible paths for timing violations. It takes into account the shortest possible latency across each logic node, but not the circuit's logical action.

Static timing analysis is faster and more complete than circuit modelling. Because it does not need to simulate many test vectors, it is faster. It's more thorough since it examines worst-case timing for all possible logic situations, not just those that are sensitive to a certain set of test vectors. Static timing analysis, on the other hand, simply analyses the design for right timing, not for accurate logical functionality.

The operation of synthesis with Design Compiler and physical implementation with IC Compiler is driven by timing, area, and power restrictions. These tools synthesize the netlist and execute physical placement and routing with the goal of producing the quickest device with the least amount of area and power in the shortest time possible while staying true to the design specifications. These tools make trade-offs between speed, area, power, and runtime based on the designer's limitations. However, in order to run at the required clock rate, a chip must meet timing limitations; hence timing is the most critical design constraint.

In Flip Flop flavor:

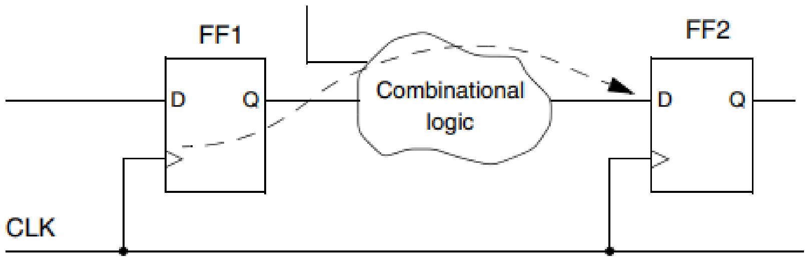


Figure 22: timing path

A timing path is represented by the dashed arrow. Before the next clock edge arrives at FF2, the change in signal data induced by a clock transition at flip-flop FF1 must be propagated to flip-flop FF2, so that the logically processed data can be properly latched onto FF2. Depending on the logic, the data value, and the values of any side inputs going into the logic, the change at FF1.Q could impact the output of the combinational logic cloud at FF2. D. If there is a change at FF2.D, it must happen before the next clock edge reaches FF2.

3.2.3.1. Setup Check Timing:

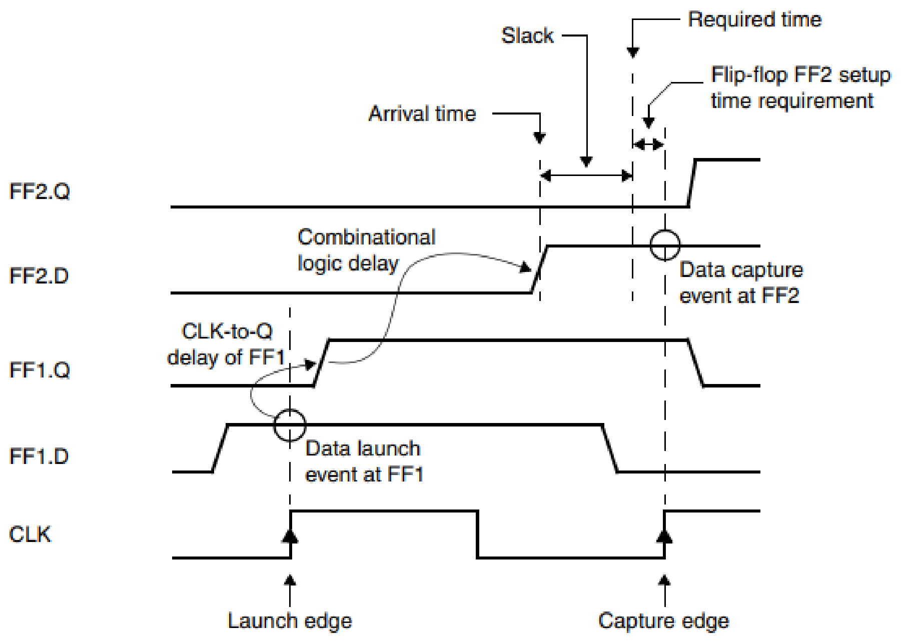


Figure 23: setup timing illustration

With some delay, this signal passes via the combinational logic. The second flip-flop, FF2.D, receives the output of the combinational logic. The arrival time for the path is the time at which the signal value changes. The change in value at FF2.D must occur at least an amount equivalent to the setup time requirement for the flip-flop before the arrival of the clock edge at FF2. The needed time for the path is the latest permissible arrival time. The capture event for the timing path is the latching of data at FF2. The capture event occurs one full clock cycle after the launch event in this case.

The slack of the timing check is the amount of time it takes to meet the timing constraint.

The amount of slack is equal to the needed time minus the arrival time. So, Setup Time is the amount of time the synchronous input (D) must show up, and be stable before the capturing edge of the clock. This is so that the data can be stored successfully in the storage device.

$$\boldsymbol{setup\ slack = the\ data\ required\ time - the\ data\ arrived\ time}$$

Equation 1: slack calculation

The slack is zero and the timing constraint is barely met if the signal arrives exactly at the appropriate time.

The slack is negative if the signal arrives later than expected.

The slack can be positive means that the design is working at specified frequency and it has some more margins as well.

Setup violations can be fixed by either slowing down the clock (increasing the period) or by decreasing the delay of the data path logic.

3.2.3.2. Hold Check Timing:

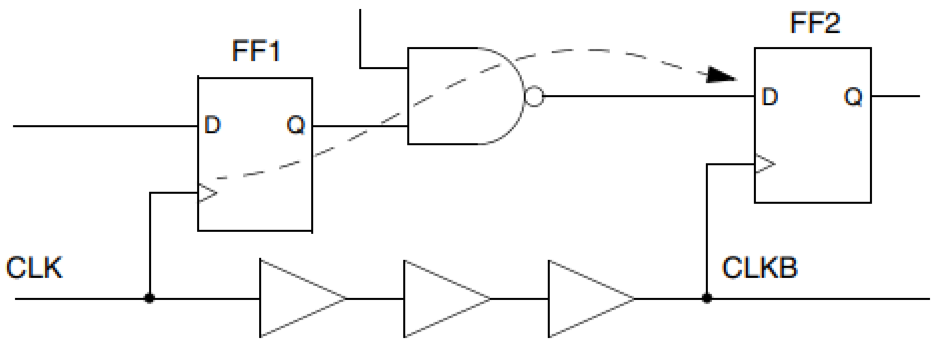


Figure 24: hold time illustration

The timing path, which consists of a single NAND gate, has a relatively short combinational delay from FF1 to FF2. Meanwhile, the three buffers generate a considerable delay in the clock signal between the two flip-flops, which is probably exacerbated by a huge RC delay due to the long route. As a result, the arrival of the capture clock signal CLK2 at FF2 is greatly delayed in comparison to the launch clock CLK1 at FF1.

$$\boldsymbol{Hold\ slack = the\ data\ arrival\ time - the\ data\ required\ time}$$

Equation 2: hold time calculation

3.2.3.3. The types of paths:

- **Register to Register:**

The time it takes for data to propagate via the source flip-flop, travel through combinational logic and routing, and arrive at the destination flip-flop before the next clock edge occurs is known as data arrival time.

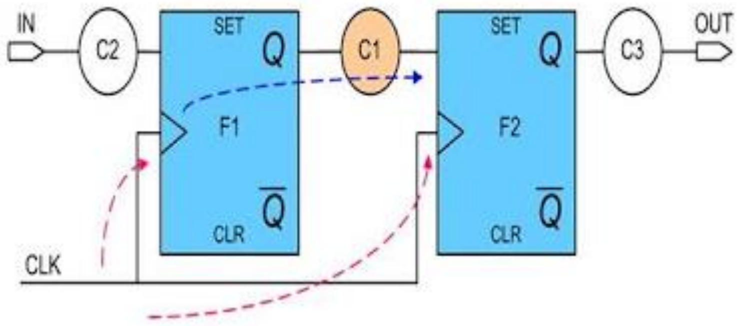


Figure 25: register to register path

$$Arrival\ Time = \ Tclk - q + Tcombo$$

$$Required\ Time\ = Tclock - Tsetup$$

$$setup\ slack = the\ data\ required\ time - the\ data\ arrived\ time$$

$$= (Tclock - Tsetup) - \ (Tclk\ - q + Tcombo)$$

Equation 3: Slack calculation for register-to-register path

Register to Output:

The time it takes for data to leave the source flip-flop, travel via combinational logic and interconnects, and exit the chip through the output port is known as data arrival time.

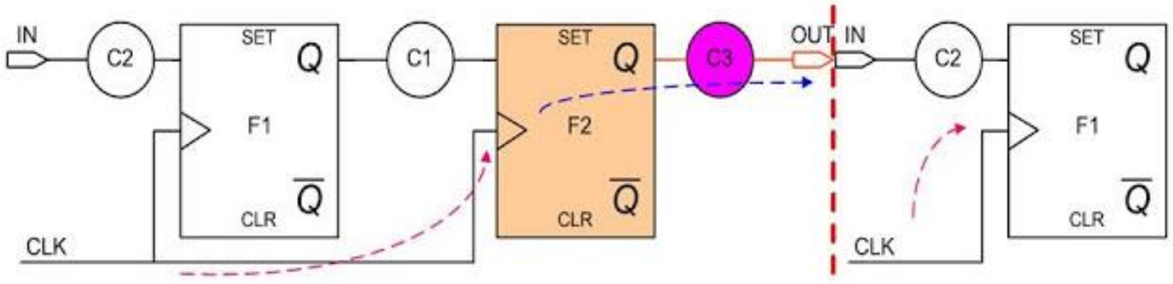


Figure 26: register to output path

$$Arrival\ Time = Tclk - q + Tcombo$$

$$Required\ time = unconstrained$$

Equation 4: Arrival time calculation

- **Input to Register:**

Data arrival time is the time required for the data to start from the input port and propagate through combinational logic and end at the data pin of the flip-flop.

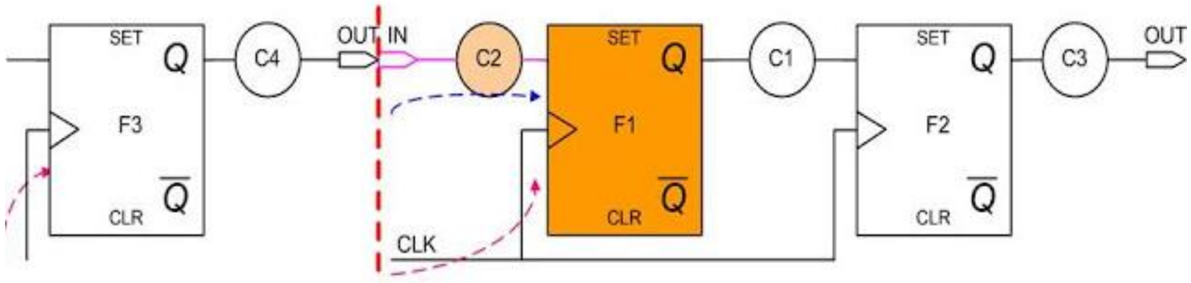


Figure 27: input to register path

$$\boldsymbol{Arrival\ time = Tcombo}$$

$$\boldsymbol{Required\ time = Tclk - Tsetup}$$

$$\boldsymbol{setup\ slack = Required\ Time - Arrival\ Time}$$

$$= (\boldsymbol{Tclock - Tsetup}) - \boldsymbol{Tcombo}$$

Equation 5: slack calculation for input to register path

3.2.3.4. Timing Analysis in the Design Flow:

Timing analysis serves different purposes in different phases of the design flow. In Design Compiler, timing drives the selection of library cells used for synthesis and the allocation of registers between combinational logic in data paths. In IC Compiler, timing drives the placement of cells and the routing of interconnections to minimize delays in the critical paths. In Primetime, exhaustive sign-off timing analysis is the main purpose of the tool.

Many timing analysis features are shared by Design Compiler, IC Compiler, and Primetime. The tools allow you to establish timing limitations and create timing reports with the same commands. The Synopsys Design Constraints, or SDC, are these instructions. These commands have the same syntax and have the same effects across all the supported tools. That implies you can constrain a design in Design Compiler, IC Compiler, Primetime, and other tools using the same SDC script. Design rule restrictions, power limitations, and time constraints can all be specified using SDC commands.

The (**write_sdc**) command generates a script containing a collection of SDC commands that describe the design's present limitations. To read in the file and apply the same restrictions in a different tool, use the read sdc command. The (**read_sdc)** command operates similarly to the source command, except it also checks for SDC compliance in the script commands. SDC script files can be used to transmit restrictions between Synopsys products as well as some external programs.

3.2.3.5. Timing Analysis After Routing:

Following routing, comprehensive nets are accessible for accurate RC extraction, resulting in more accurate delay calculation results. The report constraint command generates detailed reports on the critical paths, while the report timing command reports on timing violations.

Use a sign-off extraction tool like StarRC and a sign-off timing tool like PrimeTime for a complete and final design. A sign-off tool's purpose is to ensure that the design will perform properly in terms of time with the highest possible accuracy. The sign-off tools are more precise at extraction and timing analysis than the synthesis, physical implementation, and optimization tools.

StarRC is a parasitic RC extraction tool that takes into account all capacitive interactions between conductors and precisely mimics physical aspects of wires including dishing, erosion, and the physical closeness of surrounding structures. For a typical design, StarRC separates billions of capacitors and applies an innovative parasitic reduction method to build the shortest possible netlist capable of producing reliable timing analysis results.

PrimeTime is a full-chip static timing analyzer that shares Design Compiler and IC Compiler's libraries, gate-level netlist, parasitic RC data, and SDC timing constraints. PrimeTime performs a thorough analysis with the utmost speed and accuracy, yielding results that are extremely similar to those of the SPICE simulation.

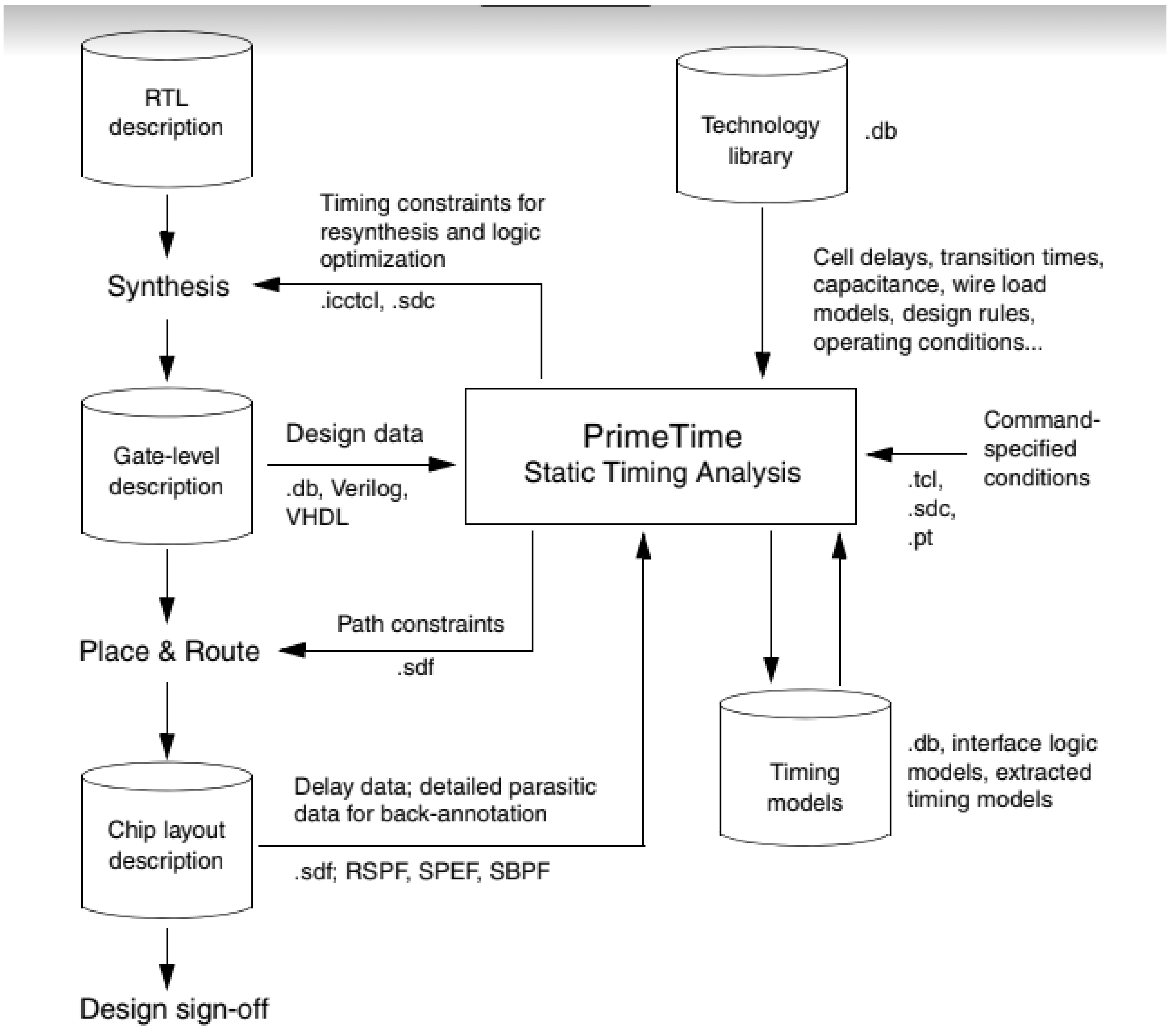

RTL description
Technology library
.db
Timing constraints for resynthesis and logic optimization
Synthesis
.icctcl, .sdc
Cell delays, transition times, capacitance, wire load models, design rules, operating conditions...
Gate-level description
Design data
.db, Verilog, VHDL
PrimeTime
Static Timing Analysis
Command-specified conditions
.tcl, .sdc, .pt
Path constraints
Place & Route
.sdf
Timing models
.db, interface logic models, extracted timing models
Chip layout description
Delay data; detailed parasitic data for back-annotation
.sdf; RSPF, SPEF, SBPF
Design sign-off

Figure 28: Primetime inputs and outputs

Reading the Design Data:

The first step is to read the gate-level design description and information from the linked technology library. Design descriptions and library information in .db format, as well as .ddc and gate-level netlists in Verilog and VHDL forms, are accepted by PrimeTime. read_db, read_ddc, read_verilog, and read_vhdl are the commands for reading design files.

The link design tool resolves all references between distinct modules in the hierarchy and produces an internal representation of the design for timing analysis when you read in a collection of hierarchical design files.

If the chip layout has been completed, back-annotating the design with detailed delay or parasitic information will yield more accurate findings. Use the read sdf command to back-annotate the design with delay information from an SDF file. Use the read parasitics command to back-annotate the design using parasitic capacitance and resistance data. PrimeTime allows the RSPF, SPEF, and SBPF formats for detailed parasitic data.

Parallel Parasitic Reading:

Before the time update can take place, the following types of information must be read in while completing a design analysis:

- Netlists using the **read_verilog**, **read_vhdl**, **read_ddc**, and **read_milkyway** commands.
- Annotations using the **read_parasitics** and **read_sdf** commands.
- Timing constraints and exceptions using the source and read_sdc commands.
- User-defined scripts and custom pre-update reports using the source command.
- Reading detailed parasitics files using the **read_parasitics** command allows the parsing of parasitics files and back annotation to happen in parallel with other unrelated commands in the script.

Constraining the Design:

PrimeTime requires information about the design-level timing limitations in order to do timing analysis:

- Clock characteristics.
- Signal transition arrival times at each input port.
- Signal transition required times at each output port.

Checking the Design and Analysis Setup:

It's a good idea to check the design's characteristics, such as the hierarchy, library elements, ports, nets, and cells, as well as the analysis setup parameters, such as clocks, wire load models, input delay constraints, and output delay constraints, before starting a thorough analysis.

The check_timing command checks for constraint problems such as undefined clocking, undefined input data arrival times, and undefined output data required times. In addition, it provides information about potential problems related to minimum clock separation (for master-slave clocking), ignored timing exceptions, combinational feedback loops, and latch fanout.

Performing a full analysis:

The report_timing command is perhaps the most flexible and powerful PrimeTime analysis command. It provides general or more information about the timing of the whole design, a group of paths, or an individual path. The command options let you specify the types of paths reported, the scope of the design to search for the specified paths, and the type of information included in the path reports.

The report includes the following information about each reported path:

- Path startpoint, endpoint, and intermediate points
- Incremental and cumulative delay at each point along the path
- Final data arrival time at the endpoint
- Time at which the data must arrive at the endpoint to meet the constraint (“data required time”)
- Setup timing slack (required time minus arrival time)

In the **report_timing** command, the -**delay_type** option specifies the type of timing checks to report. You can set the delay type to maximum using the max value for setup checks, minimum using the min value for hold checks.

Timing Analysis Updates:

The timing update at any time with the **update_timing** command. By default, PrimeTime minimizes the time it spends on a timing update by analyzing only the parts of the design that are affected by changes since the previous timing update. You can override this default behavior and update the entire delay by using the -full option of the update_timing command.

Fixing Timing Violations:

When PrimeTime indicates a timing violation, check the violation report to see if it's an actual violation rather than a condition like a false path or an erroneously defined constraint. Use the numerous "report" commands to figure out what caused each violation. PrimeTime lets you temporarily change the design in certain ways, without modifying the original netlist, so you can easily test the timing effects of those changes. To make these changes, use the insert_buffer, size_cell, and swap_cell commands or use fix_eco_timing -type setup -method size_cell command to resize the cells Automatically.

Saving and Restoring Single-Core Sessions:

The **save_session** and **restore_session** commands allow you to save the current state of a PrimeTime session and restore the same session later. When you need to examine the results of an earlier timing analysis, using the save and restore feature avoids repeating the costly time-consuming **update_timing** portion of the analysis. Additionally, you can save a session before it is updated so that you can also save and restore the time-consuming pre-update actions. The **restore_session** command takes you to the same point in the analysis using only a small fraction of the original runtime. This command clears out any existing design data and library data in PrimeTime memory before it restores the saved session.

You can use the **save_session** command to save sessions before or after using the **update_timing** command. If the timing of the design is already updated, the saved session includes the timing. If only incremental timing updates are pending, the incremental update is performed and the resulting timing is saved with the session. If a full timing update is pending or the timing has not yet been updated in the analysis, the **save_session** command does not save any timing information. Saving such sessions without timing allows designs and parasitics to be loaded once, saved, and reused, improving throughput for multiple design runs that share a netlist and parasitics. In all cases, the **save_session** command requires a linked design and causes an implicit link if the design has not yet been linked.

3.3. ASIC design methodologies

There are three flows:

- Flat flow
- Hierarchical flow
- Topographical flow

Our target is to get along with the three flows and decide which one fits more our project objective which is the highest speed.

Let's know what is the difference between these three flows:

3.3.1. Flat overview

Flat design is the design in which the whole module is flattened. It has all the controls for the smallest cell in the design and hence can serve more optimization techniques.

Most of the cases, flattening increase the area

3.3.2. Hierarchical overview

Hierarchical design uses blocks to represent lower-level modules as black boxes then gather them is a higher level reaching top module as instantiations.

There is no access to internal details, so it's harder in optimization.

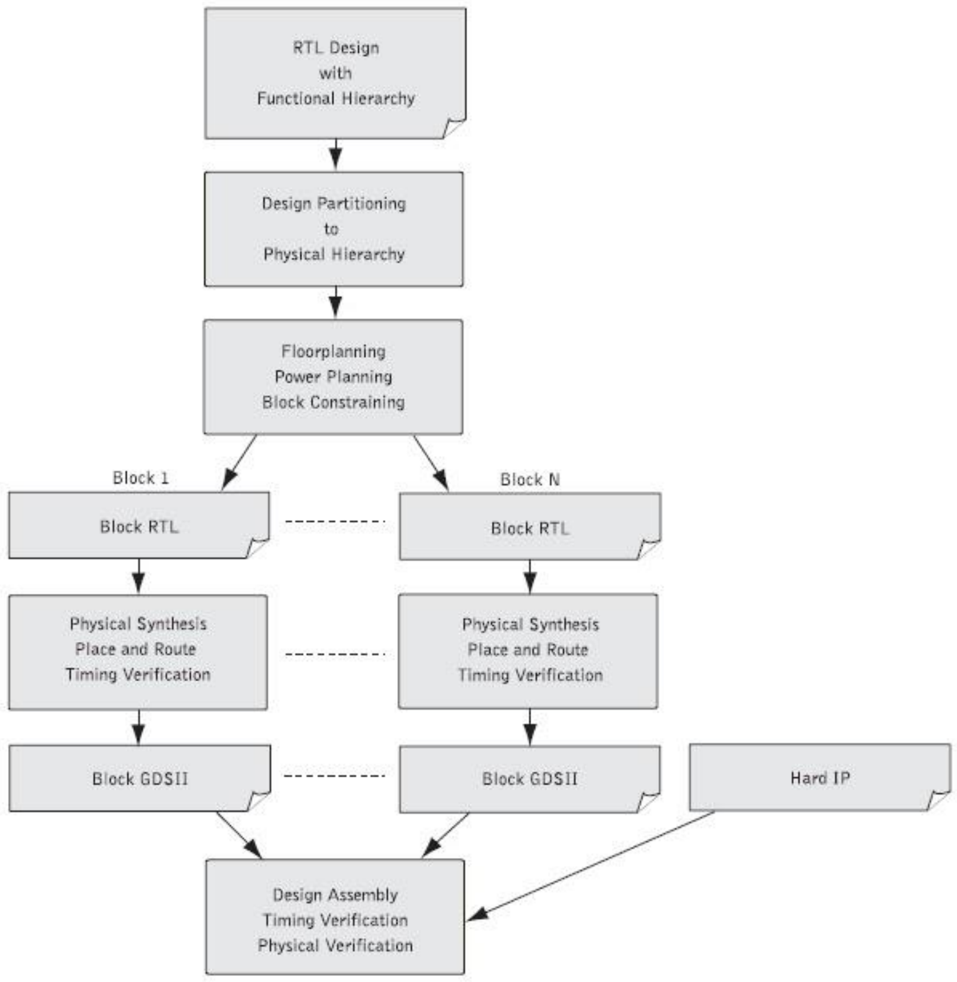

RTL Design with Functional Hierarchy
Design Partitioning to Physical Hierarchy
Floorplanning Power Planning Block Constraining
Block 1
Block N
Block RTL
Block RTL
Physical Synthesis Place and Route Timing Verification
Physical Synthesis Place and Route Timing Verification
Block GDSII
Block GDSII
Hard IP
Design Assembly Timing Verification Physical Verification

Figure 29: represents hierarchical flow bottom-up method

3.3.3. Topographical overview

‘Topographical technology offers much-needed predictability for a convergent RTL- to- GDSII path. Front-end designers no longer have to wait for layout results to uncover critical design issues; they can identify and fix them up front. In turn, back-end teams receive a better netlist for physical implementation which is more likely to meet the desired performance” said Philippe Magarshack, group vice president, Central CAD and design solutions.

This means that Topographical flow improve the design process for better optimization technique, as it gives the synthesis flow future insight about the floor plan in the PnR flow then returns after synthesizing to PnR at placement step. Synthesizing with these information gives faster, reliable and meet design specifications efficiently.

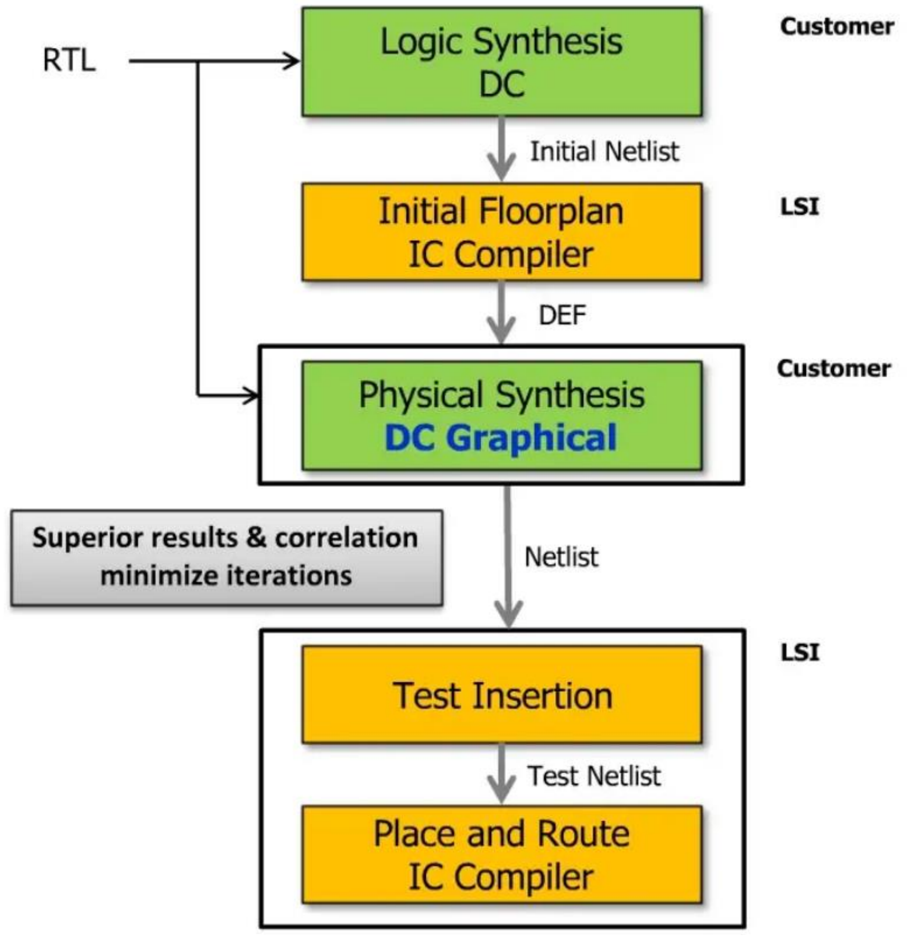


Figure 30: illustration for topographical flow

In our project, our objective is to maximize the speed of RISC-V.

We will examine the design by the three ASIC flows which are flat, hierarchical and topographical flows and choose the one which meets our objective.

The following chapters will scope our technical work in the three flows, compare results and reach final conclusion.

Chapter 4

Flat flow

4.1. synthesis in Flat flow

- First by running the synthesis script as discussed in with period = 5ns (frequency = 200MHz), the results shown in the table are got

Table 6: Synthesis results of first run in Flat Flow

Setup slack	Hold slack	Cell area	Total dynamic power	Total leakage power
-0.61 ns	0.00 ns	42793.282356	1.9909 mW	197.1497 uW

So as shown the setup slack is negative and too large so it needs much effort to reduce it as it is done in the next step

- In this step instead of using (compile -map_effort meduim) command the tool effort is increased to be high instead of medium to be (compile -map_effort high) command, the results shown in the table are got

Table 7: Synthesis results of second run in Flat Flow

Setup slack	Hold slack	Cell area	Total dynamic power	Total leakage power
-0.33 ns	0.00 ns	43072.050352	1.9991 mW	198.6210 uW

So as shown the setup slack is reduced (as magnitude) but still negative and large so it also needs much effort to reduce it as it is done in the next step

- In this step instead of using (compile -map_effort high) command the effort is increased to be high instead of medium to be (compile_ultra) command which (performs a high_effort compile on the current design for better quality of results (Qor)), the results shown in the table are got

Table 8: Synthesis results of third run in Flat Flow

Setup slack	Hold slack	Cell area	Total dynamic power	Total leakage power
0.00(met) ns	0.00 ns	26182.114529	1.8613 mW	132.8229 uW

So as shown the setup slack is 0.00 met so it is indication that the speed can be increased as it is done in the next step.

Note:

Compile_ultra: performs a **high_effort** compile on the current design for better quality of results (QoR). As with the compile command, optimization is controlled by constraints that you specify on the design. This command is targeted toward high performance designs with very tight timing constraints. It provides you with a simple approach to achieve critical delay optimization.

- In this step with the same command (compile_ultra) the period is reduced more and more till reaching that period = 2.5ns (frequency = 400MHz) which is double speed of the last step , the results shown in the table are got

Table 9: Synthesis results of forth run in Flat Flow

Setup slack	Hold slack	Cell area	Total dynamic power	Total leakage power
-0.41 ns	0.00 ns	28099.974447	3.7526mW	142.1629uW

So as shown the setup slack is negative and too large so we need much effort to reduce it as it is done in the next step.

- In this step instead of using (compile_ultra) the (compile_ultra -timing_high_effort_script) command is used which (runs a strategy intended to improve the resulting delay of the design, possibly at the cost of additional runtime), with the same period in the last step (period = 2.5 ns), the results shown in the table are got.

Table 10: Synthesis results of fifth run in Flat Flow

Setup slack	Hold slack	Cell area	Total dynamic power	Total leakage power
-0.37ns	0.00 ns	28208.236457	3.7715 mW	142.5580 uW

So as shown the setup slack is reduced (as magnitude) but still large, so it is better to proceed to the next step in the flow (the PnR) where more optimization can be done.

Note:

-**timing_high_effort_script:** runs a strategy intended to improve the resulting delay of the design, possibly at the cost of additional runtime. The strategy can make changes to variable or constraints that modify **compile_ultra** behavior and perform additional passes to achieve better delay.

- In this step the register_file latch-based file is replaced by register_file flip flop based so there is a good result in timing and with increasing the options of the compile_ultra to be compile_ultra -timing_high_effort_script –retime –incremental with retime option which Uses the adaptive -retime algorithm during optimization to improve delay and –incremental option which Runs compile_ultra in incremental mode. In the incremental mode, the tool does not run the mapping or implementation selection stages. the results shown in the table are got.

Table 11: Synthesis results of the final run in Flat Flow

Setup slack	Hold slack	Cell area	Total dynamic power	Total leakage power
0 ns (met)	0.00 ns	31962.5605 um^2	4.2435 mW	159.5308 uW

Note:

-incremental: Runs compile_ultra in incremental mode. In the incremental mode, the tool does not run the mapping or implementation selection stages.

-retime: Uses the adaptive retiming algorithm during optimization to improve delay. This option is ignored if the -only_design_rule option or the -top option is chosen at the same time.

4.2. PnR in Flat flow

The Synopsys IC Compiler tool integrates proprietary design planning, physical synthesis, clock tree synthesis, and routing for logical and physical design implementations across the design phase to give a complete netlist-to-GDSII design solution.

After proceeding to insert and route the complete core inside the chip area after successfully synthesizing the core with the target clock and limitations. Now our target to get clean timing and clean LVS. So, we followed different scenario's so as to get better performance.

- **Latch based flavor in Register file:**

Level-sensitive latches are used to implement the registers in the latch-based register file. When compared to an implementation utilizing normal flip-flops, this provides for significant area savings, making the latch-based register file the top choice for ASIC implementations. Commercial tools can be used to simulate the latch-based register file. The latch-based register file cannot be simulated using Verilator. The latch-based register file can also be used for FPGA synthesis; however, this isn't encouraged because latches aren't well supported by FPGAs. Using the source file cv32e40p_register_file_latch.sv in the project to select the latch-based register file. When the latches are not written, a technology-specific clock gating cell must be provided to keep the clock inactive. This cell must be wrapped in the cv32e40p_clock_gate module.

1st scenario:

- With period = 2.5 nsec.
- Core utilization = 40%.
- Aspect ratio = 2.
- 4-layers for the power (metal 7 and metal 9) are horizontal ring layers and (metal 8 and metal 10) are vertical ring layers

The power supply voltage is 1.1 V and the maximum IR drop is 11.05 mV which is less than 22 mV and the average power is 500 mW. Also, the worst path is register to output with slack equals -1.05 nsec.

- Using the place_opt command without focusing in any optimization

Resulting in approximately 0 percent congestion in both vertical and horizontal directions. Also, the worst path is register to output with slack equals -0.12 nsec. and the area equals 30919.840506.

- Nothing is changed in the CTS script as discussed in the CTS script section:

  - Total dynamic power is 4.2971 mW.
  - Cell leakage power is 171.1259 uW.
  - Total cell area is 31904.306500
  - Worst path of setup is -0.15 nsec.
  - Worst path in Hold is -0.03 nsec.

- The only difference in the script of the routing in this scenario is that using "route_auto" command.

To run automatic routing, use route_auto command. The route_auto command is used to perform global, track assignment, and detail routing. Zroute reads the block before starting routing and updates the block when all routing steps are completed when running route_auto. Zroute verifies the input data when restarting routing with this command if you stop automated routing before it conducts detail routing. When running routing, use the route auto command to check for convergence, congestion, and design rule quality-of-results (QoR). If congestion QoR is more important to you than timing QoR, you might wish to utilize route auto.

As a result, in the routing step:

- Total SHORT Nets are 373.
- Total Open Nets are 0.
- Total number of DRC = 21941.
- Worst path Setup slack is -0.35 nsec.
- Worst path Hold slack is -0.03 nsec.

So, it is clear that the first scenario results are not good as the shorts are large and also the setup slack is too large that we cannot continue with this scenario.

2nd scenario:

In the second scenario, our target is to decrease the shorts as we can so,

- With period = 2.5 nsec.
- Core utilization = 40%.
- Aspect ratio = 2.
- 2-layers for the power (metal 9) is horizontal ring layer and (metal 10) are vertical ring layer

The power supply voltage is 1.1 V and the maximum IR drop is 11.05 mV which is less than 22 mV and the average power is 500 mW. Also, the worst path is register to output with slack equals -1.06 nsec.

- Using the place_opt command without focusing in any optimization

Resulting in the congestion, the congestion is high in both vertical and horizontal directions. Also, the worst path is register to output with slack equals -0.08 nsec. and the area equals 30221.590496.

- Nothing is changed in the CTS script as discussed in the CTS script section:

  – Total dynamic power is 4.2540 mW.
  – Cell leakage power is 165.0084 uW.
  – Total cell area is 31110.296492.
  – Worst path of setup is -0.13 nsec.
  – Worst path in Hold 0 violated.

- Nothing is changed in the routing script of the first scenario.

As a result, in the routing step:

- Total SHORT Nets are 412.
- Total Open Nets are 0.
- Total number of DRC = 12742.
- Worst path Setup slack is -0.43 nsec.
- Worst path Hold slack is 0.01 nsec met.

So, it is clear that the first scenario results are not good compared to the first scenario and our target as the shorts are large and also the setup slack is too large that we cannot continue with this scenario.

3rd scenario:

- Period=2.5ns.
- Core utilization= 30%.
- Aspect ratio = 2.
- Using the command in the floorplanning before the virtual placement:

placer_max_cell_density_threshold 0.4:

This variable enables a mode of coarse placement in which cells are not distributed evenly across the surface of the chip, but are allowed to clump together. The value you specify sets the threshold of how tightly the cells are allowed to clump. The value of 1.0 allows no gaps between cells.

A reasonable value is one that is above the background utilization of your design but below 1.0. For example, if your background utilization is 40%, or 0.4, a reasonable value for this variable is a value between 0.4 and 1.0. The higher the value, the more tightly the cells clump together.

- 2-layers for the power (metal 9) is horizontal ring layer and (metal 10) are vertical ring layer
- Nothing is changed in the CTS script as discussed in the CTS script section:

  - Total dynamic power is 4216 uW
  - Cell leakage power is 163.6 uW.
  - Total cell area is 30882.0685
  - Worst path of setup is -0.04 nsec.
  - Worst path in Hold -0.1 nsec.

- Nothing is changed in the routing script of the first scenario.

As a result, in the routing step:

  - Total SHORT Nets are 387.
  - Total Open Nets are 0.
  - Total number of DRC = 3976.
  - Worst path Setup slack is -0.15 nsec.
  - Worst path Hold slack is -0.1 nsec.

So, it is clear that the shorts aren't accepted as well as the timing so many iterations are required.

4th scenario:

- Period=2.5ns
- Core utilization = 25%
- Aspect ratio = 2.
- Using the command in the floorplanning before the virtual placement:

placer_max_cell_density_threshold 0.3.

- 2-layers for the power (metal 9) is horizontal ring layer and (metal 10) are vertical ring layer
- Nothing is changed in the CTS script as discussed in the CTS script section:

  - Total dynamic power is 4.5105 mW.
  - Cell leakage power is 169.57 uW.
  - Total cell area is 31385.074513.
  - Worst path of setup is -0.12 nsec.
  - Worst path in Hold -0.04 nsec.

- Nothing is changed in the routing script of the first scenario.

As a result, in the routing step:

- Total SHORT Nets are 2.
- Total Open Nets are 0.
- Total number of DRC = 11.
- Worst path Setup slack is -0.16 nsec.
- Worst path Hold slack is -0.04 nsec.

- **Flip-flop based flavor in Register file:**

The registers of the flip-flop-based register file are implemented using ordinary, positive-edge-triggered flip-flops. As a result, it is the first choice when using Verilator to simulate the design. Using the source file cv32e40p_register_file_ff.sv in the project to select the flip-flop-based register file.Also, working on clock period of 2.6 nsec which is 384.6 MHz.

Floorplanning:

Because it involves the location of I/O pads and macros, as well as power and ground structures, floorplanning can be difficult. Before beginning physical floorplanning, make sure that the data that will be used during the physical design process is appropriately prepared.

All ASIC physical designs require proper data preparation in order to apply a correct-by-construct methodology. So, at the beginning the physical standard cells have to be put inside the boundaries of the die area. So, we put an aspect ratio (is the ratio between width of the die to height of the die) equals 2 and utilization of 25%. At which the utilization is defined as the ratio of the area of the standard Cells to the area of the chip minus the area of the macros and area of blockages. So as to reduce the congestion, we used the virtual placement with the congestion switch is on and high effort at which it enables congestion-driven placement mode. And the high effort to Specifies the effort level for congestion mode. The default effort level is medium. Expect a significant increase in run-time for high effort. Also, we used the timing driven option so as to enable direct timing-driven placement mode to get positive slack with margin out of this phase.

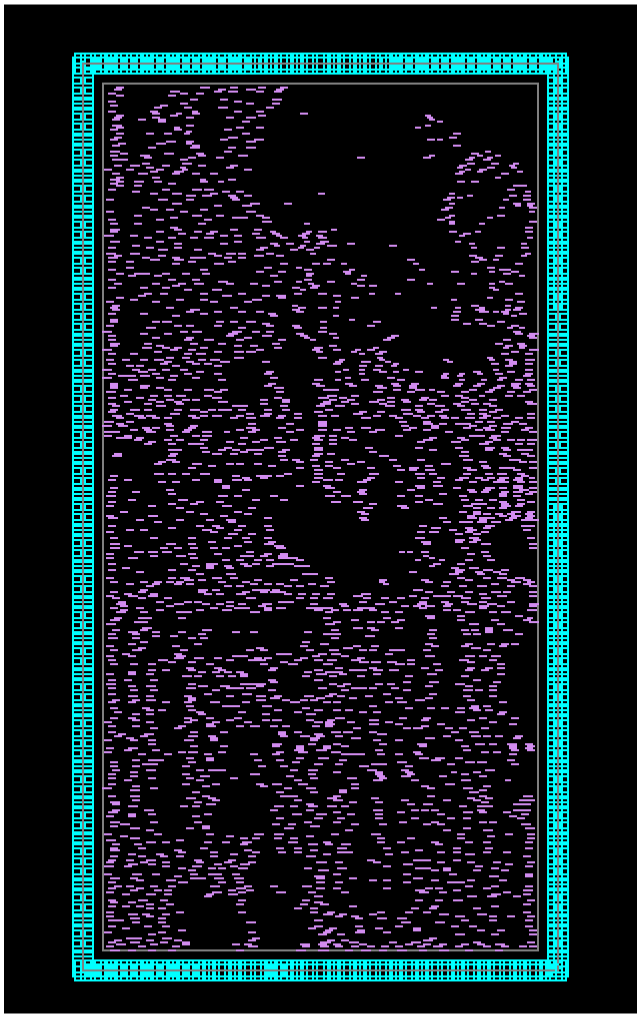


Figure 31: floor plan and virtual placement in flat flow

This figure shows the floorplanning with 25% in utilization of the core area and aspect ratio of 2.

Then we define the metal layers that will be used for routing stage, this is important at floorplanning stage so that PnR tool can perform global routing to estimate routing regions and potential congestion properly

After this stage, the congestion approximately equals to zero and the slack is zero met.

The final stage in this part is to use virtual flat placement, which is at the heart of our design flow, so that cells can freely move around the core without being constrained.

After we've virtually positioned the design in the core area and established the placement strategy, we're ready to set up the power grid that will supply the standard cells with the required voltage supply.

Power planning:

In this scenario, in each standard cell, we define the power and ground nets and ports so that the tool can connect to the grid afterwards. we used 2-layers for the power (metal 9) is horizontal ring layer and (metal 10) is vertical ring layer. and the maximum voltage drop not to exceed 2% of the normal power supply. So as to simulate the power flow in the network, the tool needs the virtual power pads so that it can compute the available current flown from them and build the power grid entirely on the basis of virtual ports.

So that the layers will be used as the following:

- Metal 9 and metal 10 are horizontal/vertical power straps
- Metal 8 is the tap layer.
- Metal 1 to metal 6 are used for routing to decrease the shorts.
- Metal 3 to metal 7 for clock signal routing as higher metal layers have less resistance than lower layers which means lower interconnections propagation delay.

Using the command:

```
define_routing_rule my_route_rule  \

  -widths {metal3 0.14 metal4 0.28 metal5 0.28 metal6 0.28 metal7 0.8}

 -spacings {metal3 0.14 metal4 0.28 metal5 0.28 metal6 0.28 metal7 0.8}
```

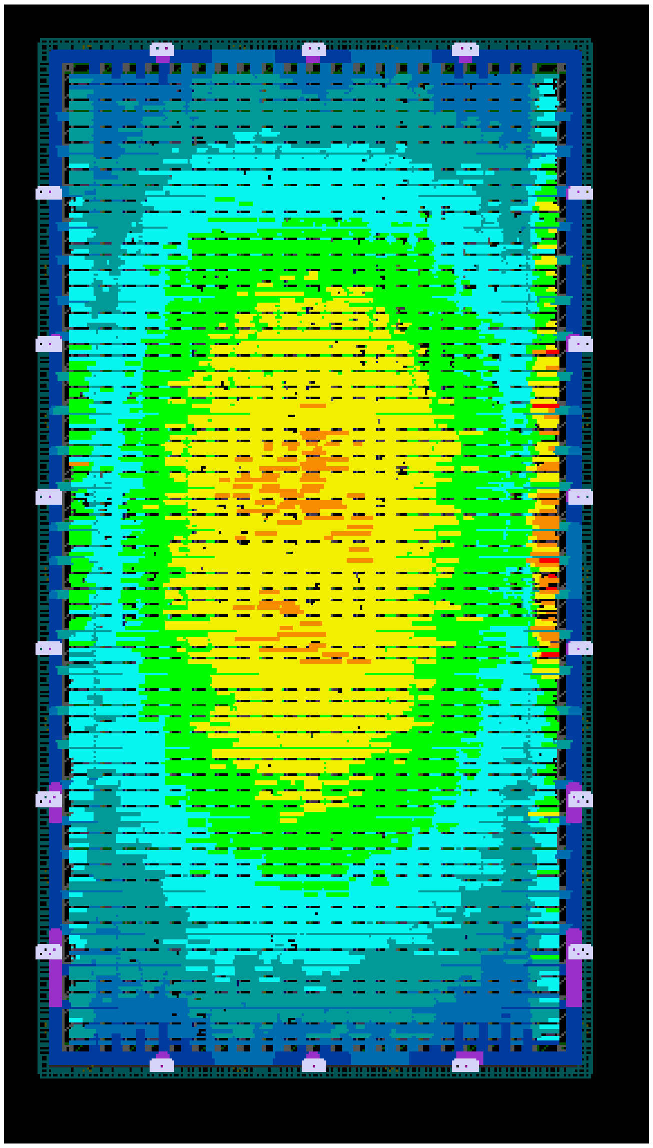


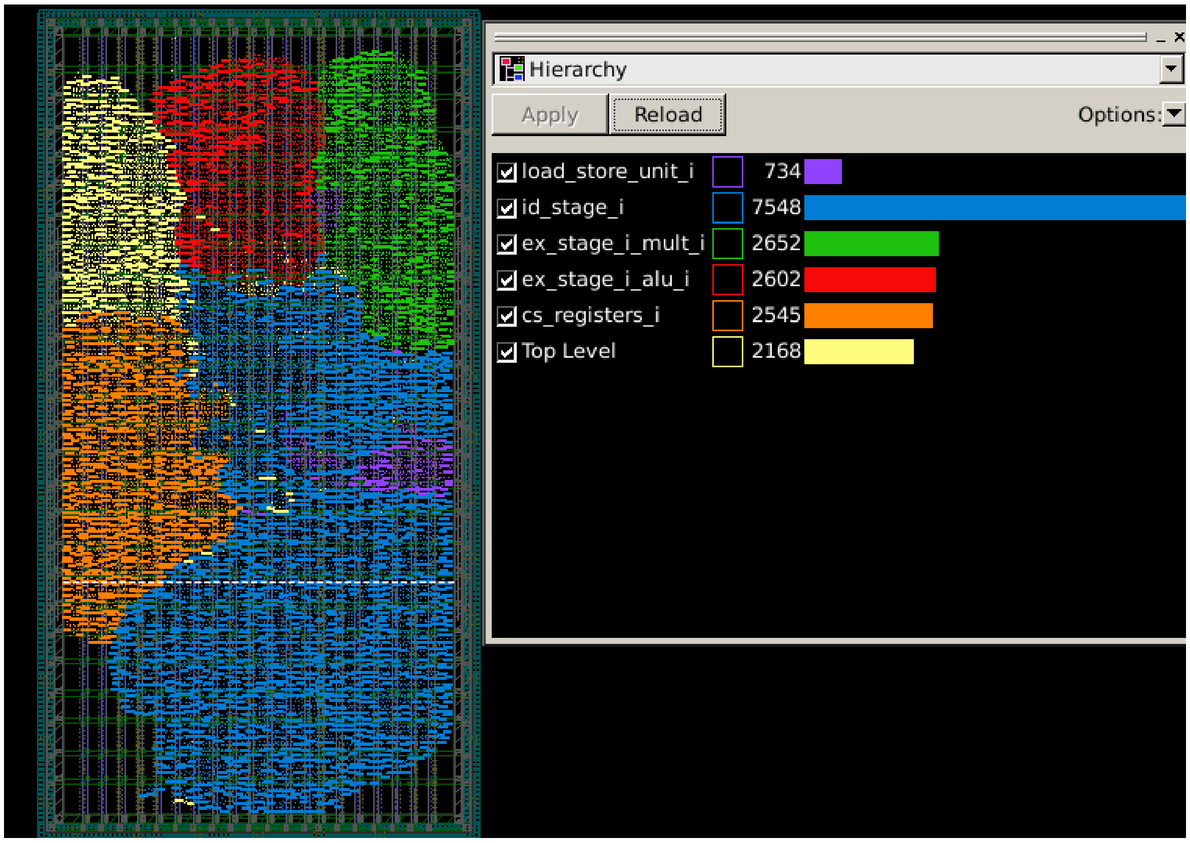
Figure 32: voltage drop map

PNA Voltage drop is constrained by the voltage supply value, here voltage supply of 1.1 V is used which constrains the maximum IR Drop value by 22 mV (2% of the voltage supply). We achieved maximum VDD IR Drop of 12.092 mV and maximum VSS IR Drop of 11.356 mV. These values are the values inside the red region in Figure 32 which are acceptable as long as they are below 22 mV.

Placement:

We used a standard coarse placement step after executing a virtual flat placement and synthesizing the power network on the chip. We used a congestion-driven placement in this stage to improve routing congestion in the design, and the outcome was an overflow of a few GRC cells, as shown by about 0% congestion in both vertical and horizontal directions. The timing requirements were also satisfied in this stage. Therefore, the design was clean in timing and congestion so,

- The setup slack is 0.05 met.
- The hold slack is 0.07 met.



✓ load_store_unit_i	🟪	734
✓ id_stage_i	🟦	7548
✓ ex_stage_i_mult_i	🟩	2652
✓ ex_stage_i_alu_i	🟥	2602
✓ cs_registers_i	🟧	2545
✓ Top Level	🟨	2168

Figure 33: placement map in flat flow

The design's post-placement hierarchy Each of the six colors represents a high-level block in the hierarchy. The color blue dominates the design area since it represents the majority of the flattened pieces.

CTS:

The next stage is to construct our clock tree after setting the design with no congestion or timing violations. A single clock source is dispersed throughout the entire semiconductor in this configuration. This stage is divided into three sub-stages, each with its unique set of features. We set the clock nets' fanout to 10 in order to reduce clock latency and skew as much as possible. We gave the tool some information about timing needs to assist it in building the network with some limits. The target early delay was set to 0.1 nanoseconds, with a minimum skew of 0.5 nanoseconds and a maximum clock transition of 0.150 nanoseconds. After all these information, we are ready to start the clock tree stages as discussed in the CTS section.

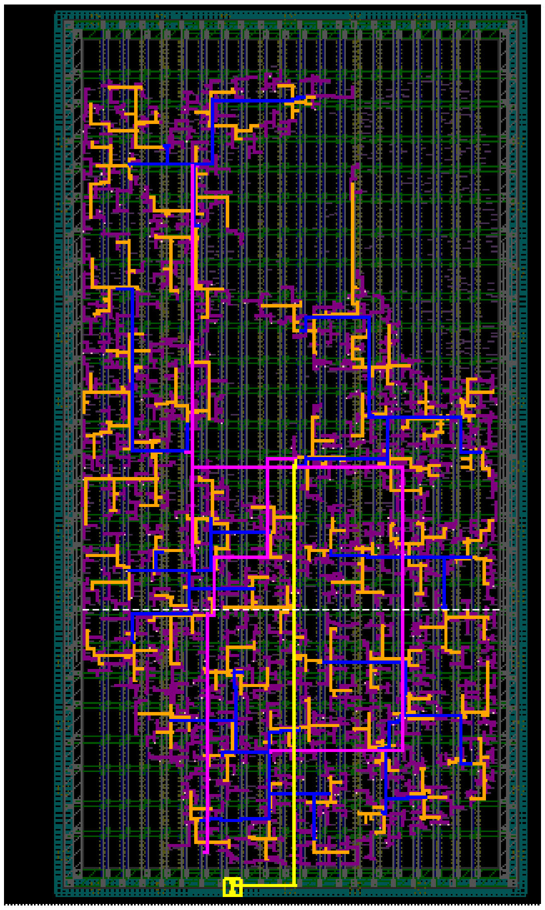


Figure 34: CTS in flat flow

The clock tree network is spread throughout the entire chip.

Because of the non-uniform distribution of sequential devices, the density of clock nets in the chip is not balanced, as seen in the figure.

As a result, in the CTS step:

- Total dynamic power is 4.9644 mW (100%).
- Cell leakage power is 174.5627 uW.
- Total cell area is 34004.376546
- Worst path of setup is 0 met.
- Worst path in Hold is 0.02 met.

Routing:

The physical design is now ready to be routed after completing the core clock network without any timing or DRC violations. However, before proceeding to the routing stage, we must first check for congestion so that the tool can route efficiently. In the post-CTS stage, the resultant congestion is nearly 0% in both vertical and horizontal directions.

First of all, specify the routing layers. We choose to route at maximum of six layers which are metal-1, metal-2, metal-3, metal-4, metal-5 and metal-6 layers. we can forward to our routing steps directly. The routing process comprises of three main phases: global routing, track assignment and detailed routing. Each of them is discussed in detail in the routing section. When routing is complete, it is time to correct physical DRC violations.

We can see that there is almost no congestion in the design by examining the congestion one more time before routing. As a result, the separate global routing stage can be skipped. The IC compiler tool has a routing command that can complete all three routing processes in a single hit.

To run automatic routing, use route_auto command. The route_auto command is used to perform global, track assignment, and detail routing. Zroute reads the block before starting routing and updates the block when all routing steps are completed when running route_auto. Zroute verifies the input data when restarting routing with this command if you stop automated routing before it conducts detail routing. When running routing, use the route auto command to check for convergence, congestion, and design rule quality-of-results (QoR). If congestion QoR is more important to you than timing QoR, you might wish to utilize route auto.

After performing this step, we found many DRC violations on nets and timing requirements was not met. So, we had to bypass this step and jump right into the routing phase. This step addresses all routing difficulties, although it takes a long time to complete. We discovered few time and physical DRC violations after successfully completing the first full routing phase. The program also provides an incremental routing phase that allows you to resolve time and DRC violations without having to reroute the entire design. After all, we have got a clean timing and physical DRC results.

As a result, in the routing step:

- Total SHORT Nets are 0.
- Total Open Nets are 0.
- Worst path Setup slack is -0.04 nsec.
- Worst path Hold slack is 0.03 met.

Chip finishing:

The finishing touches on the chip are the final stage in the place and route. The main reason for this phase is that the manufacturing process necessitates certain measures in order to ensure an error-free chip during the manufacturing process. The following are the four elements that make up this stage:

- Metal layers spreading and widening
- Standard cell filling
- Redundant vias insertion
- Metal filling

Metal layers spreading and widening:

Is carried out to prevent minimum width interconnections from openings and minimum-spacing interconnections from shorts, which could occur as a result of any random particles that may fall on the chip and cause damage during the fabrication process. However, spreading and expanding may result in new temporal inconsistencies.

Standard cell filling:

Is carried out in the empty areas in the normal cell rows to make the chip's density uniform and to increase its yield. Some places may still be unoccupied since filling would result in DRC violations if they were filled.

Redundant vias insertion:

Is the addition of extra vias to the original vias so that if one of the vias in both the original and extra vias fails, the connection between the metal layers will not fail since another contact will join the metal layers together.

Metal filling:

Is used to prevent over-etching of metal interconnections in low-metal-density locations during the fabrication process. Metal fill near key nets on the same layer is deleted or trimmed, and timing driven metal filling is performed precisely to protect timing on critical nets.

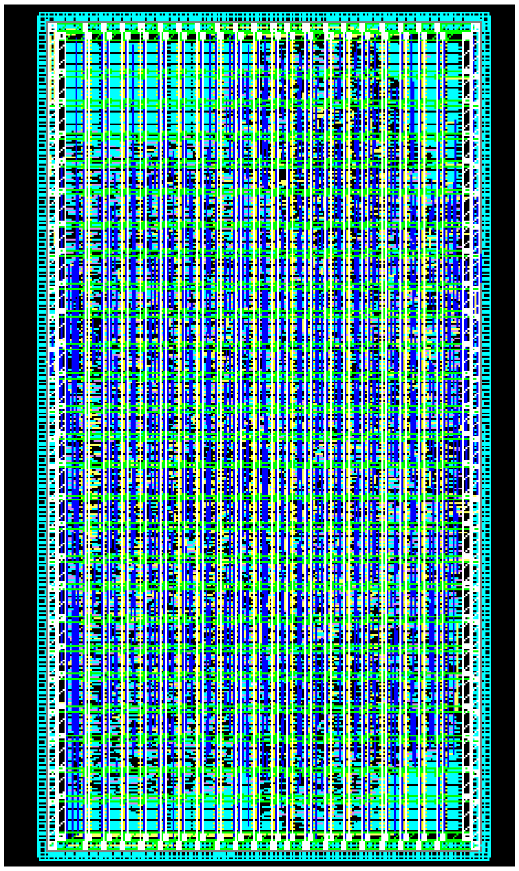


Figure 35: Chip after finishing in Flat Flow

To fill the spaces between the actual cells and wires, the die has been filled with dummy standard cells and false metal wires. Actual cells appear in a dark violet color, while dummy cells appear in a pale blue color.

After completing this stage, we must do a final check for timing and DRC violations across the entire design. If any of them appear, we can use a simple synthesis to correct them, and then the chip is ready to be built following a post-layout sign-off in a STA tool to ensure that the timing of the entire chip is accurate.

4.3. STA in Flat Flow

The next stage after completing a clean place-and-route phase is to Use a high-precision static timing analysis tool to verify the timing. The timing factor is quite important to consider. We have a STA tool that we use.

Synopsys PrimeTime is one of the projects we've worked on, and we're going to talk about it on our project.

The goal of utilizing this tool at this stage is to check timing across a range of projected operating situations for the design. Temperature, operating voltage, and manufacturing process are the three key dimensions of these operating circumstances. MOSFET threshold voltage is another factor that can be considered.

We can divide manufacturing process analysis into three categories: slow slow, typical-typical, and fast-fast process. The slow-slow process, as indicated by its name, has the highest standard cell delay among the others, while the fast-fast process has the shortest. In terms of the temperature outside, the highest temperature-based cells have the greatest delay, while the lowest temperature-based cells have the smallest delay.

The operating voltage has a considerable impact on the propagation delay, as the highest voltage-based cells have the shortest delay and the lowest voltage-based cells might contribute with the longest delay.

The threshold voltage of the MOSFET is the final influencing factor. Low voltage threshold (LVT), regular voltage threshold (RVT), and high voltage threshold (HVT) are the three major forms of threshold voltage found in MOS devices (HVT).

This element has a substantial impact on the typical cells' delay. The lowest threshold-based cells have the shortest propagation delay, whereas the highest threshold-based cells have the longest propagation delay. We only have regular voltage threshold-based libraries in our scenario. This sort of library has a wide range of working circumstances, as described earlier.

As a result, the best-case library has the lowest temperature, greatest voltage, and fastest procedure, whereas the worst-case library has the opposite of these conditions.

We can go through our job and address the predicted instances after we've identified them.

In order to fulfil the setup and hold timing requirements, all of these situations must be met.

Referring to the tool's inputs, the tool must read the Verilog netlist. In addition to the operational libraries that we will examine in this scenario, so that we can define the real delays of the physical nets, we'll need the restrictions and parasitics files.

The issue is that the tool estimates the nets' propagation delay using non-real resistance and capacitance values, resulting in non-real delay values. The standard cell library provides a table that can be used to compute the delay. When parasitic values exceed certain thresholds, the STA tool starts extrapolating based on the table's most recent parasitic values. It then adds an additional 10% delay to the maximum value of the defined delay. This might lead to inaccuracy in delay calculations. That finding demonstrated that the parasitic values are incorrectly written in the file in some way.

We proceeded to analyze the design against the various instances after successfully completing the setup. In most situations, the setup timing was perfect, and the hold time was the issue. The way to get the actual setup time is to fix them in STA tool itself and to write the changes that have done to the design, and then forward them the layout tool to make these changes on the design, and finally the tool can write a new DDC-based file that contains the changes. That new file can be forwarded again to the STA tool for further analysis. The PrimeTime STA tool offers a built-in synthesis engine to fix the setup and hold timing violations.

Chapter 5

Hierarchical flow

There are two topologies in hierarchical flow:

- **Top-down approach**: here we analyze all RTL design files including top module at the same time, the tool handles interdependencies between files automatically. This method is recommended most of times but is hard in large designs; it needs large memory to adopt all the files at simultaneously during analysis step.
- **Bottom-up approach**: here each file is constrained and analyzed independently; at the end of the flow, they are assembled together.

In our project we used the **top-down** approach which fit for RISC-V design.

The open-source RISC-V in PULP has constrained file at period= 5nsec, our objective is highest possible speed with clean lvs and met timing.

Hence, we started to try the flow with period= 2.5 nsec and utilization = 0.4 it gave us large violation for timing and DRCs:

– Slack = -0.12 nsec
– Hold= -0.01 nsec
– Shorts > 300

Then we tried further optimizations including reducing cell placement density in single area, maximizing the number of routing layers and routing optimization.

These helped us to reach maximum optimization in period= 2.5 nsec and utilization=0.25 with results:

– Slack = -0.06 nsec
– Hold= -0.01 nsec
– Shorts <10

So, we increased the period gradually till reaching the final iteration with the following results at period=2.7 and utilization= 0.5:

- Slack = 0.01 nsec (met)
- Hold= 0.02 nsec (met)
- Shorts =0
- Opens=0
- DRCs=1 which can be solved by other tools not available with us.

Here are the steps for the final successful iteration

5.1. Synthesis in hierarchical flow

```
set_app_var search_path "/home/standard_cell_libraries
        /NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End
        /Liberty/NLDM"
```

```
set_app_var link_library " * NangateOpenCellLibrary_ss0p95vn40c.db"
```

```
set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"
```

Here we started to read our open-source library Nangate and included the slow-slow corner database in link and target libraries at temperature -40 celsius degrees.

```
sh rm -rf work

sh mkdir -p work
```

Here we remove any old folder named work and create another new one named in “work”.

```
define_design_lib work -path ./work
```

Here we define the path of design library work which is the folder we created before.

```
set hdlin_sverilog_std        2009
```

The open-source RTL had different version than the tool we have, so we created this command to match the versions in reading the files.

Then we analyzed all the RTL files included in the open-source RTL and the packages needed that are instantiated inside these files.

These files should be analyzed in order of which the file instantiated in the other should be called before the other.

Analyzing is done by ***analyze*** command

The clock gating file is created with us as we instantiated it from library, because the one downloaded from PULP is for simulation only.

Then elaborated, checked and linked the design as discussed in the ASIC flow on chapter 3

uniquify

This command is very important in hierarchical design, it differentiates by name the instantiations in the design that have the same names. This is important for optimizations as any change for example in the size of any instantiated cell, it will change only this cell and doesn't affect the other identical instantiated cells because they have different names after this command.

Then we compiled the design with ***compil_ultra*** command with option ***–no_autoungroup*** which differentiate between flat flow and hierarchical as it tells them not to flatten the design while compiling. Another option is added which are ***−timing_high_effort_script*** and ***–retime*** to fix violated slack

To finish the synthesis with **0 MET** timing we had to repeat the previous compile command 3 more times with another added option to the above ***−incremental***

Then we wrote the generated netlist in verilog format and. ddc using the following

write_sdc output/${design}.sdc

define_name_rules no_case − case_insensitive

change_names − rule no_case − hierarchy

change_names − rule verilog − hierarchy

set verilogout_no_tri true

$$\boldsymbol{set\ verilogout\_equation\ \ false}$$

$$\boldsymbol{write\ -hierarchy\ \ -format\ verilog\ -output\ output/\$\{design\}.v}$$

$$\boldsymbol{write\ -f\ ddc\ -hierarchy\ \ -output\ output/\$\{design\}.ddc}$$

Define name rules and change names are done to make the recent files match the next files in the flow.

Final synthesis results:

Table 12: The synthesis results of the Hierarchical Flow

Slack	Area	Total power
0 met	31567.284529	3.8288e+03 uW

5.2. PnR in hierarchical flow

We started with the design setup reading the netlist, constraints, TLU+ files as stated in design setup in chapter 3

Floor planning:

- Aspect ratio: 2
- Utilization: 0.25
- Flipping the first row
- Io2core 12.4

Setting the maximum routing layer to be metal 8

$$\boldsymbol{set\_ignored\_layers\ -max\_routing\_layer\ metal8}$$

Then setting the following command with 0.3 which gives more priority to reduce the density of cells at small areas than the wire length

$$\boldsymbol{set\_app\_var\ placer\_max\_cell\_density\_threshold\ 0.3}$$

Then followed by virtual placement for cells

$$\mathrm{c}\boldsymbol{reate\_fp\_placement}$$

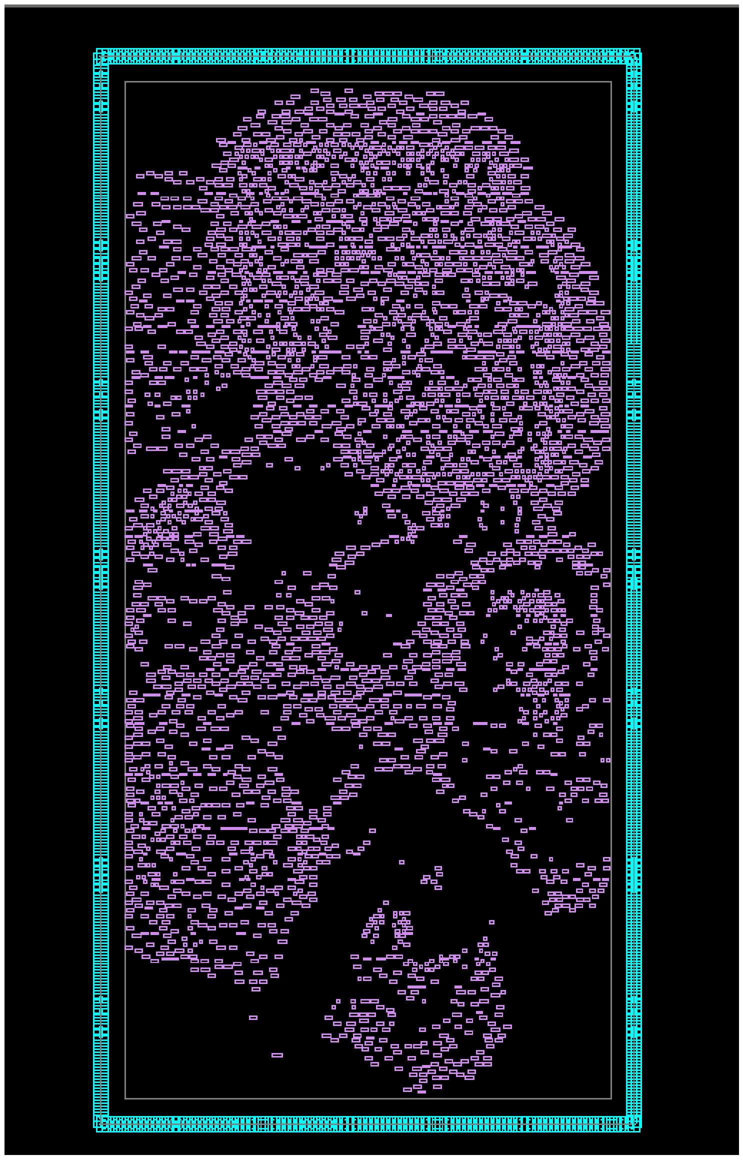


Figure 36: floor plan and virtual placement in hierarchical flow

Power planning:

We used power ring and power strips to be in layer 10 and 9 only. Minimum Strap width is 2.5 nm.

Then we added power pads and tap cells.

$$\boldsymbol{add\_tap\_cell\_array\ -master\ \ TAP} \backslash$$

$$\boldsymbol{-distance\ 30} \backslash$$

$$\boldsymbol{-pattern\ \ stagger\_every\_other\_row}$$

Tap cells are used to prevent the latch up problem.

In this step, we should check for the max IR drop, it should be less than 1%-2% of the VDD.

In our design, we use 1.1V for VDD so we need the max IR drop <22 mV

Using the following command

$$\boldsymbol{analyze\_fp\_rail\ \ -nets\ \{VDD\ VSS\}\ -power\_budget\ 500}$$
$$\boldsymbol{-\ voltage\_supply\ 1.1}$$

We found that for VDD and VSS the max IR drop is 11.328 mV and 11.743 mV respectively which are less than 22 mV

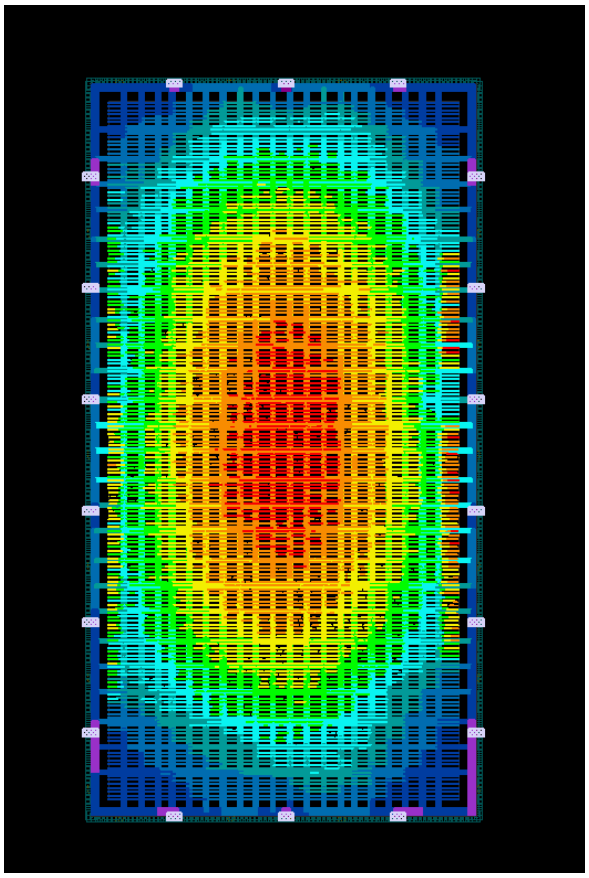


Figure 37: IR drop map for hierarchical flow

If IR drop is violated it needs other tools to fix it, so you should then increase the metal layers for power grid.

Placement:

Started with initial placement by ***place_opt*** command then optimization with ***psynopt*** command

And added tie cells to tie logic 0 and logic 1 to VDD and VSS.

We should connect VDD and VSS to cells after each flow in PnR or any insertion for cells using the following command.

```
derive_pg_connection    −power_net VDD\
−ground_net VSS\
−power_pin VDD\
−ground_pin VSS
```

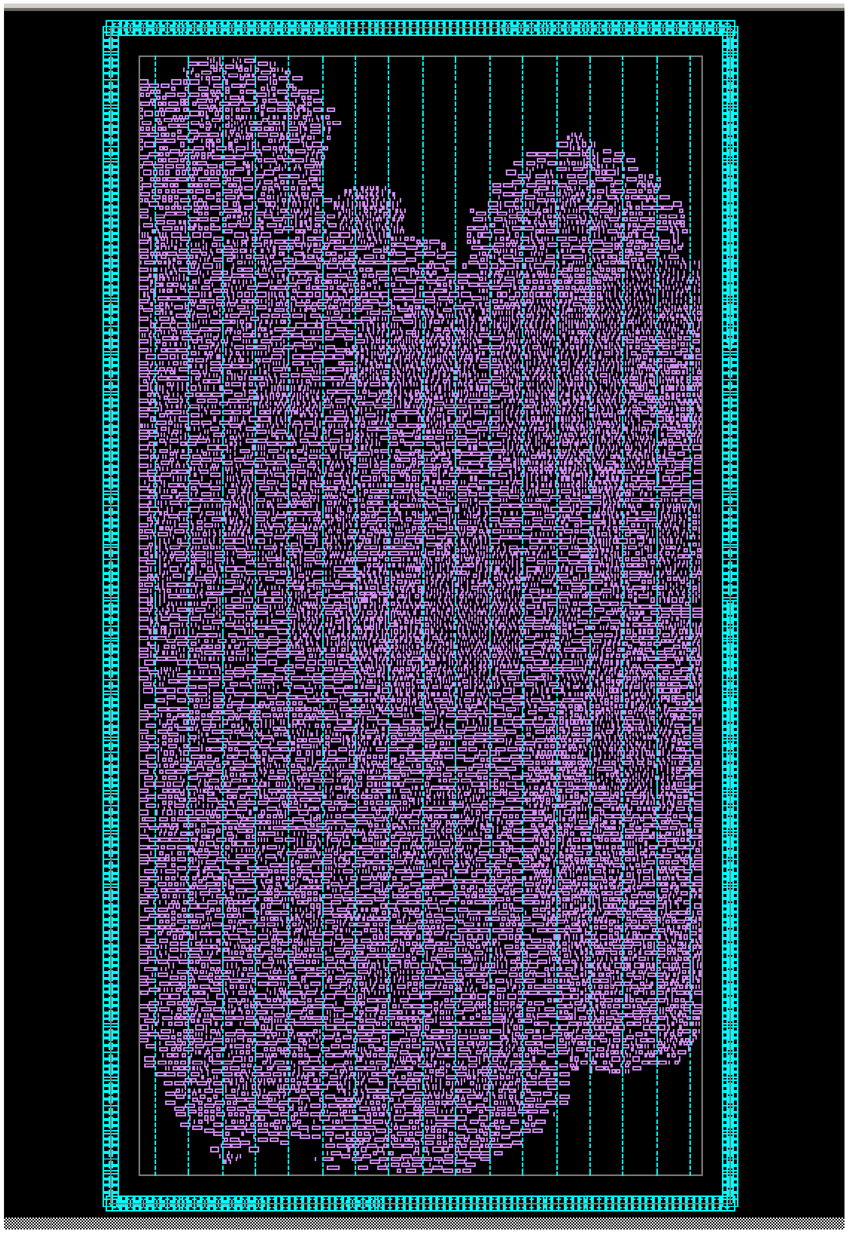
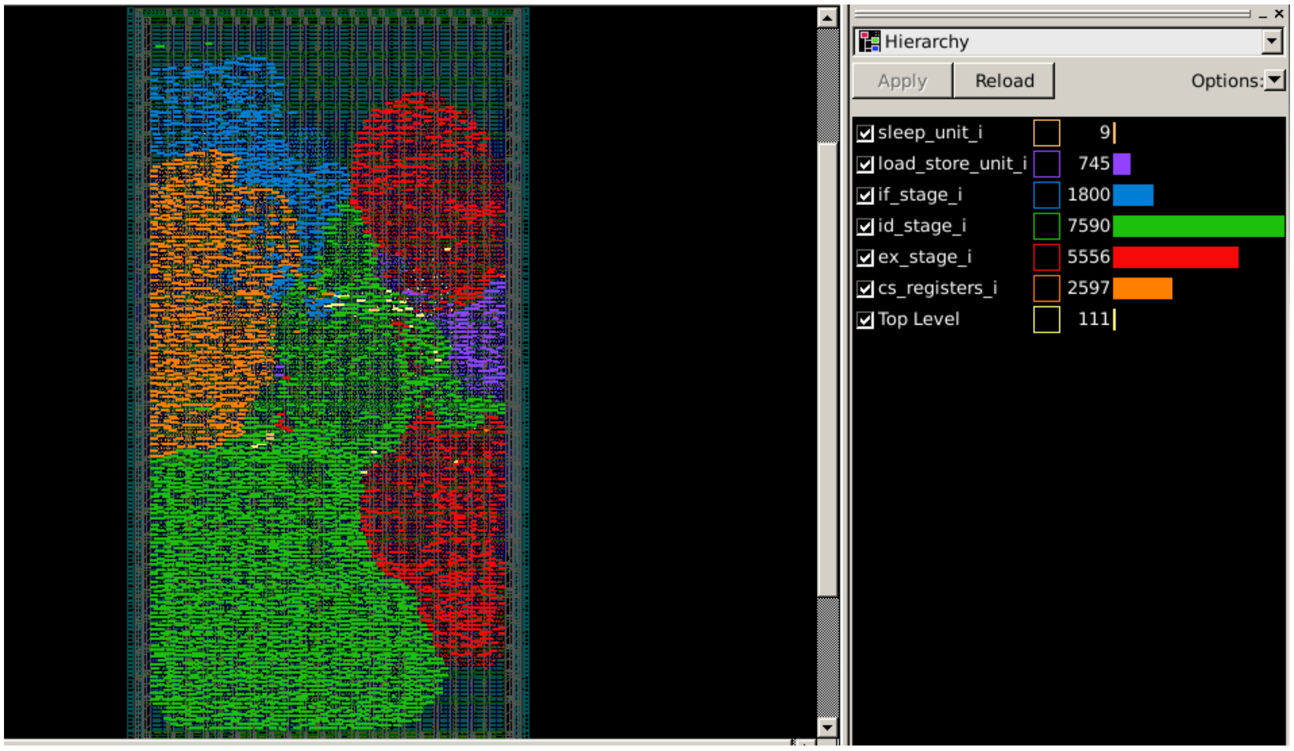


Figure 38: after placement in hierarchical flow



✓ sleep_unit_i	🟧	9
✓ load_store_unit_i	🟪	745
✓ if_stage_i	🟦	1800
✓ id_stage_i	🟩	7590
✓ ex_stage_i	🟥	5556
✓ cs_registers_i	🟧	2597
✓ Top Level	🟨	111

Figure 39: placement map in hierarchical flow

CTS:

we set the driving cell to be BUF_X16, and target skew is 0.2 in clock tree options.

Layer list is: metal3, metal4, metal5, metal6 and metal7.

Then we synthesized the clock tree using ***compile_clock_tree***

Then performed optimization using:

$$\boldsymbol{clock\_opt\ -only\_cts\ -no\_clock\_route}$$

Then to fix hold:

$$\boldsymbol{set\_fix\_hold}\ [\boldsymbol{all\_clocks}]$$

$$\boldsymbol{set\_fix\_hold\_options\ -prioritize\_tns\ -effort\ low}$$

$$\boldsymbol{set\_propagated\_clock}\ [\boldsymbol{all\_clocks}]$$

```
clock_opt −only_psyn −no_clock_route
```

Then routing the clock tree using ***route_group − all_clock_nets*** **.**

Also connected the design after updates again with power grid VDD and VSS.

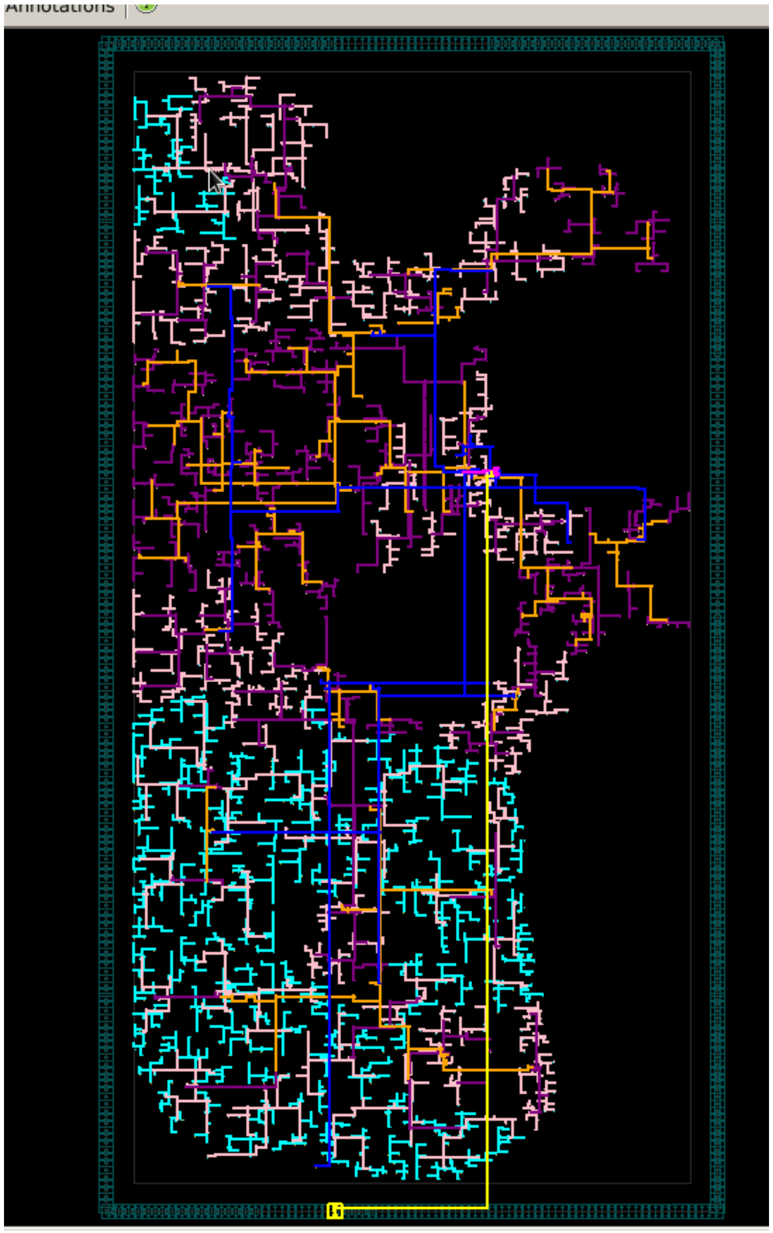


Figure 40: CTS in hierarchical flow

Routing:

Before the routing step, most of designers add spare cells to fix small errors after fabrication if exit, but we didn't put it in our hierarchical design and helped us in reducing DRCs violations.

We started setting route options as discussed in chapter 3. Then we don't have hold violations at this step so we didn't set fix hold before the following step.

First route command: $\boldsymbol{route\_auto}$

This command gives first route iteration with large DRCs violations, large number for short nets and large setup slack violation.

Then we followed with the command $\boldsymbol{route\_opt}$

It reduced shorts and slack violations, therefore we decided to make other iterations using this command

$$\boldsymbol{route\_opt - incremental - effort\ high}$$

We repeated the above command 2 times then reached good slack, hold and shorts at this step

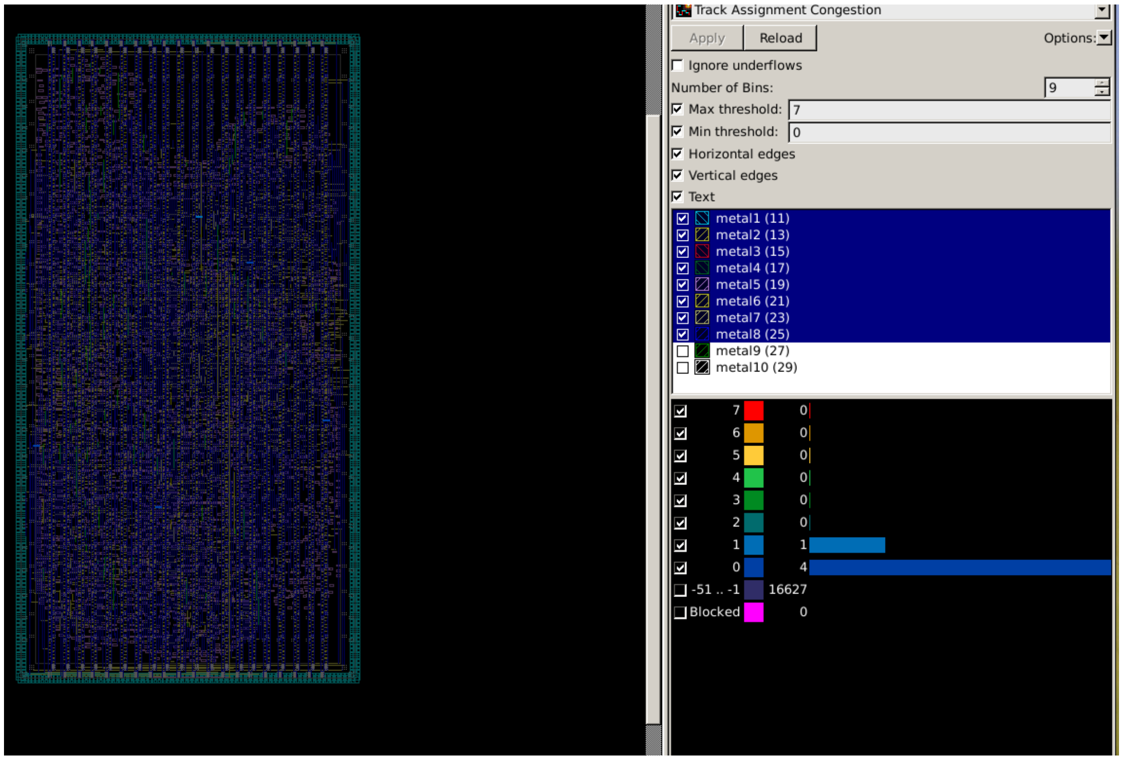
- Short nets = 2
- Slack= -0.02 nsec
- Hold= 0.01 nsec (MET)

Then we connected the modified chip to power grid again.

Note:

After each route step we should verify lvs to check for number of short and open nets which results from large congestion or large max IR drop. Then we check for timing and IR drop

We recognized that IR drop doesn't change too much after the routing commands.


Track Assignment Congestion

Apply Reload Options:

☐ Ignore underflows
Number of Bins: 9
☑ Max threshold: 7
☑ Min threshold: 0
☑ Horizontal edges
☑ Vertical edges
☑ Text

☑ metal1 (11)
☑ metal2 (13)
☑ metal3 (15)
☑ metal4 (17)
☑ metal5 (19)
☑ metal6 (21)
☑ metal7 (23)
☑ metal8 (25)
☐ metal9 (27)
☐ metal10 (29)

☑	7	🟥	0
☑	6	🟧	0
☑	5	🟨	0
☑	4	🟩	0
☑	3	🟩	0
☑	2	🟦	0
☑	1	🟦	1
☑	0	🟦	4
☐	-51 .. -1	🟪	16627
☐	Blocked	🟪	0

Figure 41: congestion at track assignment in hierarchical flow

Finishing

Here we insert filler cells and redundant vias then make our final checks for lvs and timing

We found that:

- Short nets = 0
- Open nets=0
- DRCs violations=1
- Floating nets=0
- Slack= -0.02 nsec
- Hold= 0.01 nsec (MET)

There exist floating ports but all of them are output ports.

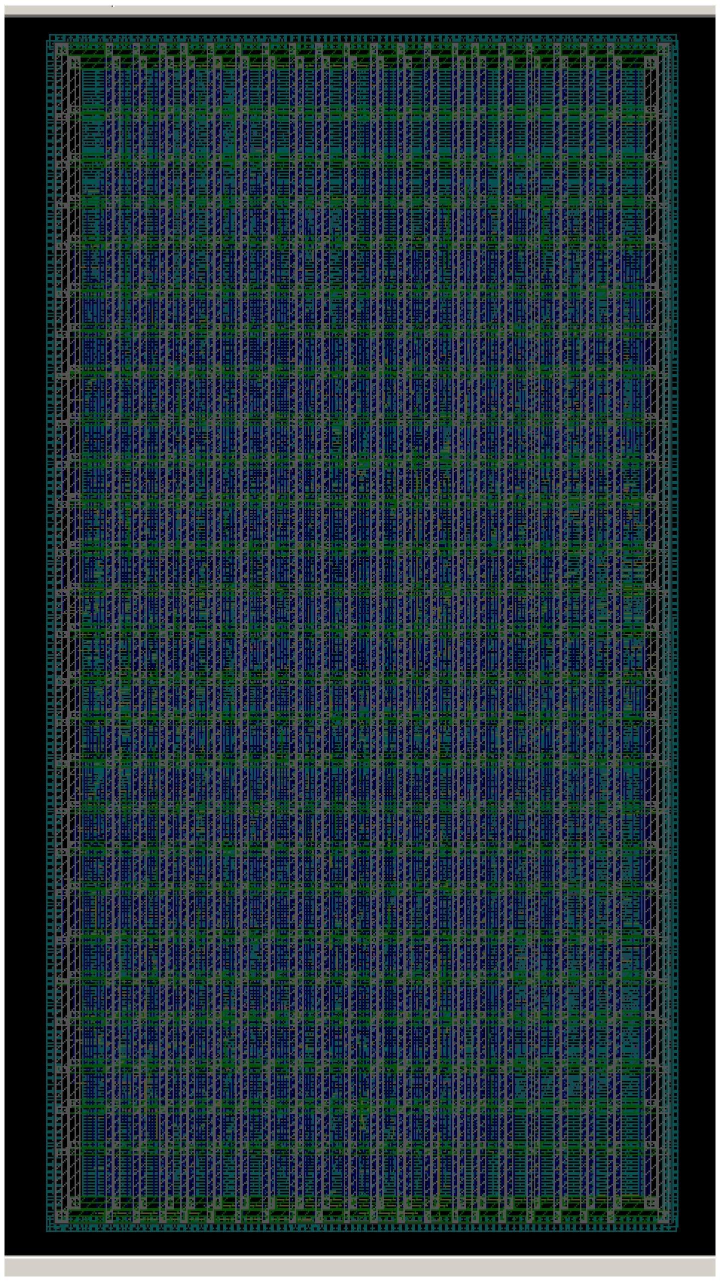


Figure 42: after finishing the hierarchical flow

Extraction

we used ***extract_rc*** command in ICC tool and wrote the spef file.

```
write_parasitics − output {./output/hierarch.spef}

create_rail_setup
```

There is another and more accurate method which is StarRc tool, we tried it also and, in our case, it gives us the same values in PrimeTime compared to extract_rc command.

Note:

In our case, to use the StarRC tool we had to change reading name of the bus using the following commands:

```
define_name_rules new_verilog − special verilog
                − target_bus_naming_style {%s[%d]}
                − check_internal_net_name − check_bus_indexing

change_names − rule new_verilog – hierarchy
```

To match the StarRC tool.

5.3. STA in hierarchical flow

we read the parasitic files in PrimeTime tool, we read the slack by using library at slow slow corner and maximum capacitance spef file, and read the hold time at fast fast corner library and minimum capacitance spef file.

The final timing results:

- Setup Slack= 0.01 nsec.
- Hold time= 0.02 nsec.

Chapter 6

Topographical flow

Topographical technology enables you to accurately predict post-layout timing, area, and power during RTL synthesis without the need for wireload model-based timing approximations. It uses Synopsys' placement and optimization technologies to drive accurate timing prediction within synthesis, ensuring better correlation to the final physical design.

In ultra-deep submicron designs, interconnect parasitic have a major effect on path delays; accurate estimates of resistance and capacitance are necessary to calculate path delays. In topographical mode, Design Compiler leverages the Synopsys physical implementation solution to derive the "virtual layout" of the design so that the tool can accurately predict and use real net capacitances instead of wire load model-based statistical net approximations. If wire load models are present, they are ignored.

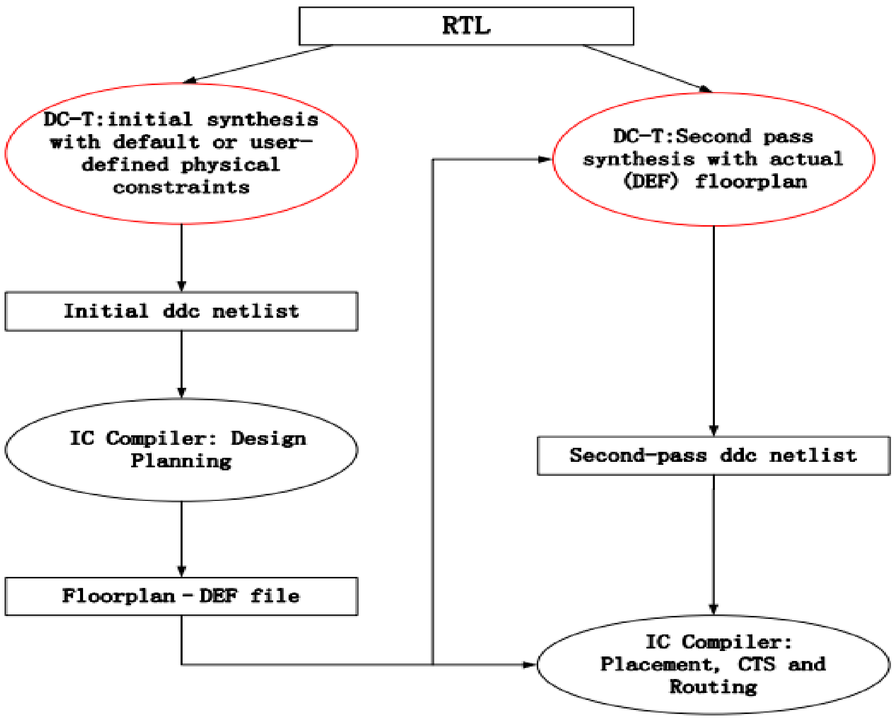

RTL
DC-T:initial synthesis with default or user-defined physical constraints
Initial ddc netlist
IC Compiler: Design Planning
Floorplan - DEF file
DC-T:Second pass synthesis with actual (DEF) floorplan
Second-pass ddc netlist
IC Compiler: Placement, CTS and Routing

Figure 43: Topographical flow

6.1. Synthesis in Topographical flow

Synthesis is the process of converting the RTL design into a gate-level netlist. It consists mainly from three

Steps: translation, optimization and mapping.

First of all, we perform first-pass synthesis depending on flat synthesis, where flat designs contain no subdesigns and have only one structural level.

- **First-pass synthesis flow:**

It is typically the same as synthesis flow that is explained previously in synthesis section.

The recommended compile flow in topographical mode is top down.

- **Floorplan for topographical flow:**

After first-pass synthesis, we perform floorplanning and power network and write def file to use it in the second-pass synthesis as explained in PnR section.

The reason for using floorplan constraints in topographical mode is to accurately estimate interconnect parasitics and improve timing with the post-place-and-route tools, such as IC Compiler, by considering floorplanning information during optimizations.

Design Compiler topographical mode supports high-level physical constraints such as die area, core area and shape, port location, macro location and orientation, keep out margins, placement blockages, preroutes, bounds, vias, tracks, voltage areas, and wiring keep outs.

- **Second-pass synthesis flow:**

We enter dc_shell –topo to run second pass synthesis flow.

- First, we read libraries and tlu_plus files.
- Open milkyway library using open_mw_lib command.
- We perform the reading design step in the synthesis flow (analyze & elaborate) and specifying constraints step.

- Then we extract physical constraints from the def file which we wrote before, using **extract_physical_constraints** command.
- Then we synthesized the design using **compile_ultra -spg -timing_high_effort_script -gate_clock** command, we used the option –spg to enable physical guidance in Design Compiler.
- Finally, we completed the flow performing creating netlist step and checking synthesis.

So, during the second pass, it uses floorplan constraints in Design Compiler topographical mode to create an optimized netlist and performs detailed design closure in IC Compiler.

So as shown in the figure the inputs of second-pass flow are:

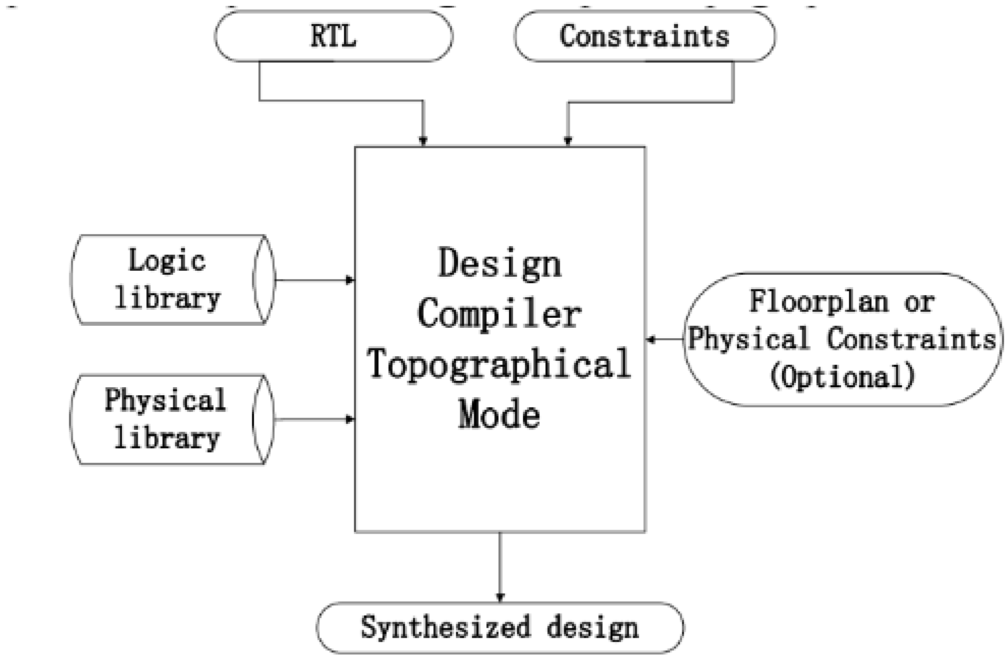

RTL
Constraints
Logic library
Physical library
Design Compiler Topographical Mode
Floorplan or Physical Constraints (Optional)
Synthesized design

Figure 44: inputs for the second pass flow

Here are some trials that we have tried in synthesis in the topographical flow:

All the trials have been done with floorplan with utilization 0.25, core aspect ratio 2 and power in 2 layers (metal9, metal10).

Clock period=5ns

Firstly, we synthesized the design with the clock period (5 ns) that is provided in the constraints file.

a. **First-pass synthesis:**

Using **compile_ultra**

Table 13: the synthesis results of the first path in the Topographical Flow

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
0 (Met)	0.09	1.9156 mW	140.1103 uW	28440.720561

b. **Second-pass synthesis**

Using compile_ultra -spg

Table 14: the synthesis results of the second path in the Topographical Flow

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
0 (Met)	0.07	2.000e+03 uW	1.537e+05 nW	29985.914563

Clock Period 2.5 ns

Here, we reduced clock period to 2.5 ns as our target in the project to increase the speed of Ri5cy.

a. First-pass synthesis

Using compile_ultra

Table 15: the synthesis results of the first path in the Topographical Flow at 2.5 nsec.

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
-0.1	0.09	3.8697 mW	159.9576 uW	31607.450477

b. Second-pass synthesis

Using compile_ultra -spg

Table 16: the synthesis results of the second path in the Topographical Flow at 2.5 nsec.

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
0(Met)	0.07	4.045e+03 uW	1.652e+05 nW	31831.422533

Clock Period 2.5 ns

Here, we tried to optimize timing by using the **timing_high_effort_script** option.

a. First-pass synthesis

Using **compile_ultra -timing_high_effort_script**

Table 17: the synthesis results of the first path in the Topographical Flow at 2.5 nsec at timing_high_effort_script

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
-0.03	0.09	3.9285 mW	161.3372 uW	31701.880461

b. Second-pass synthesis

Using **compile_ultra -spg -timing_high_effort_script**

Table 18: the synthesis results of the second path in the Topographical Flow at 2.5 nsec at timing_high_effort_script

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
0 (Met)	0.07	4.033e+03 uW	1.623e+05 nW	31585.638529

Clock Period 2.5 ns

Here, we tried to perform more optimization in timing by using the **timing_high_effort_script** and **retime** options.

a. First-pass synthesis

Using **compile_ultra -timing_high_effort_script -retime**

Table 19: the synthesis results of the first path in the Topographical Flow at 2.5 nsec at timing_high_effort_script -retime

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
-0.03	0.09	4.0566 mW	162.0495 uW	31977.722479

b. Second-pass synthesis

Using **compile_ultra -spg -timing_high_effort_script -retime**

Table 20: the synthesis results of the first path in the Topographical Flow at 2.5 nsec at timing_high_effort_script -retime

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
0 (Met)	0.07	4.465e+03 uW	1.711e+05 nW	33406.674571

Final Results

Finally, we get the best results after using **incremental** option.

a. First-pass synthesis

Table 21: final results of the synthesis of the first pass in the Topographical Flow

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
-0.02	0.09	3.9085 mW	160.8004 uW	31692.836457

b. Second-pass synthesis

Table 22: final results of the synthesis of the second pass in the Topographical Flow

Setup Slack	Hold Slack	Total Dynamic power	Total Leakage power	Cell area
0	0.07	4.344e+03 uW	1.637e+05 nW	31768.646537

We have tried to make the power in 4 layers (metal7, metal8, metal9, metal10), but after going through pnr, we found the best results for both timing and short nets while using the 2 layers.

6.2. PnR in Topographical flow

In this section, we describe in detail the PnR stages in the topographical mode. After first pass synthesis, we open IC Compiler and go through the floorplanning and power planning stage. Then, write the def file and go back to Design Compiler, but in topographical mode to perform the second pass synthesis. After second pass synthesis, we have the final netlist and the floorplan file that contains the physical guidance information that is used by the IC Compiler to perform placement optimization. Finally, we go back to IC Compiler to perform the remaining PnR flow starting from the placement and going through CTS and routing.

Overview of the PnR Topographical Flow:

– **Floor planning and Powerplanning:**

We started with utilization 0.3, aspect ratio = 2, and routing layers from layer 1 to layer 6. As layers from 7 to 10 have the power rings. Layer 6 is the tap layer as it is the last layer having power n=mesh and standard cells can be routed in it also. Tap layer also should be a vertical layer to make sure that it goes through all power rails.

After multiple iterations, we found that utilization = 0.25, improves both timing and DRCs. Also, we reduced the power ring layers to 2 layers instead of 4. To reduce the number of DRCs, and help the tool eliminate shorts and opens from the design by increasing the routing resources.

– **Placement:**

In topographical flow, the main placement of the cells happens in the design compiler in topo mode during the second pass synthesis as it uses accurate estimation of parasitics and delays without the need for wire-load models. Therefore, Topographical mode uses Synopsys' placement and optimization technologies to drive accurate timing prediction within synthesis, ensuring better correlation to the final physical design.

In IC Compiler, we started by **place_opt -spg** to enable the physical guidance during the placement optimization.

– **CTS:**

After completing the placement stage with accepted congestion, setup and hold slacks (~0ns slack), we go to the next stage which is the clock synthesis tree. And as we discussed in section (), the main goal of CTS stage is to build a buffer/inverter network in order to balance the relative delays and optimize the global skew of the clock domain. There is no difference between CTS in topographical flow, and in other flows. Clock synthesis tree was built based on buffers, in metal layers 3, 4, and 5.

a. Constraints:

- Clock synthesis tree was built based on buffers, in metal layers 3, 4, and 5.
- target_early_delay 0.1
- target_skew 0.5
- max_capacitance 300
- max_fanout 10
- max_transition 0.15

- We used non-Default clock routing to make the clock nets have double width and double spacing. This makes the clock routes less sensitive to cross talk or electromigration effects as non-default rules are used to "harden" the clock. Making the nets have double-width, increases the threshold current for EM, and as the clock nets are the highest in switching activity, they have the highest power and current going through them. Therefore, it is very important to make these nets have double width.

- **Synhtesis, Optimization**

At this point, we are ready to build our clock buffer tree and optimize it using the **clock_opt** command.

- **Routing**

After optimizing the clock buffer tree, we use command **route_group -all_clock_nets** to route it before going to the standard cells routing stage.

– **Routing:**

Before starting to route the standard cells, we insert spare cells:

$$insert\_spare\_cells - lib\_cell \{NOR2\_X4\ NAND2\_X4\} \backslash$$

$$-num\_instances\ 20 \backslash$$

$$-cell\_name\ SPARE\_PREFIX\_NAME \backslash$$

$$-tie$$

Spare cells are main cells that any logic can be implemented with like NAND and NOR cells. They are added to the chip cells before being fabricated to make sure that if something wrong happens in the chip and it is needed to be refabricated, we won't need to do regeneration for the front end of masks again, which costs very much. Because the problem can be solved using the spare cells. However, spare cells are considered as overhead because they consume static power

The results of multiple iterations in the Topographical Flow:

In this section, we describe in more details, the multiple iterations that we did to reach the final results.

- Utilization = 0.3, power in 4 layers, clock = 2.5

Table 23: Results after PnR at Utilization = 0.3, power in 4 layers, clock = 2.5

stage	placement	CTS	Routing
setup	0 (met)	-0.12	-0.46
hold	0.07 (met)	-0.02	0 (violated)

- Ending routing with shorts equal to 461

In this iteration, we ended up with a very high negative setup slack and too many shorts that were hard to optimize and solve. However, creating the power rings in 4 layers makes a very good IR drop, and is very optimized with respect to the power.

It was very hard to optimize these routing results, and after optimization, the results weren't improved in an optimistic way. Therefore, we tried to increase the routing resources.

- Utilization = 0.3, power in 2 layers, clock = 2.5

In order to eliminate negative slack and shorts, we tried to reduce the power metal layers to increase the routing resources for standard cells. We also checked the IR drop to make sure that it is within the acceptable percentage of the VDD and it was accepted.

Table 24: Results after PnR at Utilization = 0.3, power in 2 layers, clock = 2.5

stage	**placement**	**CTS**	**Routing**
setup	0 (met)	-0.12	-0.27
hold	0.07 (met)	0 (violated)	0.01(met)

- Total short nets are 446.

We noticed that there was improvement in the setup slack and the number of shorts reduced, however, increasing the routing layers didn't solve the problem completely.

It was hard to optimize these routing results, and after optimization, the results weren't much better. Therefore, we did multiple iterations and trials.

- Utilization = 0.3, power in 2 layers, clock = 2.5, set_app_var placer_max_cell_density_threshold 0.3

After multiple iterations, we added to our placement script the command set _var placer_max_cell_density_threshold to be equal to 0.3 to control the cells density and distribution.

Table 25: Results after PnR at Utilization = 0.3, power in 4 layers, clock = 2.5 using the placer_max command

stage	**placement**	**CTS**	**Routing**
setup	0(met)	-0.12	-0.22
hold	0.07(met)	0 (met)	0.01(met)

- Total short nets are 397

There was an improvement in both setup slack and number of short nets, also after performing routing optimization we got better results.

<u>**After optimization:**</u>

- Setup: -0.17
- Hold: 0.00 (met)
- Shorts: 4

Topographical Flow Final Results

In this section, we describe in detail the final PnR iteration, and its final results.

- clock = 2.5 ns, which is equivalent to frequency 400 MHz.
- Standard Cells Utilization = 0.25.
- Aspect ratio = 2.
- Power ring in 2 layers.
- Using command set_app_var placer_max_cell_density_threshold 0.3.

Floorplanning:

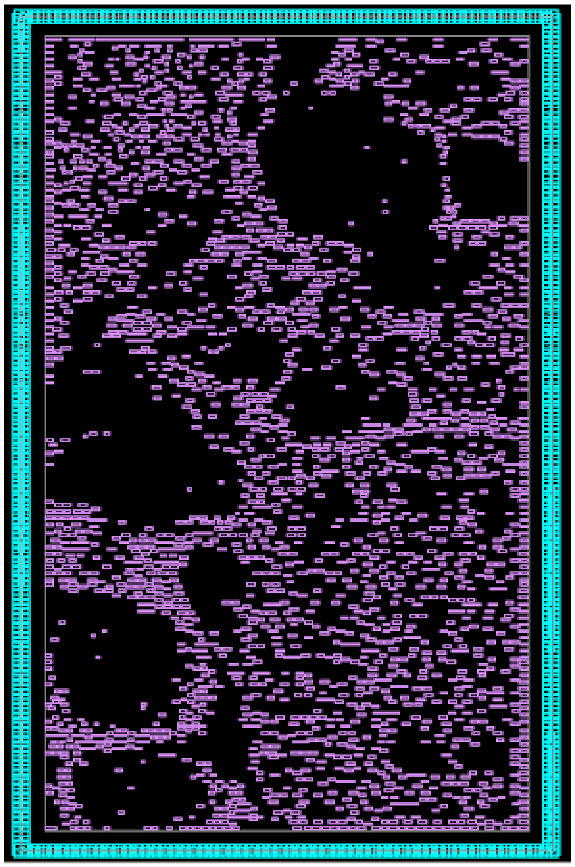
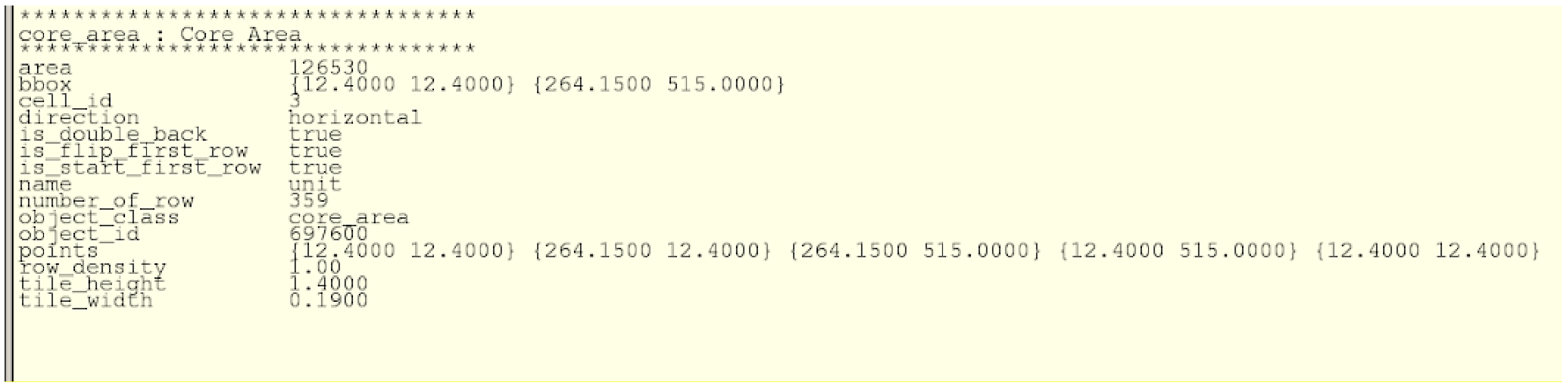


Figure 45: floor plan and virtual placement in topographical flow

Figure 45 shows the floorplan of the chip. And as shown in Figure 45 With core utilization equals 25%, and aspect ratio equals 2, the total area of the chip equals to 126530 The width is 502.6, and the height is 251.75.



```
*********************************
core_area : Core Area
*********************************
area                 126530
bbox                 {12.4000 12.4000} {264.1500 515.0000}
cell_id              3
direction            horizontal
is_double_back       true
is_flip_first_row    true
is_start_first_row   true
name                 unit
number_of_row        359
object_class         core_area
object_id            697600
points               {12.4000 12.4000} {264.1500 12.4000} {264.1500 515.0000} {12.4000 515.0000} {12.4000 12.4000}
row_density          1.00
tile_height          1.4000
tile_width           0.1900
```

Figure 46: core area

Powerplanning:

In order to create the power mesh, we should specify the metal layers where the power ring is built. We chose to create the power ring in the highest two metal layers, metal 9 (vertical) and metal 10 (horizontal).

The other power ring constraints are as follows:

- ring spacing equals to 0.8
- ring width 5
- ring offset 0.8

After this, we specify the constraints for the power rails as follows:

- power rails are in metal layers 8, 9, and 10.
- Metal 8 is the tap layer, and it is vertical as preferred.
- Maximum straps number is 128, and minimum straps number is 20.
- Minimum width is 2.5.
- Minimum spacing between straps.

Therefore, the metal layers are used as following:

- From metal 1 to metal 8 for signal routing.
- Metal 8 is the tap layer.
- Metal 9 and metal 10 for power rings and straps.

The next step is defining the virtual pads, so that the tool can simulate and analyze the power network.

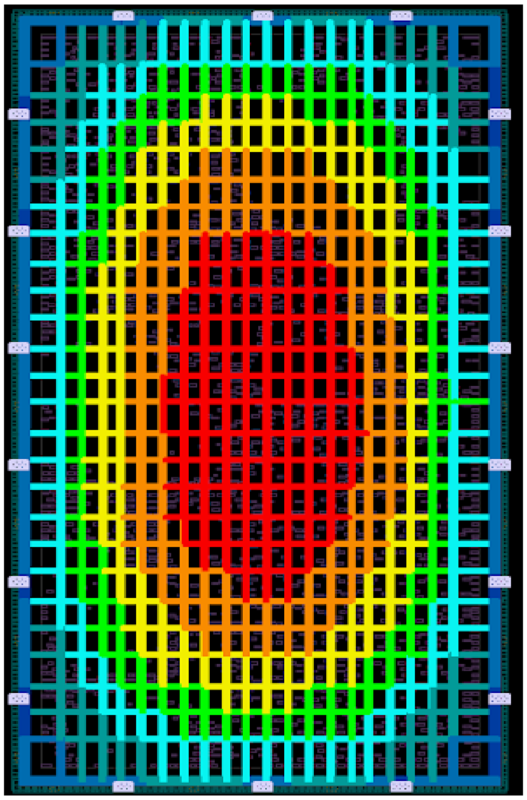
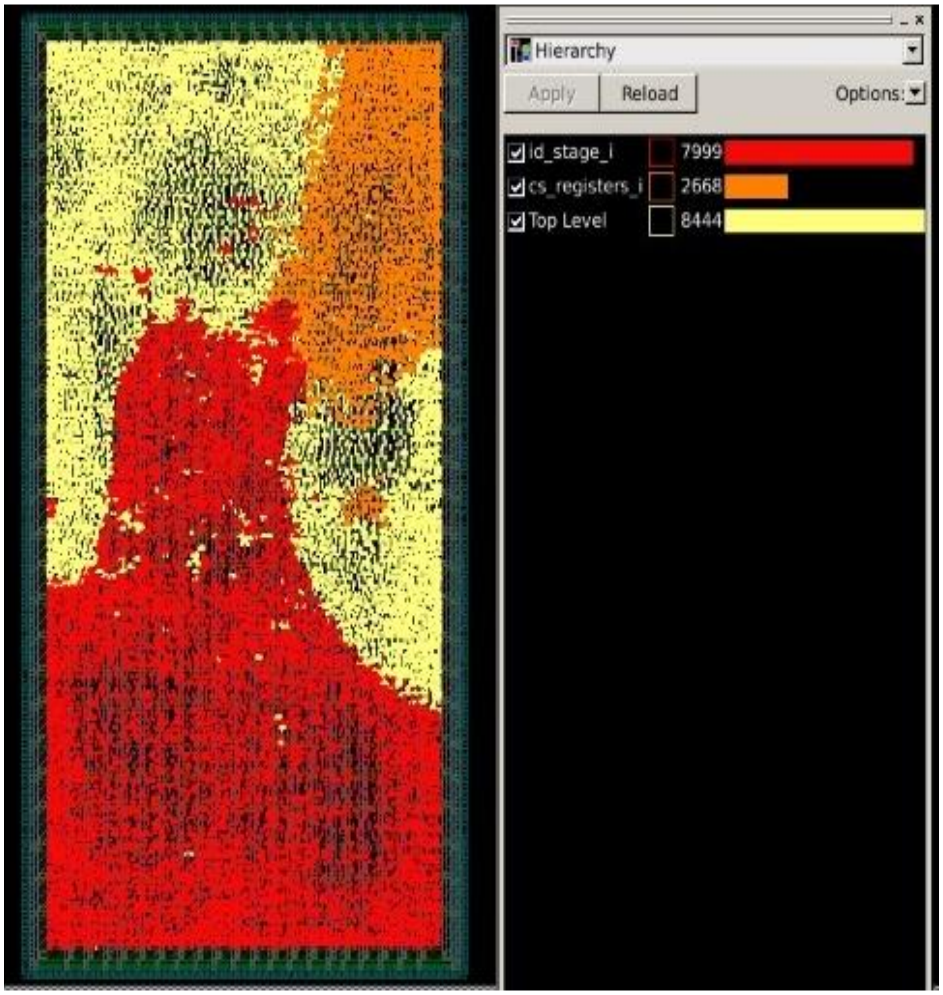


Figure 47: IR drop across the chip in the Topographical Flow

Figure 47 shows the IR drop across the whole chip. PNA is constrained by the voltage supply value as it should be less than 2% from the power supply value. Here the voltage supply is 1.1 v, therefore the max acceptable IR drop equals to 22mv. The max IR drop in our chip is 10.016mv, therefore it is accepted.

The final step in the powerplanning stage is adding the well tie cells.

placement:



☑ id_stage_i	7999	🟥
☑ cs_registers_i	2668	🟧
☑ Top Level	8444	🟨

Figure 48: placement of topographical flow

After second pass synthesis, we go back to IC Compiler to continue the pnr flow with an optimized netlist. Using command place_opt -spg -effort high -congestion. With the -spg option, the place_opt uses Design Compiler's Physical guide information to guide optimization of the placement of standard cells. We added the switch – effort high to force the tool to spend more time to optimize the quality of results (QoR) as the default for place_opt command is medium effort. The congestion switch is for more optimization with respect to congestion in order to make the following stage tasks easier, and to make sure that the routing will end up with a minimum number of DRCs. Figure (0 shows the hierarchy map of the core after the placement of the cells. At the end of the placement stage, the time slacks are as follows:

- The setup slack: - 0.00177
- The hold slack: 0.05362

<u>**Clock Tree Synthesis:**</u>

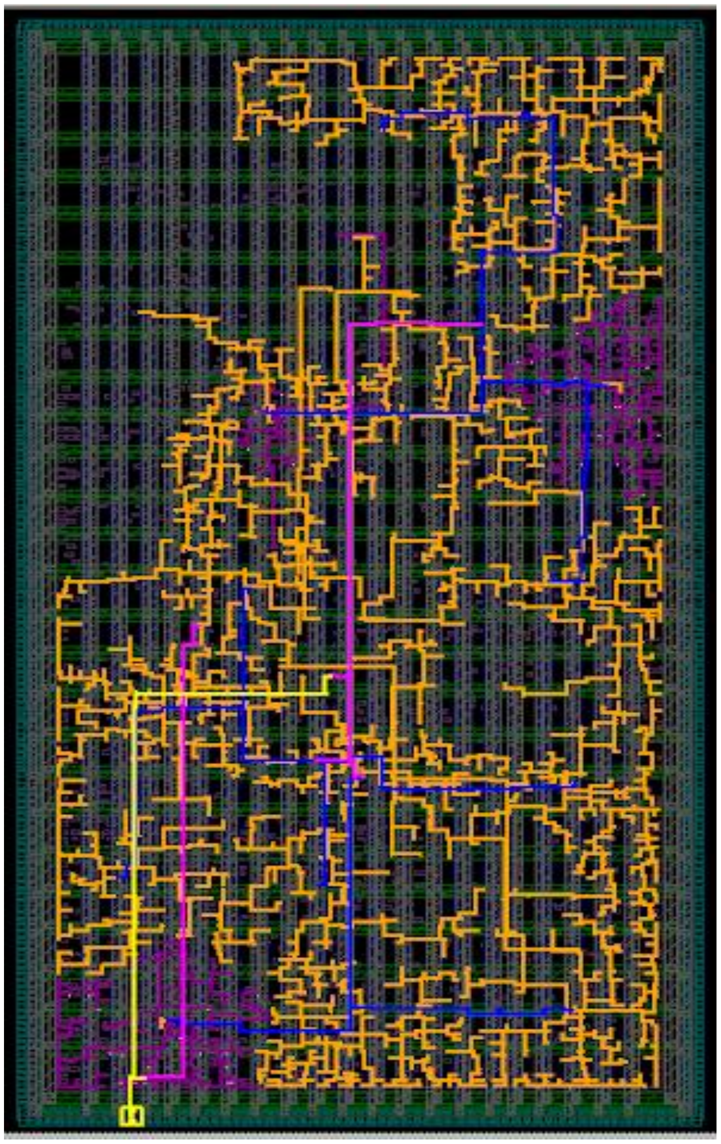


Figure 49: CTS in topographical flow

The figure shows the clock network of the core, Clock network is built across 5 levels. Each level is colored in a different color as shown. Here the clock tree is built exactly as described in section (). At the end of the cts stage, the quality of results is as follows:

- Setup slack: -0.00247
- Hold slack: 0.07092
- Dynamic power: 4.7819 mW
- Leakage power: 183.5204 uW
- Cell area: 34161.848552

Routing

All signal nets are routed during the Routing stage as clock signals are already routed in the CTS stage. The signal nets are routed starting from metal 1 up to metal 8 as discussed in previous sections. Before starting to route the signal nets, we should verify that the design is routable and to have a prediction of the results of the routing stage in order to not consume too much time in the routing of design and resulting in bad results. We use command check_routeability. This command checks pin access points, cell instance wire tracks, pin out of boundaries, and many other things to ensure they meet the design requirements. It performs a check of the design for optimization in order to substantiate any errors in the design that might need to be fixed or what could help to improve the design.

Then, we are ready to route our design and as discussed in section (), routing is performed in four main steps:

1. Global Route
2. Track assignment
3. Detail Route
4. Search and Repair

- The first command we used in routing is route_auto. This command performs the four steps in only one step.
- The second command is route_opt. This command performs simultaneous routing and post-route optimization on the current design.

- The third command is route_opt also, but with switches: -effort high and -incremental. "-effort high" is used to increase the time and the effort of the tool to optimize the routing of the chip. "- incremental" is used to run topology-based incremental optimization and ECO routing.
- In order to solve the remaining shorts, we used the command" route_zrt_detail -incremental true -initial_drc_from_input true".

The final results of the routing stage and the PnR flow are as follows:

- Setup Slack is -0.01ns
- Hold Slack is 0.06ns
- DRCs are 3
- Total short nets are 1
- Total open nets are 0

Chip finishing

Chip finishing is the same as discussed in the other flows, it consists of four main steps:

- Metal layers spreading and widening
- Standard cell filling
- Redundant vias insertion
- Metal filling

6.3. STA in Topographical flow

After chip finishing, we go to the sign off tool for static timing analysis which is Prime Time. We checked the setup slack at the slow slow corner. using command “fix_eco_timing -type setup” to fix violating paths by resizing their cells to have higher speed. After that, we went back to IC Compiler to reroute the nets that needed to be rerouted again due to the changes that happened by primeTime. finally, checked the setup slack again by the primeTime. We found it is a positive slack and equals 0.02ns. Hold slack is checked at the fast fast corner and it is also positive and equals to 0.02ns.

Chapter 7

Results

7.1. Flat flow

7.1.1. Area results

Table 26: The area results of the Flat Flow

	After removing hierarchy	After optimization
Total cell area	32678.366549 um^2	34461.630550 um^2

7.1.2. Power results

Table 27: The power results of the Flat flow

	After removing hierarchy	After optimization
Total leakage power	0.16388 mW	0.17903 mW
Total switching power	0.659 mW	1.3115 mW
Total power	4.3065 mW	5.2742 mW

the Leakage power which is the power consumed in transistor due to the constant current from Vdd to ground and its value is 0.17903 mw which is 3.3944 % of total power. The switching power or dynamic power which is the power consumed in transistors due to switching from 1 to 0 or from 0 to 1 and its value is 1.3115 mW which is 24.86 % of total power.

7.1.3. Timing results

A. <u>After removing hierarchy:</u>

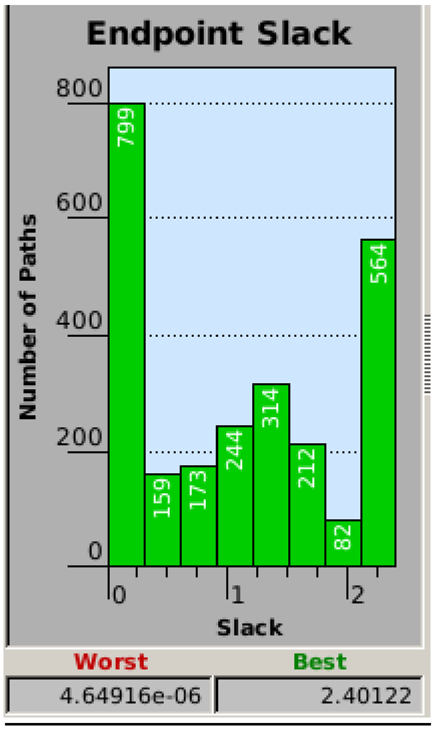


Figure 50: The setup slack after removing hierarch in Flat Flow

the setup slacks of all paths (2547 path) are represented as:

- First column consists of 799 path and is from 0 to 0.286.
- Second column consists of 159 path and is from 0.306 to 0.608.
- Third column consists of 173 path and is from 0.613 to 0.9.
- Forth column consists of 244 path and is from 0.907 to 1.204.
- Fifth column consists of 314 path and is from 1.21 to 1.5.
- Sixth column consists of 212 path and is from 1.51 to 1.8.
- Seventh column consists of 82 path and is from 1.8 to 2.02.
- Eighth column consists of 564 path and is from 2.21 to 2.40122.

B. After PnR:

– Maximum delay timing results:

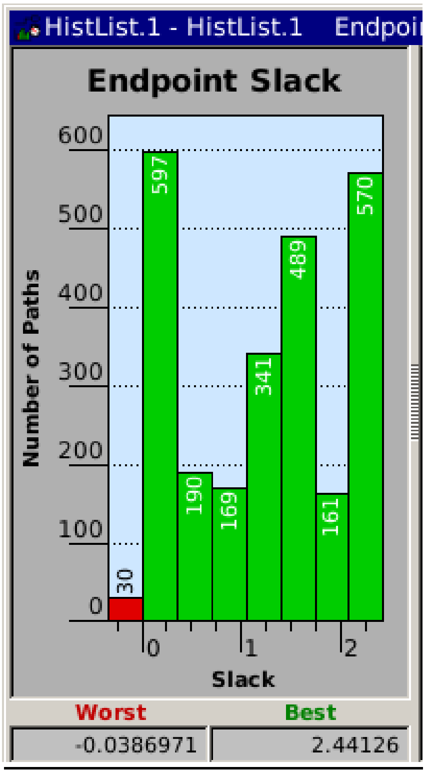
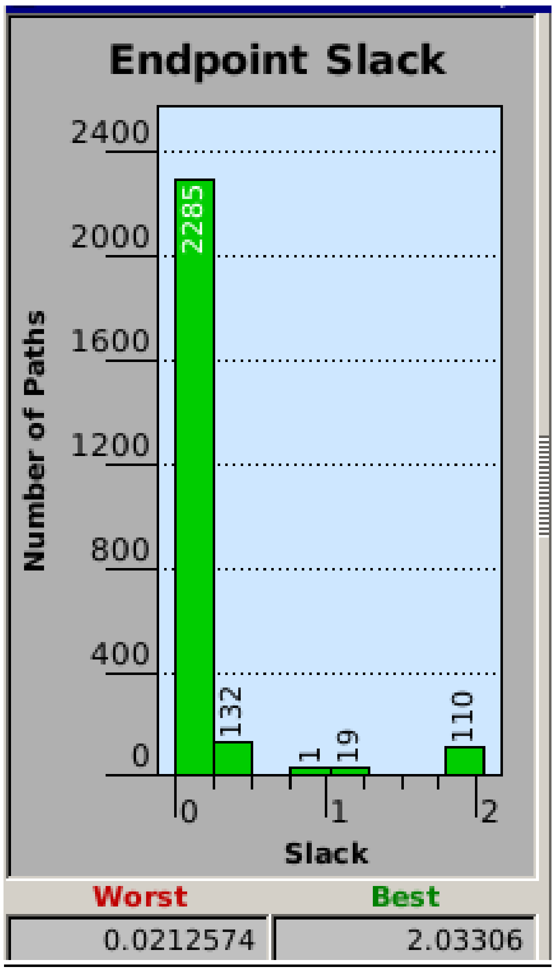


Figure 51: The setup slack after PnR in Flat Flow

As said before, after the routing stage the slack is -0.04 nsec.

- Minimum delay timing results:



0.0212574

2.03306

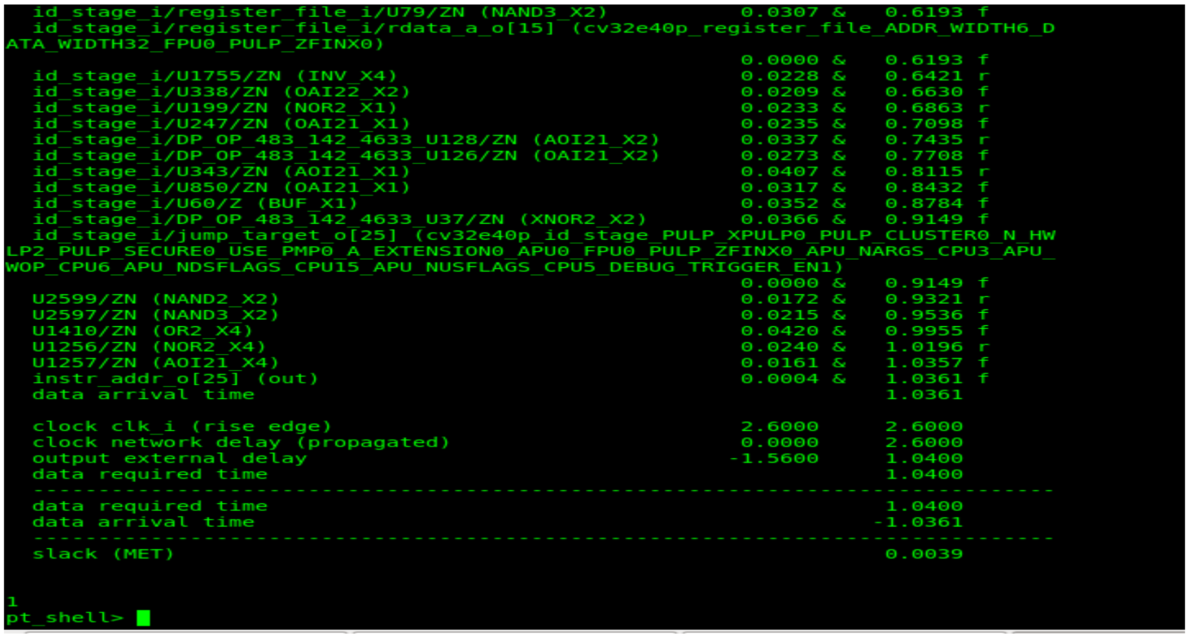
Figure 52: The Hold slack after PnR in Flat Flow

based timing paths, there is no hold time requirements.

C. <u>After optimization:</u>

The design timing was tested using static timing analysis (STA) in primetime. In STA you define the design constraints and then test the design and see if it passes them. These constraints are the same constraints used to build up the design in addition to the parasitics come from layout.

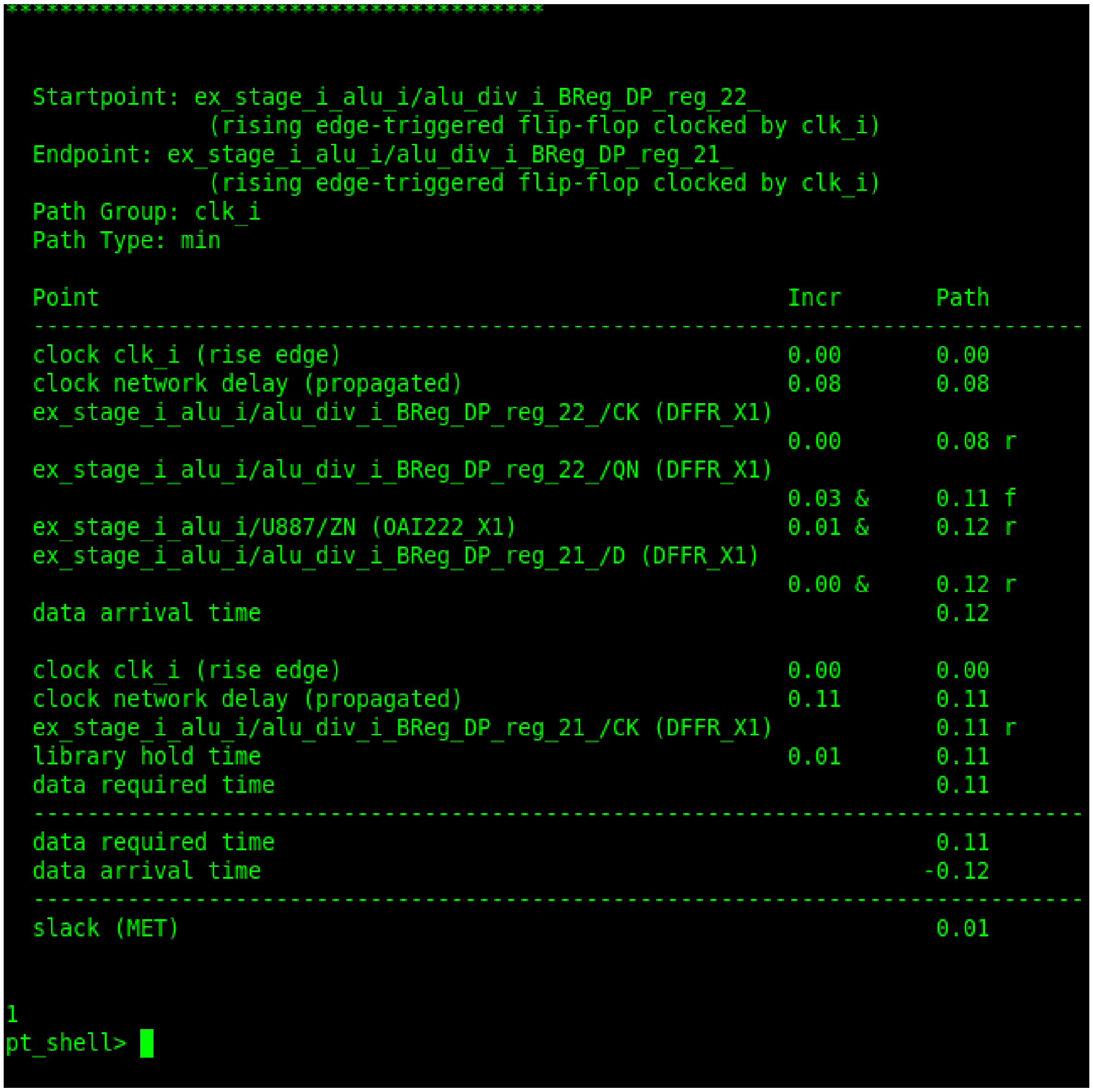
- Maximum delay timing results:



```
  id_stage_i/register_file_i/U79/ZN (NAND3_X2)          0.0307 &    0.6193 f
  id_stage_i/register_file_i/rdata_a_o[15] (cv32e40p_register_file_ADDR_WIDTH6_D
ATA_WIDTH32_FPU0_PULP_ZFINX0)
                                                        0.0000 &    0.6193 f
  id_stage_i/U1755/ZN (INV_X4)                          0.0228 &    0.6421 r
  id_stage_i/U338/ZN (OAI22_X2)                         0.0209 &    0.6630 f
  id_stage_i/U199/ZN (NOR2_X1)                          0.0233 &    0.6863 r
  id_stage_i/U247/ZN (OAI21_X1)                         0.0235 &    0.7098 f
  id_stage_i/DP_OP_483_142_4633_U128/ZN (AOI21_X2)      0.0337 &    0.7435 r
  id_stage_i/DP_OP_483_142_4633_U126/ZN (OAI21_X2)      0.0273 &    0.7708 f
  id_stage_i/U343/ZN (AOI21_X1)                         0.0407 &    0.8115 r
  id_stage_i/U850/ZN (OAI21_X1)                         0.0317 &    0.8432 f
  id_stage_i/U60/Z (BUF_X1)                             0.0352 &    0.8784 f
  id_stage_i/DP_OP_483_142_4633_U37/ZN (XNOR2_X2)       0.0366 &    0.9149 f
  id_stage_i/jump_target_o[25] (cv32e40p_id_stage_PULP_XPULP0_PULP_CLUSTER0_N_HW
LP2_PULP_SECURE0_USE_PMP0_A_EXTENSION0_APU0_FPU0_PULP_ZFINX0_APU_NARGS_CPU3_APU_
WOP_CPU6_APU_NDSFLAGS_CPU15_APU_NUSFLAGS_CPU5_DEBUG_TRIGGER_EN1)
                                                        0.0000 &    0.9149 f
  U2599/ZN (NAND2_X2)                                   0.0172 &    0.9321 r
  U2597/ZN (NAND3_X2)                                   0.0215 &    0.9536 f
  U1410/ZN (OR2_X4)                                     0.0420 &    0.9955 f
  U1256/ZN (NOR2_X4)                                    0.0240 &    1.0196 r
  U1257/ZN (AOI21_X4)                                   0.0161 &    1.0357 f
  instr_addr_o[25] (out)                                0.0004 &    1.0361 f
  data arrival time                                                 1.0361

  clock clk_i (rise edge)                               2.6000      2.6000
  clock network delay (propagated)                      0.0000      2.6000
  output external delay                                -1.5600      1.0400
  data required time                                                1.0400
  ---------------------------------------------------------------------------
  data required time                                                1.0400
  data arrival time                                                -1.0361
  ---------------------------------------------------------------------------
  slack (MET)                                                       0.0039


1
pt_shell>
```

Figure 53: The setup slack after PrimeTime in Flat Flow

– Minimum delay timing results:



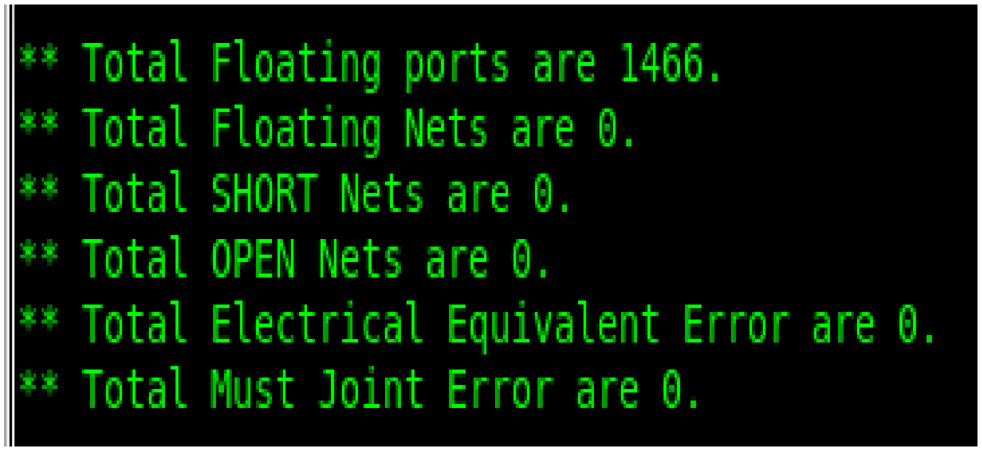
```
****************************************


  Startpoint: ex_stage_i_alu_i/alu_div_i_BReg_DP_reg_22_
              (rising edge-triggered flip-flop clocked by clk_i)
  Endpoint: ex_stage_i_alu_i/alu_div_i_BReg_DP_reg_21_
              (rising edge-triggered flip-flop clocked by clk_i)
  Path Group: clk_i
  Path Type: min

  Point                                                    Incr       Path
  ---------------------------------------------------------------------------
  clock clk_i (rise edge)                                  0.00       0.00
  clock network delay (propagated)                         0.08       0.08
  ex_stage_i_alu_i/alu_div_i_BReg_DP_reg_22_/CK (DFFR_X1)
                                                           0.00       0.08 r
  ex_stage_i_alu_i/alu_div_i_BReg_DP_reg_22_/QN (DFFR_X1)
                                                           0.03 &     0.11 f
  ex_stage_i_alu_i/U887/ZN (OAI222_X1)                     0.01 &     0.12 r
  ex_stage_i_alu_i/alu_div_i_BReg_DP_reg_21_/D (DFFR_X1)
                                                           0.00 &     0.12 r
  data arrival time                                                   0.12

  clock clk_i (rise edge)                                  0.00       0.00
  clock network delay (propagated)                         0.11       0.11
  ex_stage_i_alu_i/alu_div_i_BReg_DP_reg_21_/CK (DFFR_X1)             0.11 r
  library hold time                                        0.01       0.11
  data required time                                                  0.11
  ---------------------------------------------------------------------------
  data required time                                                  0.11
  data arrival time                                                  -0.12
  ---------------------------------------------------------------------------
  slack (MET)                                                         0.01


1
pt_shell>
```

Figure 54: The Hold slack after PrimeTime in Flat Flow

Table 28: The timing Results afte PrimeTime in Flat Flow

	Setup	Hold
Primetime	0.0039 (MET)	0.01 (MET)

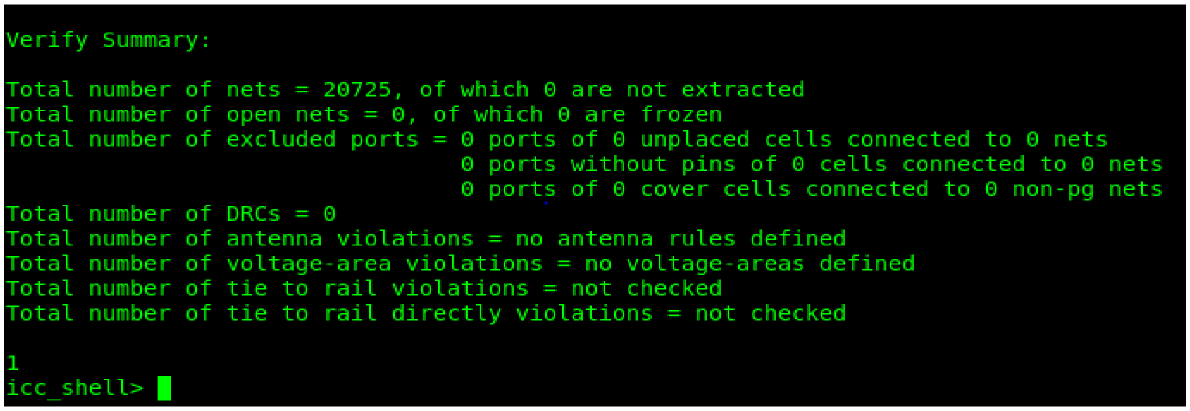
7.1.4. LVS results



```
** Total Floating ports are 1466.
** Total Floating Nets are 0.
** Total SHORT Nets are 0.
** Total OPEN Nets are 0.
** Total Electrical Equivalent Error are 0.
** Total Must Joint Error are 0.
```

Figure 55: The LVS results of Flat Flow

7.1.5. DRC results



```
Verify Summary:

Total number of nets = 20725, of which 0 are not extracted
Total number of open nets = 0, of which 0 are frozen
Total number of excluded ports = 0 ports of 0 unplaced cells connected to 0 nets
                                 0 ports without pins of 0 cells connected to 0 nets
                                 0 ports of 0 cover cells connected to 0 non-pg nets
Total number of DRCs = 0
Total number of antenna violations = no antenna rules defined
Total number of voltage-area violations = no voltage-areas defined
Total number of tie to rail violations = not checked
Total number of tie to rail directly violations = not checked

1
icc_shell>
```

Figure 56: The DRC results of Flat Flow

7.2. Hierarchical flow

7.2.1. Area results

Table 29: The total area results of the Hierarchical Flow

	Post-synthesis	Post-PNR
Total cell area	31598.406520 um^2	33082.9525 um^2

7.2.2. Power results

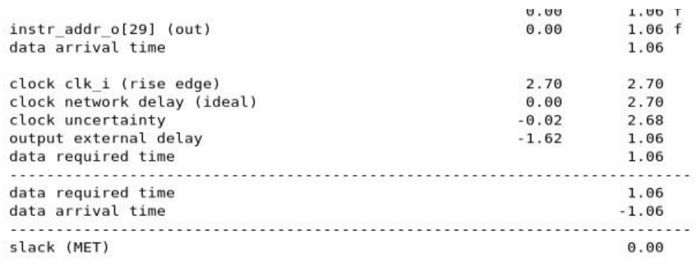
Table 30: The power results of the Hierarchical flow

	Post-synthesis	Post-PNR
Total leakage power	0.161649 mW	0.17506 mW
Total switching power	0.62509 mW	1.2604 mW
Total power	3.6707 mW	4.8474 mW

the Leakage power which is the power consumed in transistor due to the constant current from Vdd to ground and its value is 0. 161649 mw which is 4.403 % of total power. The switching power or dynamic power which is the power consumed in transistors due to switching from 1 to 0 or from 0 to 1 and its value is 0.62509 mW which is 17.029 % of total power.

7.2.3. Timing results

A. Post synthesis



```
                                                      0.00       1.06 f
instr_addr_o[29] (out)                                0.00       1.06 f
data arrival time                                                1.06

clock clk_i (rise edge)                               2.70       2.70
clock network delay (ideal)                           0.00       2.70
clock uncertainty                                    -0.02       2.68
output external delay                                -1.62       1.06
data required time                                               1.06
-----------------------------------------------------------------------------
data required time                                               1.06
data arrival time                                               -1.06
-----------------------------------------------------------------------------
slack (MET)                                                      0.00
```

Figure 57: Setup timing after synthesis

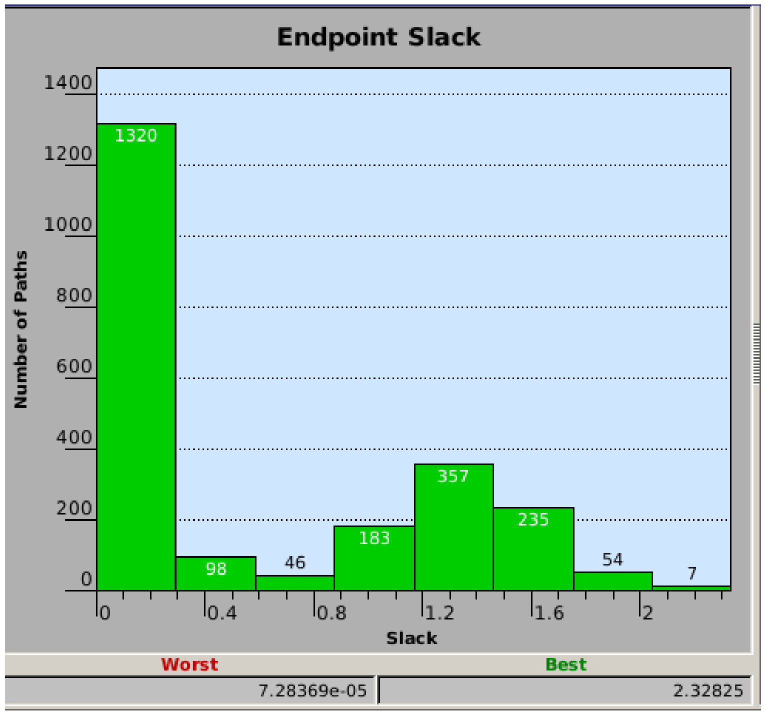


Figure 58: Endpoint lack for max timing

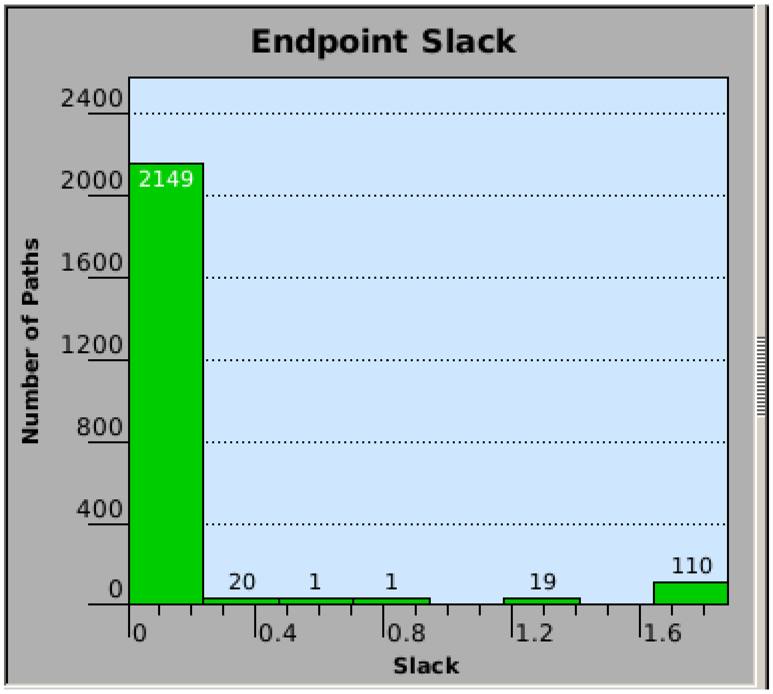
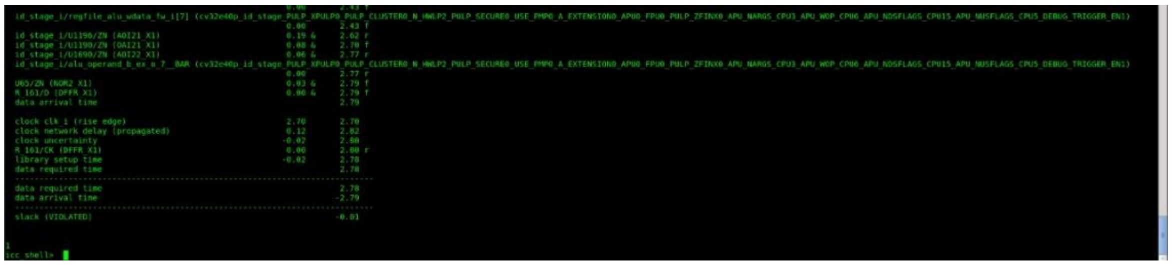


Figure 59: Endpoint slacks for min timing

Table 31: The timing results after synthesis of The Hierarchical Flow

	Setup	Hold
First compile iteration	-0.04 (VIOLATED)	0.07 (MET)
Fourth compile iteration	0.00 (MET)	0.00 (MET)

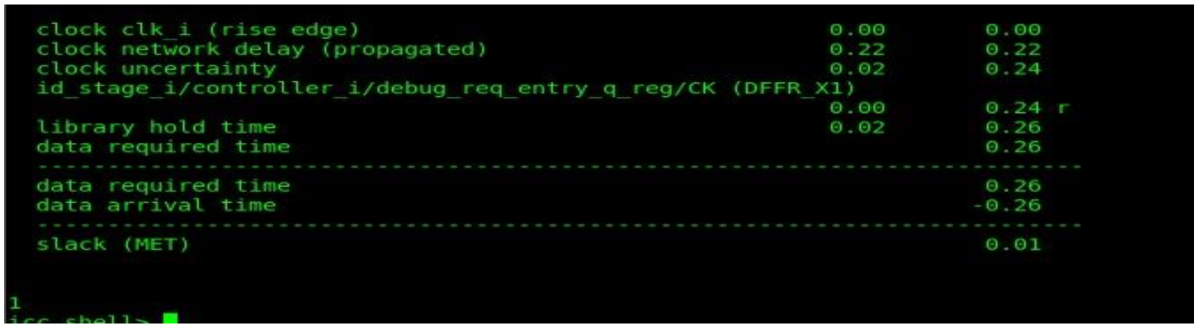
B. PNR stages:



```
                                                          0.00         2.43 f
  id_stage_i/regfile_alu_wdata_fw_i[7] (cv32e40p_id_stage_PULP_XPULP0_PULP_CLUSTER0_N_HWLP2_PULP_SECURE0_USE_PMP0_A_EXTENSION0_APU0_FPU0_PULP_ZFINX0_APU_NARGS_CPU3_APU_WOP_CPU6_APU_NDSFLAGS_CPU15_APU_NUSFLAGS_CPU5_DEBUG_TRIGGER_EN1)
                                                          0.00         2.43 f
  id_stage_i/U1196/ZN (AOI21_X1)                          0.19 &       2.62 r
  id_stage_i/U1190/ZN (OAI21_X1)                          0.08 &       2.70 f
  id_stage_i/U1690/ZN (AOI22_X1)                          0.06 &       2.77 r
  id_stage_i/alu_operand_b_ex_o_7__BAR (cv32e40p_id_stage_PULP_XPULP0_PULP_CLUSTER0_N_HWLP2_PULP_SECURE0_USE_PMP0_A_EXTENSION0_APU0_FPU0_PULP_ZFINX0_APU_NARGS_CPU3_APU_WOP_CPU6_APU_NDSFLAGS_CPU15_APU_NUSFLAGS_CPU5_DEBUG_TRIGGER_EN1)
                                                          0.00         2.77 r
  U65/ZN (NOR2_X1)                                        0.03 &       2.79 f
  R_161/D (DFFR_X1)                                       0.00 &       2.79 f
  data arrival time                                                    2.79

  clock clk_i (rise edge)                                 2.70         2.70
  clock network delay (propagated)                        0.12         2.82
  clock uncertainty                                      -0.02         2.80
  R_161/CK (DFFR_X1)                                      0.00         2.80 r
  library setup time                                     -0.02         2.78
  data required time                                                   2.78
  ---------------------------------------------------------------------------
  data required time                                                   2.78
  data arrival time                                                   -2.79
  ---------------------------------------------------------------------------
  slack (VIOLATED)                                                    -0.01


1
icc_shell>
```

Figure 60: Setup timing after PNR stage

.



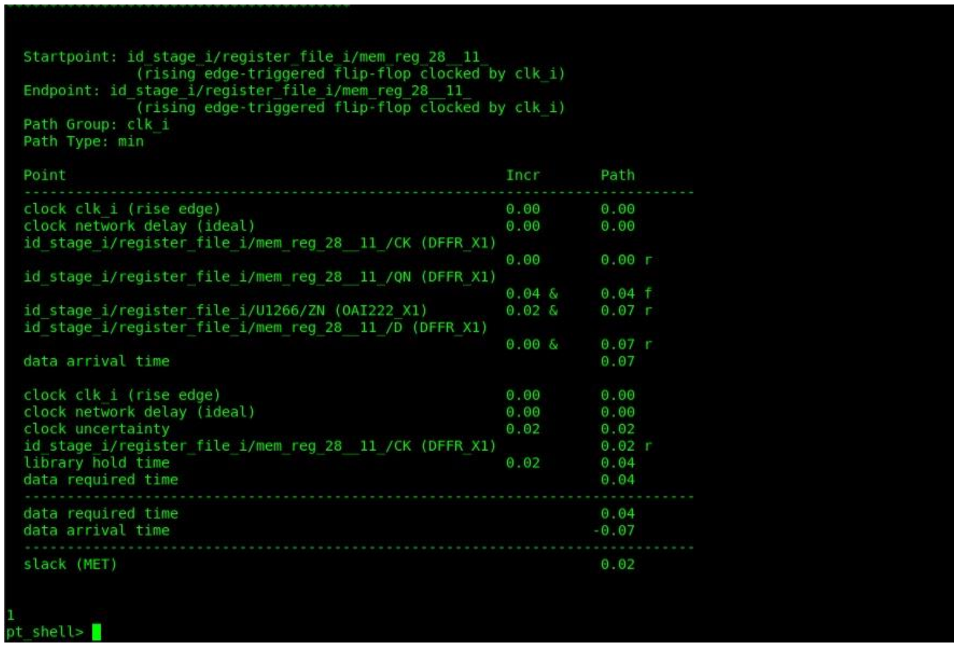
```
  clock clk_i (rise edge)                                 0.00         0.00
  clock network delay (propagated)                        0.22         0.22
  clock uncertainty                                       0.02         0.24
  id_stage_i/controller_i/debug_req_entry_q_reg/CK (DFFR_X1)
                                                          0.00         0.24 r
  library hold time                                       0.02         0.26
  data required time                                                   0.26
  ---------------------------------------------------------------------------
  data required time                                                   0.26
  data arrival time                                                   -0.26
  ---------------------------------------------------------------------------
  slack (MET)                                                          0.01


1
icc_shell>
```

Figure 61: Hold timing after PNR stage.

Table 32: the timing results in the PnR flow of the Hierarchical Flow

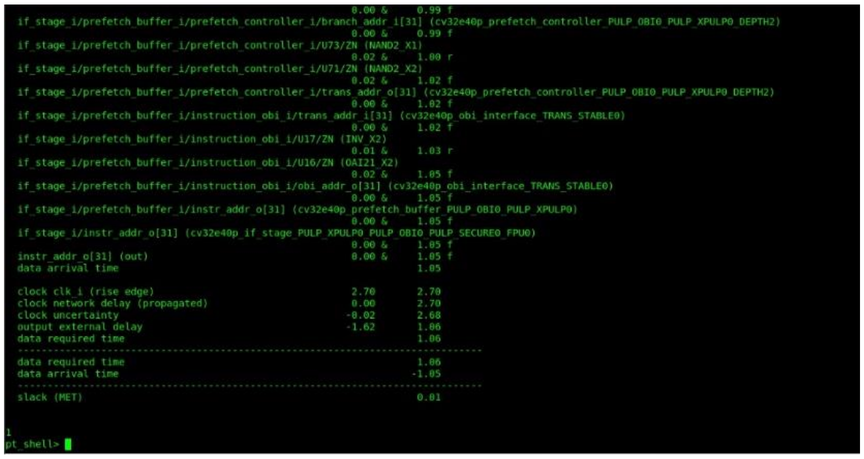
	Setup	Hold
placement	0.00 (MET)	-0.09 (VIOLATED)
CTS	0.02 (MET)	0.01 (MET)
Routing	-0.29 (VIOLATED)	0.01 (MET)
Route_opt (3 iterations)	-0.01 (VIOLATED)	0.01 (MET)
Finishing	-0.01 (VIOLATED)	0.00 (MET)

C. Post-PrimeTime



```
Startpoint: id_stage_i/register_file_i/mem_reg_28__11_
            (rising edge-triggered flip-flop clocked by clk_i)
Endpoint: id_stage_i/register_file_i/mem_reg_28__11_
            (rising edge-triggered flip-flop clocked by clk_i)
Path Group: clk_i
Path Type: min

Point                                                   Incr       Path
-------------------------------------------------------------------------------
clock clk_i (rise edge)                                 0.00       0.00
clock network delay (ideal)                             0.00       0.00
id_stage_i/register_file_i/mem_reg_28__11_/CK (DFFR_X1)
                                                        0.00       0.00 r
id_stage_i/register_file_i/mem_reg_28__11_/QN (DFFR_X1)
                                                        0.04 &     0.04 f
id_stage_i/register_file_i/U1266/ZN (OAI222_X1)         0.02 &     0.07 r
id_stage_i/register_file_i/mem_reg_28__11_/D (DFFR_X1)
                                                        0.00 &     0.07 r
data arrival time                                                  0.07

clock clk_i (rise edge)                                 0.00       0.00
clock network delay (ideal)                             0.00       0.00
clock uncertainty                                       0.02       0.02
id_stage_i/register_file_i/mem_reg_28__11_/CK (DFFR_X1)            0.02 r
library hold time                                       0.02       0.04
data required time                                                 0.04
-------------------------------------------------------------------------------
data required time                                                 0.04
data arrival time                                                 -0.07
-------------------------------------------------------------------------------
slack (MET)                                                        0.02


1
pt_shell>
```

Figure 62: hold timing after PrimeTime



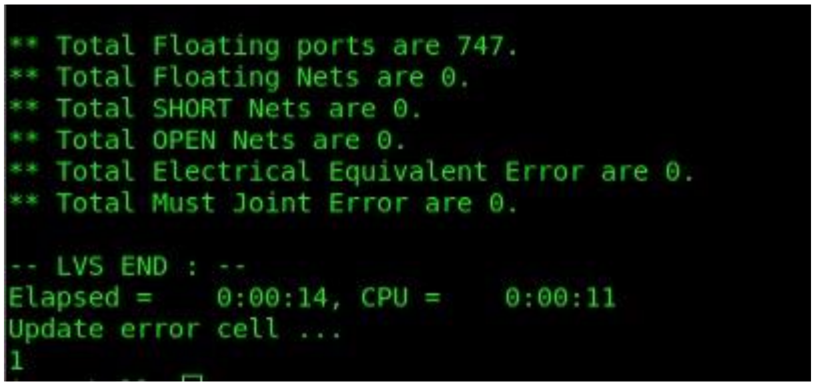
```
                                                        0.00 &     0.99 f
if_stage_i/prefetch_buffer_i/prefetch_controller_i/branch_addr_i[31] (cv32e40p_prefetch_controller_PULP_OBI0_PULP_XPULP0_DEPTH2)
                                                        0.00 &     0.99 f
if_stage_i/prefetch_buffer_i/prefetch_controller_i/U73/ZN (NAND2_X1)
                                                        0.02 &     1.00 r
if_stage_i/prefetch_buffer_i/prefetch_controller_i/U71/ZN (NAND2_X2)
                                                        0.02 &     1.02 f
if_stage_i/prefetch_buffer_i/prefetch_controller_i/trans_addr_o[31] (cv32e40p_prefetch_controller_PULP_OBI0_PULP_XPULP0_DEPTH2)
                                                        0.00 &     1.02 f
if_stage_i/prefetch_buffer_i/instruction_obi_i/trans_addr_i[31] (cv32e40p_obi_interface_TRANS_STABLE0)
                                                        0.00 &     1.02 f
if_stage_i/prefetch_buffer_i/instruction_obi_i/U17/ZN (INV_X2)
                                                        0.01 &     1.03 r
if_stage_i/prefetch_buffer_i/instruction_obi_i/U16/ZN (OAI21_X2)
                                                        0.02 &     1.05 f
if_stage_i/prefetch_buffer_i/instruction_obi_i/obi_addr_o[31] (cv32e40p_obi_interface_TRANS_STABLE0)
                                                        0.00 &     1.05 f
if_stage_i/prefetch_buffer_i/instr_addr_o[31] (cv32e40p_prefetch_buffer_PULP_OBI0_PULP_XPULP0)
                                                        0.00 &     1.05 f
if_stage_i/instr_addr_o[31] (cv32e40p_if_stage_PULP_XPULP0_PULP_OBI0_PULP_SECURE0_FPU0)
                                                        0.00 &     1.05 f
instr_addr_o[31] (out)                                  0.00 &     1.05 f
data arrival time                                                  1.05

clock clk_i (rise edge)                                 2.70       2.70
clock network delay (propagated)                        0.00       2.70
clock uncertainty                                      -0.02       2.68
output external delay                                  -1.62       1.06
data required time                                                 1.06
-------------------------------------------------------------------------------
data required time                                                 1.06
data arrival time                                                 -1.05
-------------------------------------------------------------------------------
slack (MET)                                                        0.01


1
pt_shell>
```

Figure 63: Setup timing after PrimeTime

Table 33: The timing results after PrimeTime of the Hierarchical Flow

	Setup	Hold
Primetime	0.01 (MET)	0.02 (MET)

7.2.4. LVS results



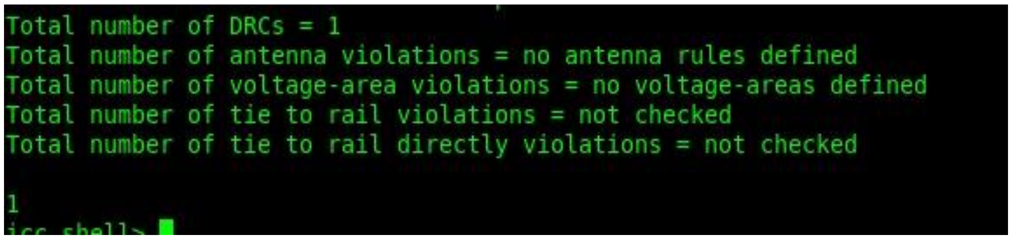
```
** Total Floating ports are 747.
** Total Floating Nets are 0.
** Total SHORT Nets are 0.
** Total OPEN Nets are 0.
** Total Electrical Equivalent Error are 0.
** Total Must Joint Error are 0.

-- LVS END : --
Elapsed =    0:00:14, CPU =    0:00:11
Update error cell ...
1
```

Figure 64: The LVS results of the Hierarchical Flow

Table 34: The LVS results of the Hierarchical Flow

	Shorted nets	Open nets
Route auto	171	0
Route opt (3x)	0	0

7.2.5. DRC results



```
Total number of DRCs = 1
Total number of antenna violations = no antenna rules defined
Total number of voltage-area violations = no voltage-areas defined
Total number of tie to rail violations = not checked
Total number of tie to rail directly violations = not checked

1
icc_shell>
```

Figure 65: DRC ICC results

7.3. Topographical flow

7.3.1. Area results

Table 35: The total area results of the Topographical Flow

	After removing hierarchy	After optimization
Total cell area	31768.646537 um^2	34267.716551 um^2

7.3.2. Power results

Table 36: The power results of the Topographical Flow

	After removing hierarchy	After optimization
Total leakage power	1.6366e+05 nW	1.8474e+05 nW
Total switching power	839.8591 uW	1.2763e+03 uW
Total power	4.5074 mW	5.0174 mW

7.3.3. Timing results

A. <u>After removing hierarchy:</u>

– Maximum delay timing results:

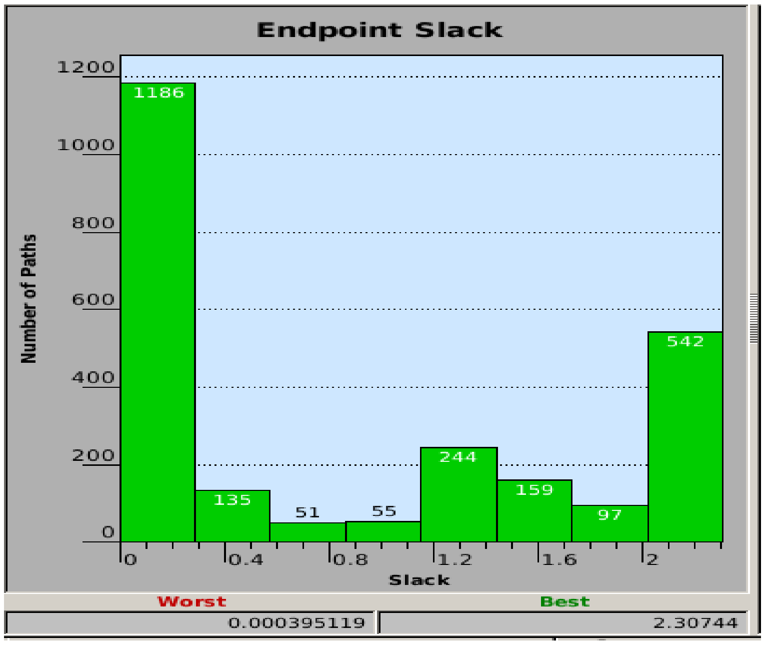


Figure 66: The setup slack after synthesis in Topographical Flow

– Minimum delay timing results:

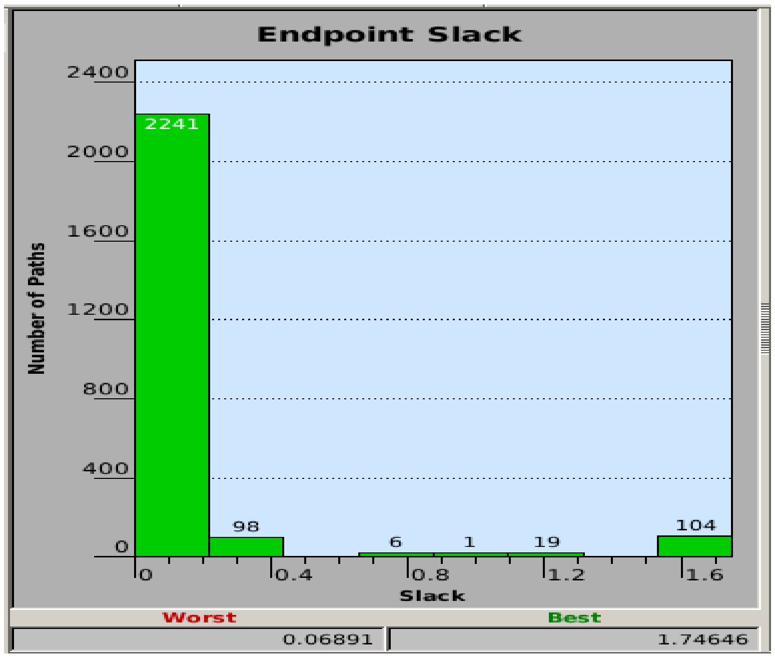
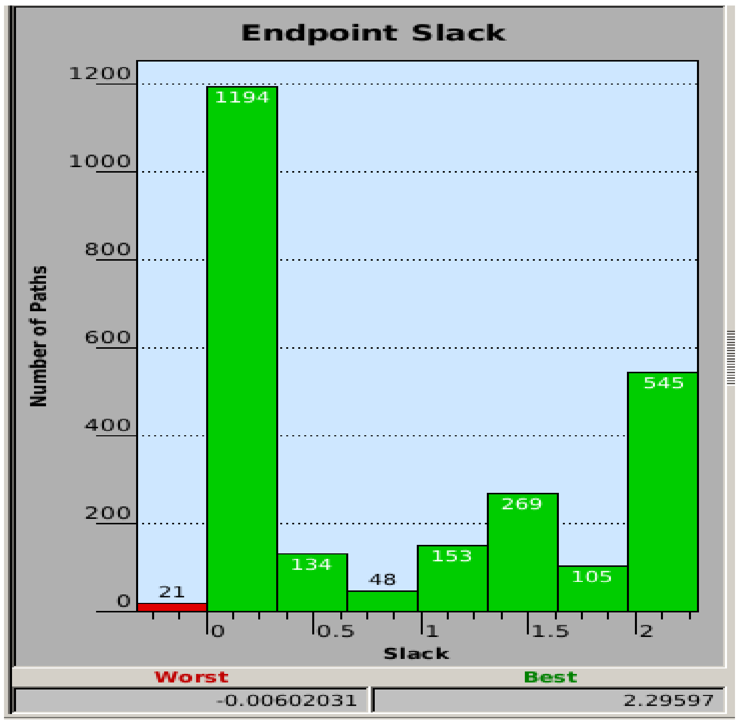


Figure 67: The Hold slack after synthesis in Topographical Flow

B. After PnR

– Maximum delay timing results:



-0.00602031

2.29597

Figure 68: The setup slack after PnR in Topographical Flow

as said before the setup slack -0.01 ns.

– Minimum delay timing results:

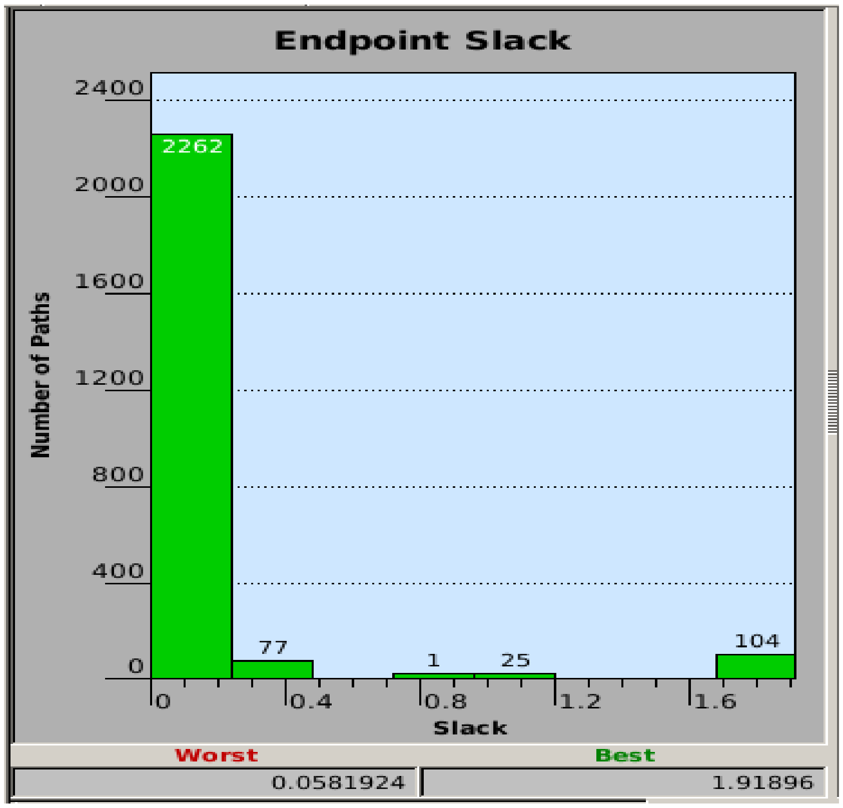
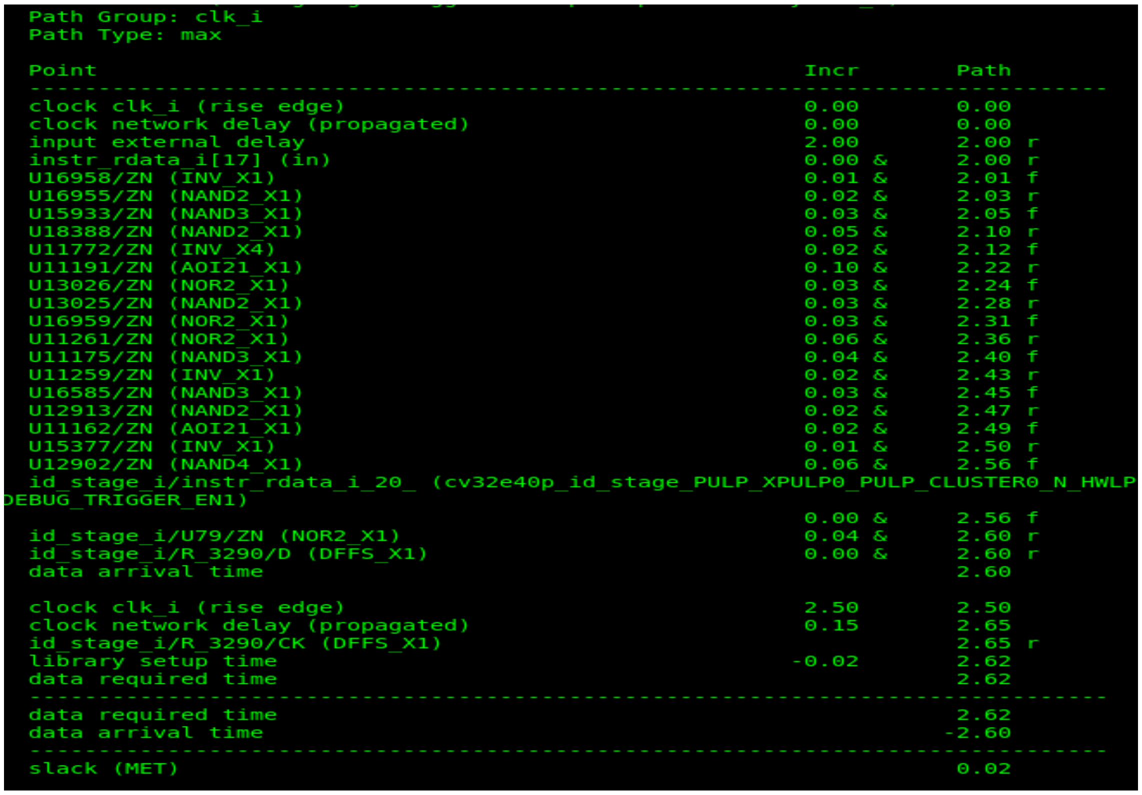


Figure 69: The Hold slack after PnR in Topographical Flow

as said before the hold slack 0.06 ns.

C. **After optimization:**

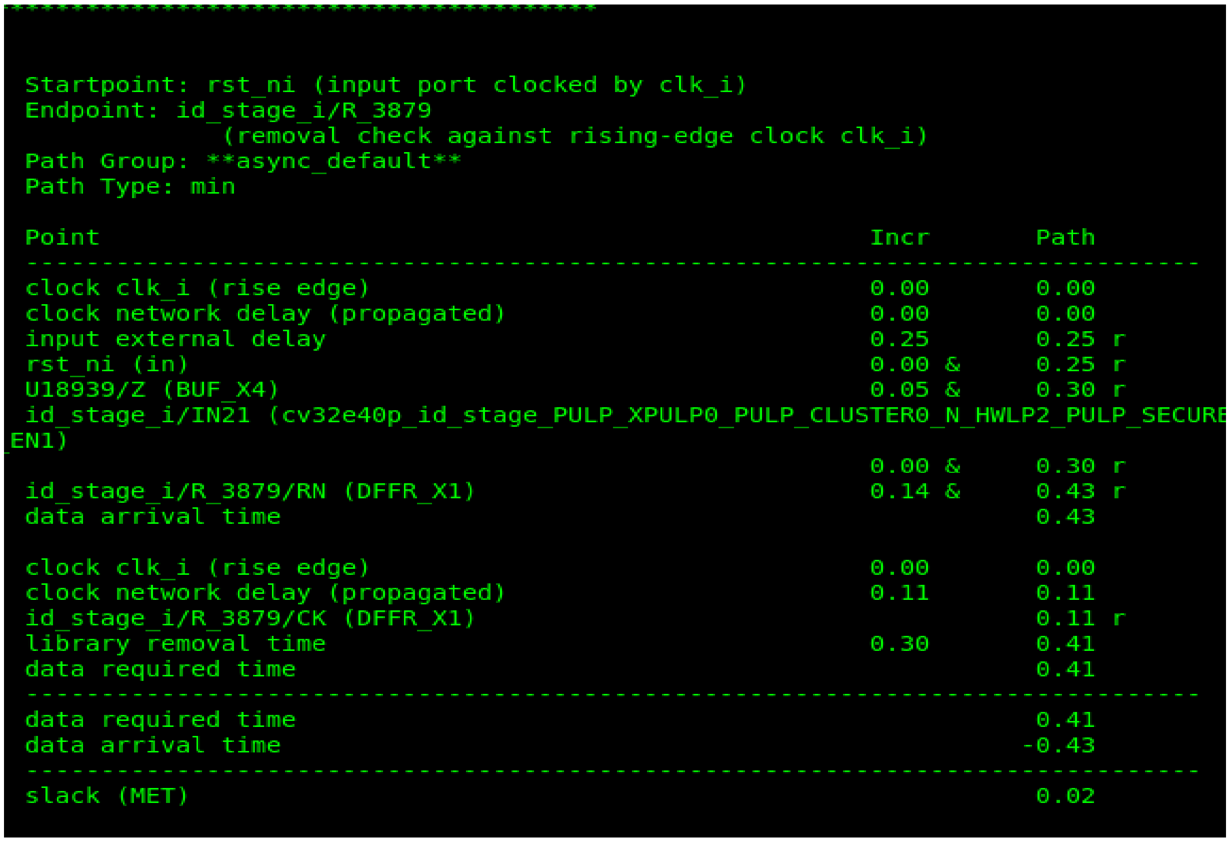
– Maximum delay timing results:



```
  Path Group: clk_i
  Path Type: max

  Point                                                      Incr       Path
  ------------------------------------------------------------------------------
  clock clk_i (rise edge)                                    0.00       0.00
  clock network delay (propagated)                           0.00       0.00
  input external delay                                       2.00       2.00 r
  instr_rdata_i[17] (in)                                     0.00 &     2.00 r
  U16958/ZN (INV_X1)                                         0.01 &     2.01 f
  U16955/ZN (NAND2_X1)                                       0.02 &     2.03 r
  U15933/ZN (NAND3_X1)                                       0.03 &     2.05 f
  U18388/ZN (NAND2_X1)                                       0.05 &     2.10 r
  U11772/ZN (INV_X4)                                         0.02 &     2.12 f
  U11191/ZN (AOI21_X1)                                       0.10 &     2.22 r
  U13026/ZN (NOR2_X1)                                        0.03 &     2.24 f
  U13025/ZN (NAND2_X1)                                       0.03 &     2.28 r
  U16959/ZN (NOR2_X1)                                        0.03 &     2.31 f
  U11261/ZN (NOR2_X1)                                        0.06 &     2.36 r
  U11175/ZN (NAND3_X1)                                       0.04 &     2.40 f
  U11259/ZN (INV_X1)                                         0.02 &     2.43 r
  U16585/ZN (NAND3_X1)                                       0.03 &     2.45 f
  U12913/ZN (NAND2_X1)                                       0.02 &     2.47 r
  U11162/ZN (AOI21_X1)                                       0.02 &     2.49 f
  U15377/ZN (INV_X1)                                         0.01 &     2.50 r
  U12902/ZN (NAND4_X1)                                       0.06 &     2.56 f
  id_stage_i/instr_rdata_i_20_ (cv32e40p_id_stage_PULP_XPULP0_PULP_CLUSTER0_N_HWLP
DEBUG_TRIGGER_EN1)
                                                             0.00 &     2.56 f
  id_stage_i/U79/ZN (NOR2_X1)                                0.04 &     2.60 r
  id_stage_i/R_3290/D (DFFS_X1)                              0.00 &     2.60 r
  data arrival time                                                     2.60

  clock clk_i (rise edge)                                    2.50       2.50
  clock network delay (propagated)                           0.15       2.65
  id_stage_i/R_3290/CK (DFFS_X1)                                        2.65 r
  library setup time                                        -0.02       2.62
  data required time                                                    2.62
  ------------------------------------------------------------------------------
  data required time                                                    2.62
  data arrival time                                                    -2.60
  ------------------------------------------------------------------------------
  slack (MET)                                                           0.02
```

Figure 70: The setup slack after PrimeTime in Topographical Flow

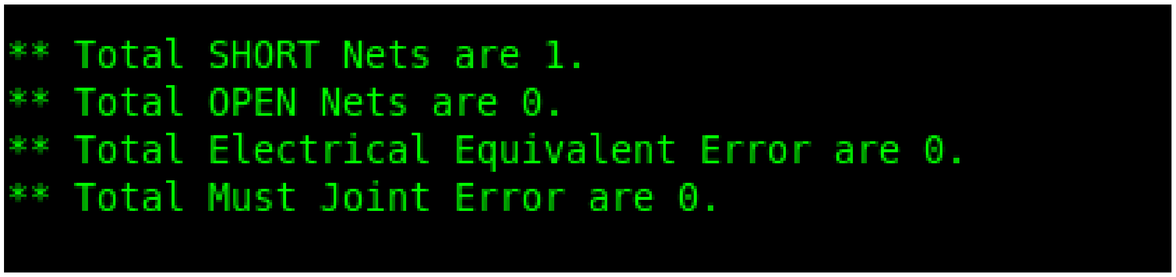
– Minimum delay timing results:



```
****************************************


 Startpoint: rst_ni (input port clocked by clk_i)
 Endpoint: id_stage_i/R_3879
              (removal check against rising-edge clock clk_i)
 Path Group: **async_default**
 Path Type: min

 Point                                                        Incr       Path
 -----------------------------------------------------------------------------
 clock clk_i (rise edge)                                      0.00       0.00
 clock network delay (propagated)                             0.00       0.00
 input external delay                                         0.25       0.25 r
 rst_ni (in)                                                  0.00 &     0.25 r
 U18939/Z (BUF_X4)                                            0.05 &     0.30 r
 id_stage_i/IN21 (cv32e40p_id_stage_PULP_XPULP0_PULP_CLUSTER0_N_HWLP2_PULP_SECURE
_EN1)
                                                              0.00 &     0.30 r
 id_stage_i/R_3879/RN (DFFR_X1)                               0.14 &     0.43 r
 data arrival time                                                       0.43

 clock clk_i (rise edge)                                      0.00       0.00
 clock network delay (propagated)                             0.11       0.11
 id_stage_i/R_3879/CK (DFFR_X1)                                          0.11 r
 library removal time                                         0.30       0.41
 data required time                                                      0.41
 -----------------------------------------------------------------------------
 data required time                                                      0.41
 data arrival time                                                      -0.43
 -----------------------------------------------------------------------------
 slack (MET)                                                             0.02
```

Figure 71: The Hold slack after PrimeTime in Topographical Flow

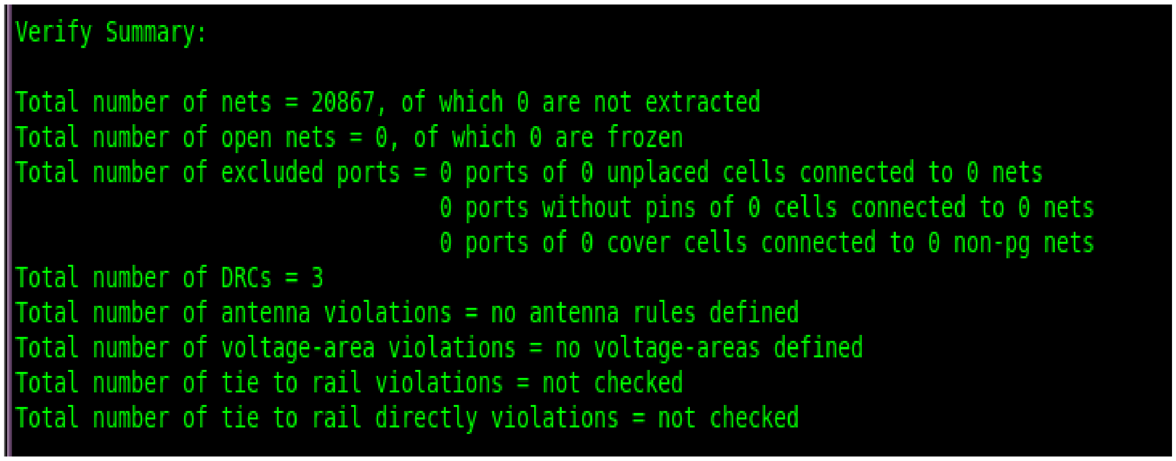
7.3.4. LVS results



```
** Total SHORT Nets are 1.
** Total OPEN Nets are 0.
** Total Electrical Equivalent Error are 0.
** Total Must Joint Error are 0.
```

Figure 72: The LVS results of the Topographical Flow

7.3.5. DRC results



```
Verify Summary:

Total number of nets = 20867, of which 0 are not extracted
Total number of open nets = 0, of which 0 are frozen
Total number of excluded ports = 0 ports of 0 unplaced cells connected to 0 nets
                                 0 ports without pins of 0 cells connected to 0 nets
                                 0 ports of 0 cover cells connected to 0 non-pg nets
Total number of DRCs = 3
Total number of antenna violations = no antenna rules defined
Total number of voltage-area violations = no voltage-areas defined
Total number of tie to rail violations = not checked
Total number of tie to rail directly violations = not checked
```

Figure 73: The DRC results of the Topographical Flow

Chapter 8

Conclusion

The goal of this thesis is to help ASIC designers in selecting the most appropriate flow that meets their requirements. A complete RTL-to-GDSII flow for the OpenPULP Core known as RI5CY was accomplished in this thesis, with three distinct synthesis techniques yielding three different flow outcomes. Following that, we'll look at the primary advantages and disadvantages of each flow in comparison to other flows. We will summarize our findings in both the post-synthesis and post-layout stages of the implementation flows in the sections below.

8.1. Post-Synthesis Results

In post-synthesis stage as shown in Table 37, we can notice that the Flat Flow has the biggest area. And the Hierarchical Flow has the least area as it works with the design as blocks. Also in power, the Topographical Flow and the Flat flow has nearly the same total power. According to the critical delay, the Topographical Flow has the least clock period comparing to the other flows. However, we will distinguish key benefits of each flow later on after actual layout is made for the core. Below charts that demonstrate the results between flows in post-synthesis stage.

Table 37: The Post-synthesis results

	Flat Flow	Topographical Flow (Second path)	Hierarchical Flow
Critical path delay (n sec)	2.6	2.5	2.7
Total cell area (μm^2)	32678.366549	34267.716551	33082.9525
Total Power (mW)	4.3065	4.5074	5.0174

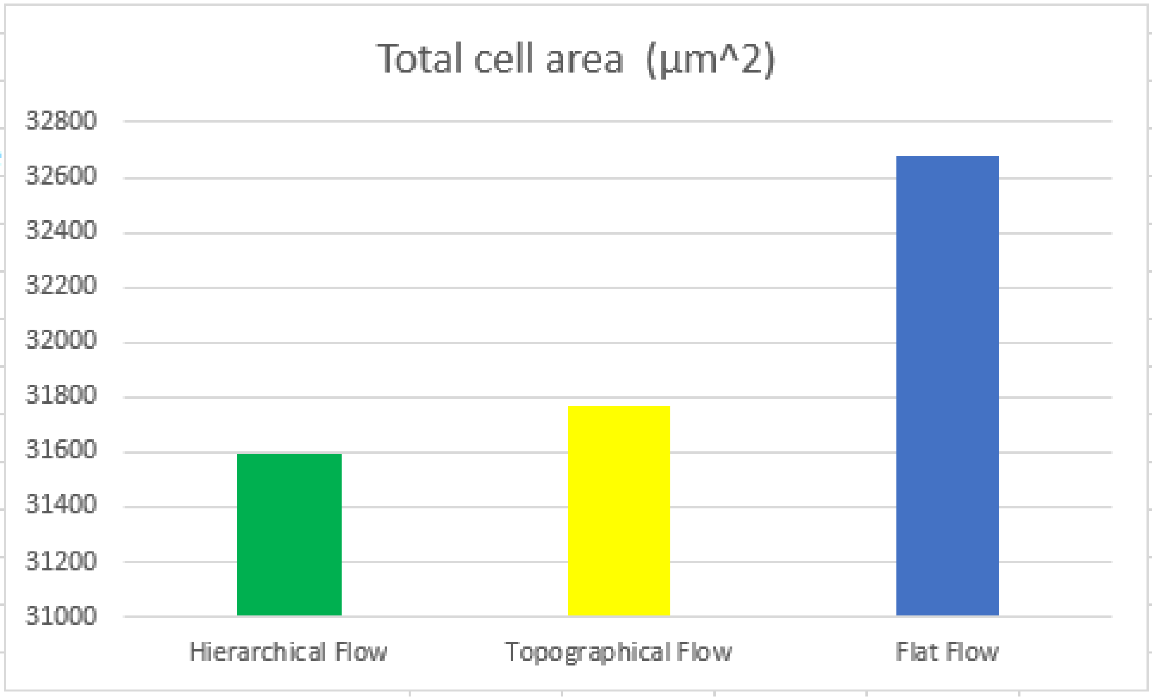


Figure 74: The total area comparison after post-synthesis

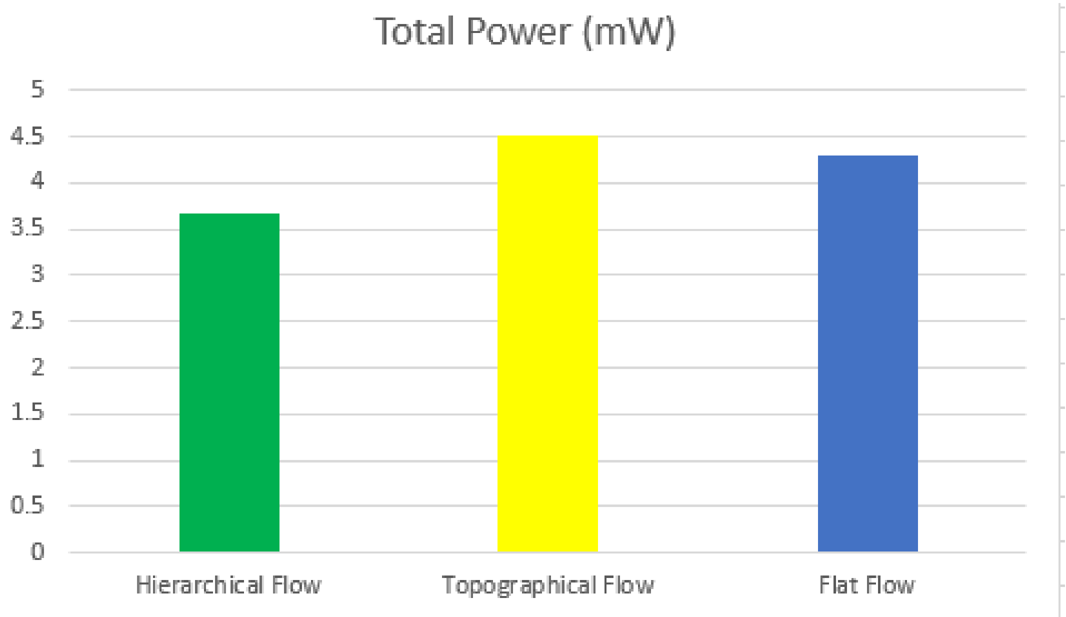


Figure 75: The total power comparison after post-synthesis

8.2. Post-layout Results

The post-layout results in addition to the total power calculated at each PDK corner of the verified corners, including the corner which is used for actual implementation of the core in all flows which is (Slow-Slow / 0.95 V / 40 °C). Below are charts that help visualize these results.

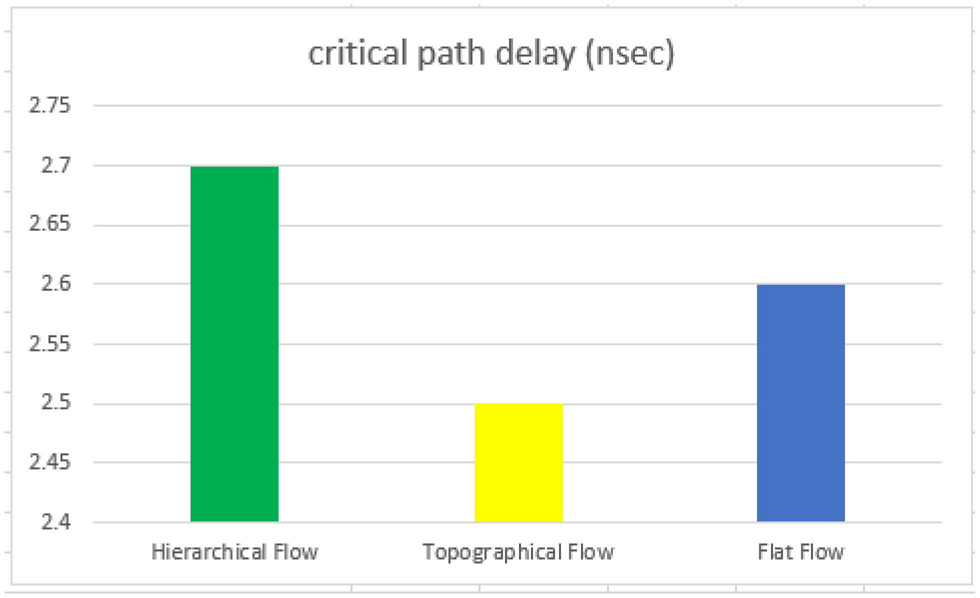


Figure 76: The critical path delay comparison

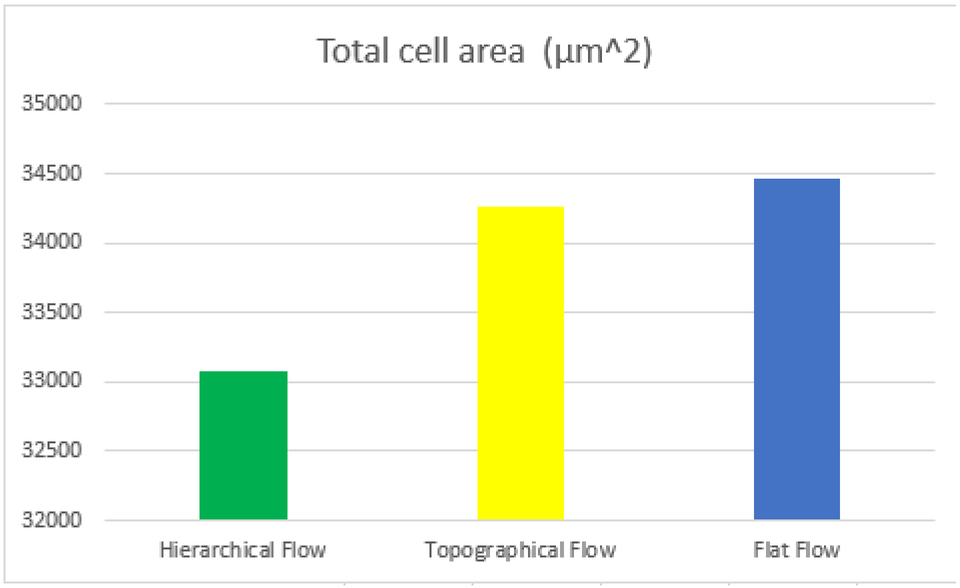


Figure 77: The total area comparison after post-Layout

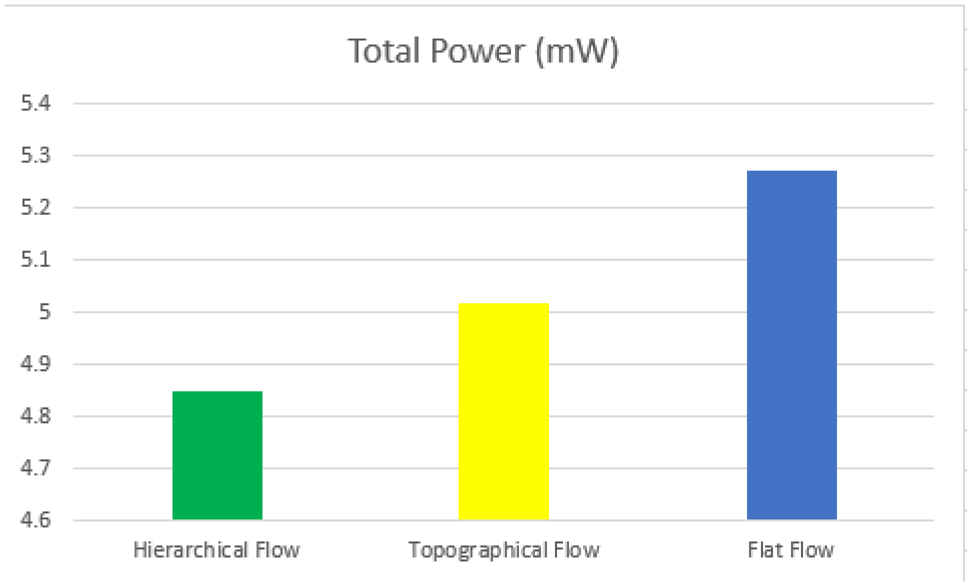


Figure 78: The total power comparison after post-Layout

Table 38: The key results

	Flat Flow	Topographical Flow (Second path)	Hierarchical Flow
Critical path delay (n sec)	2.6	2.5	2.7
Total cell area (μm^2)	34461.630550	31768.646537	31598.406520
Total Power (mW)	5.2742	5.0174	4.8474

Table 38 summarizes the key results that distinguish each flow. We can see that Hierarchical flow has the least total cell area and the least total power, but as our target is to work on maximum speed, So, the Topographical Flow is the best flow and it can be run in less clock period than the 2.5 nsec as the slack margin is 0.02 nsec which is a big margin. But our tool doesn't support the Topographical Flow as perfect as it has to be and all the flows are clean LVS with almost clean DRCs and that was our target.

Therefore, if an ASIC designer might want to achieve maximum speed, we strongly advice him/her with working with Topographical flow. If the goal is to minimize occupied area as much as possible, Hierarchical flow is the best choice in that case. And if the goal is to reduce the total power consumed by the design, then it is preferred to perform the implementation with Hierarchical flow to achieve highest cell area optimizations.

8.3. Future Work

We present some solutions in this area to improve design performance and speed across all implementation flows. We'll go over these concepts in greater depth further down.

8.3.1. Switch from RVT Cells to LVT Cells

LVT standard cells have high speed than RVT cells due to the less threshold voltage. This can fix the timing setup very quickly than using the RVT cells at high frequency.

References

[1] Synopsys. (2016). IC Compiler™ II Implementation User Guide : Version L-2016.03-SP4. Mountain View, CA: Synopsys.

[2] Synopsys. (2005). Design Compiler™ User Guide : Version L-2016.03-SP4. Mountain View, CA: Synopsys.

[3] Rakesh ChadhaJ, Bhasker. (2009) Static Timing Analysis for Nanometer Designs : 1st edition. Boston, MA: Springer.

[4] Bhatnagar, Himanshu. (2002). Static Timing Analysis for Nanometer Designs : 2nd edition. Boston, MA: Springer.

[5] OpenHW Group, link: https://github.com/openhwgroup/cv32e40

[6] OpenHW Group, link: https://pulp-platform.org/implementation.html

[7] Semi engineering, link:

https://semiengineering.com/knowledge_centers/low-power/low-power-design/power-consumption/

[8] Sign-off semi-conductors, link:

https://www.signoffsemi.com/sign-off-checks/

Appendix A

RTL-to-GDSII Flow

A.1 GDSII format

In the design of integrated circuits, the most popular format for interchange is the Calma GDS II stream format (GDS II is a trademark of Calma Company, a wholly owned subsidiary of General Electric Company, U.S.A.). For many years, this format was the only one of its kind and many other vendors accepted it in their systems. Although Calma has updated the format as their CAD systems have developed, they have maintained backward compatibility so that no GDS II files become obsolete. This is important because GDS II is a binary format that makes assumptions about integer and floating-point representations.

A GDS II circuit description is a collection of cells that may contain geometry or other cell references. These cells, called structures in GDS II parlance, have alphanumeric names up to 32 characters long. A library of these structures is contained in a file that consists of a library header, a sequence of structures, and a library tail. Each structure in the sequence consists of a structure header, a sequence of elements, and a structure tail. There are seven kinds of elements: boundary defines a filled polygon, path defines a wire, structure reference invokes a subcell, array reference invokes an array of subcells, text is for documentation, node defines an electrical path, and box places rectangular geometry.

For a larger IC circuit, we cannot use schematic drawing to represent and store its layout for obvious reasons.

- A language and database format were developed for this problem. This so called GDSII serves as a stream format database file format and has been used as the de facto industry standard for data exchange of integrated circuit or IC layout artwork.
- It is a binary file format representing planar geometric shapes, text labels, and other information about the IC layout in a hierarchical form.
- The data can be used to reconstruct all or part of the artwork to be used in sharing layouts, transferring artwork between different EDA tools, or creating photo masks for fabrication.

A.2 Constraint file

In this section, we describe the constraint file we use in our implementation. The constraint file is provided by open pulp and it is as follows:

- Specifying clocks and defining default clock definitions using command create_clock. The create_clock command specifies the characteristics of a clock, including the clock name, source, period, and waveform. *-name* switch specifies the name of the clock being created. *-period* switch specifies the period of the clock waveform in library time units. The defined clock period is 5ns, but we changed it according to the speed of each flow. For the hierarchal flow, clock period is 2.7ns. clock period is 2.6 for the flat flow, and 2.5 for the topographical flow.

- Defining I/O constraints. Defining the input and output delays is very important to make sure that there will be no problems when the block we worked on integrate with other blocks. They are used to model the environment of that module. set_input_delay command sets input delay on pins or input ports relative to a clock signal. It has two main arguments; first is the "delay_value". It specifies the path delay. The delay_value must be in units consistent with the technology library used during optimization. The delay_value represents the amount of time the signal is available after a clock edge. This represents a combinational path delay from the clock pin of a register. The second argument is clock_name. It specifies the clock to which the specified delay is related. If -clock is not specified, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with the period determined by considering the sequential cells in the transitive fanout of each port.

The percentage of input and output delays from the clock period differs for different input and output ports. These percentages are defined in our constraint file as follows:

- Input delay for interrupts is 0.5 * clock period.
- Output delay for interrupt related signals is 0.25*clock period.
- Input delay for early signals is 0.1 * clock period.
- OBI input delays are 0.8 * clock period.
- OBI output delays are 0.6 * clock period
- Input delays for non-RISC-V Bus Interface ports are 0.1*clock period.
- output delays for non-RISC-V Bus Interface ports are 0.6*clock period.
- core_sleep_o output delay is 0.25 * clock period.

A.3 clock gating

Clock gating is a very common technique to save power by stopping the clock to a module when the module is not operating. Clock gating cells are required in CV32E40P, especially in sleep unit and latch-based register file. These cells are specific to the selected target technology and thus not provided as part of the RTL design. Therefore, the clock gate RTL code that provided by open pulp is a simulation only version of the clock gating cell. This file contains a module called cv32e40p_clock_gate that has the following ports:

- clk_i: Clock Input
- en_i: Clock Enable Input
- scan_cg_en_i: Scan Clock Gate Enable Input (activates the clock even though en_i is not set)
- clk_o: Gated Clock Output

We manually instantiated clock gating cells from our standard cell library (Nangate Open Cell Library) that is wrapped in a module called cluster clock gating and has the following ports:

- clk_i: Clock Input
- en i: Clock Enable Input
- test en i: Test Enable Input (activates the clock even though en i is not set)
- clk_o: Gated Clock Output

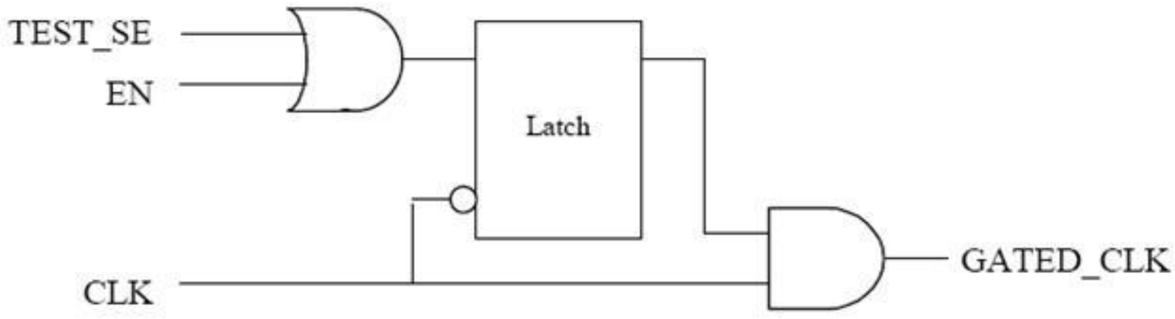


Figure 79: The clock gate

A.4 IR Drop Analysis:

The voltage drops in metal wires that make up power grids before it reaches the vdd pins of the cells is known as IR Drop. When there are cells with high current requirements or high switching regions, the IR drops. The IR drop creates a voltage loss, which delays the cells, triggering setup and hold violations. Once the chip is produced, hold violations cannot be corrected.

A device's power consumption is divided into two categories: dynamic, also known as switching power, and static, also known as leakage power. Leakage power has become the primary power consumer in geometries lower than 90nm, whilst switching is the larger contribution in larger geometries. Both forms of power can be reduced using power reduction measures.

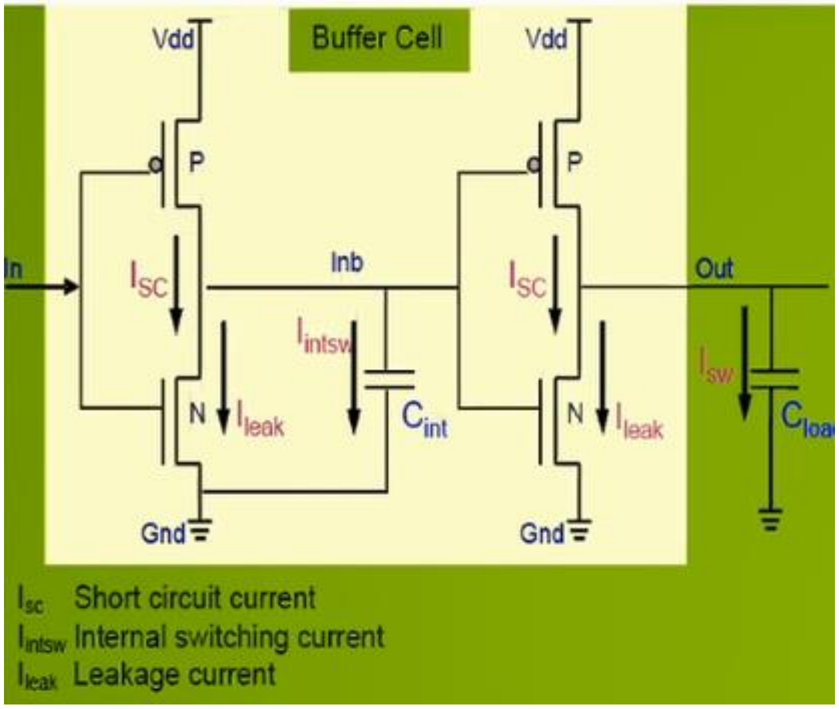


Figure 80: The IR analysis

A.4.1. Power Dissipation in CMOS

Total power is a function of switching activity, capacitance, voltage, and the transistor structure itself.

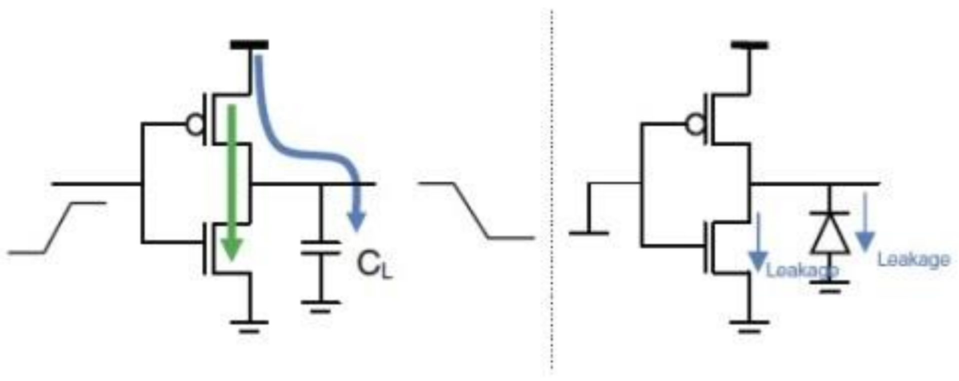


Figure 81: The power dissipation in CMOS

Total power is the sum of the dynamic and leakage power

$$Total\ Power = P_{switching} + P_{short\ circuit} + P_{leakage}$$

Equation 6:The total power

There are two types of IR drop analysis namely:

A.4.1.1. Static IR drop analysis:

Leakage power is a function of the supply voltage V_{dd}, the switching threshold voltage V_{th}, and the transistor size.

$$P_{Leakage} = f\ (V_{dd}, V_{th}, W/L)$$

Equation 7: The leakage power

Where V_{dd} = the supply voltage, V_{th} = the threshold voltage, W = the transistor width and L = the transistor length.

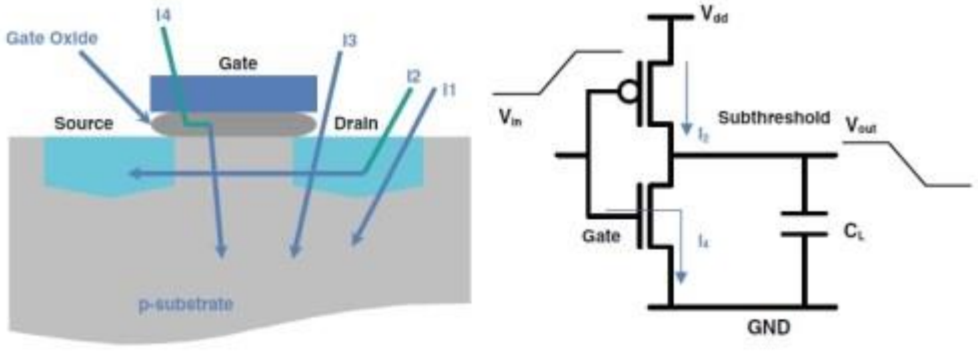


Figure 82: The leakage power

Of the following leakage components, sub-threshold leakage is dominant.

- I1: Diode reverse bias current
- I2: Sub-threshold current
- I3: Gate-induced drain leakage
- I4: Gate oxide leakage

While dynamic power is dissipated only when switching, leakage power due to leakage current is continuous.

So, static power analysis:

- Calculates the average voltage drop of entire design assuming current drawn across is constant.
- As average current is calculated this analysis depends on time period. This analysis is good for signoff checks in older technology.

$$P_{static} = V_{dd} \times I_{leakage}$$

Equation 8: The static power

A.4.1.2. Dynamic power:

Dynamic IR drop analysis:

- Depends on switching activity of the logic.
- Is vector dependent.
- Less dependent on clock period as depends on instantaneous current.
- Analysis of peak current demand and highly localized cells.

Dynamic power is the sum of two factors: switching power plus short-circuit power.

Switching power is dissipated when charging or discharging internal and net capacitances. Short-circuit power is the power dissipated by an instantaneous short-circuit connection between the supply voltage and the ground at the time the gate switches state.

$$P_{switching} = a.f.C_{eff}.{V_{dd}}^2$$

Equation 9: The switching power

Where a = switching activity, f = switching frequency, C_{eff} = the effective capacitance and V_{dd}= the supply voltage.

$$P_{short\,circuit} = I_{sc}.V_{dd}.f$$

Equation 10: The short-circuit power

Where I_{sc} = the short-circuit current during switching, V_{dd}= the supply voltage and f = switching frequency.

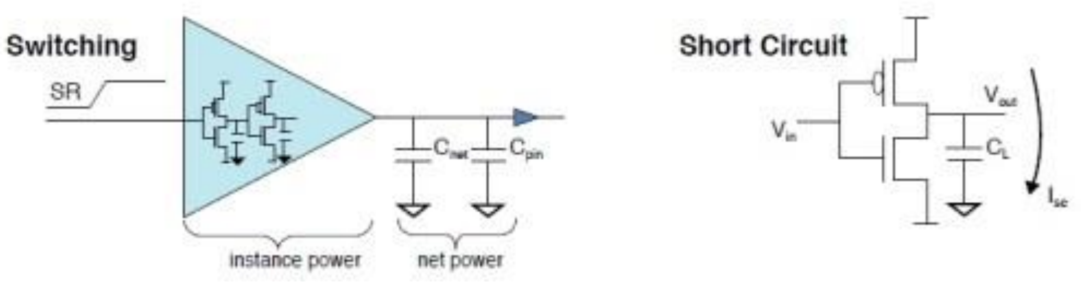


Figure 83: The dynamic power

Dynamic power can be lowered by reducing switching activity and clock frequency, which affects performance; and also, by reducing capacitance and supply voltage. Dynamic power can also be reduced by cell selection-faster slew cells consume less dynamic power.

A.4.2. Methods to reduce IR drop

Robust power mesh– Initial power grid is made based on static IR analysis due to late availability of switching activity. If there is IR drop due to some of the clustered cells then adding a strip will make the power mesh more robust.

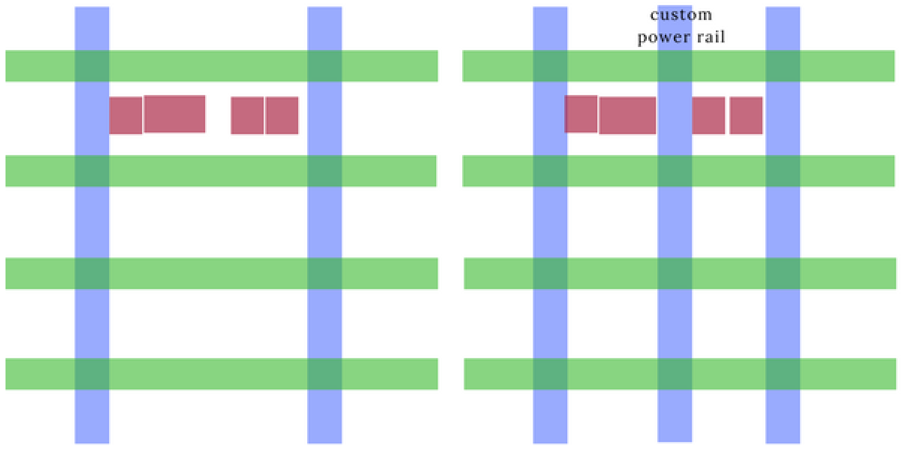
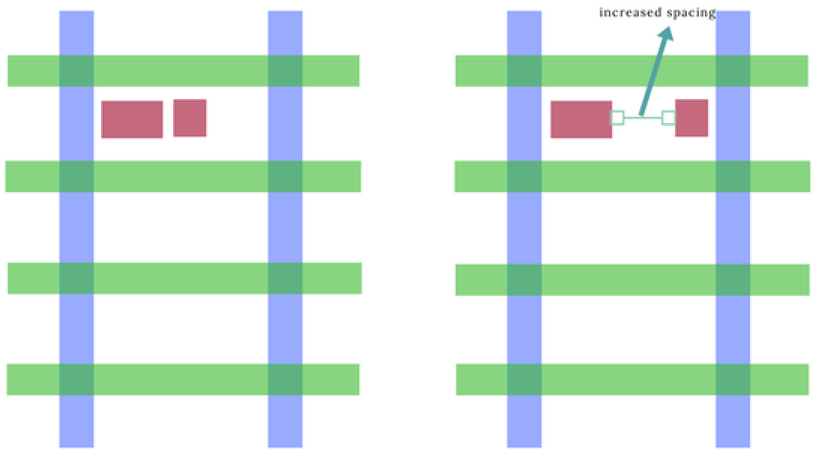


Figure 84: Custom power rail added to make it robust

- De-cap– These are decoupling capacitors which are spread across the high switching region to maintain the voltage.
- Spacing– If clock cells are clustered and causing IR drop, then by spacing them apart near to different power rails will reduce the IR drop. While shifting the cell to next power rail, it should be made sure that the power rail is not driving many cells, because adding another cell may give IR drop.



- Reducing load– Cells driving more load will be drawing more current. Hence reducing load will reduce IR drop.
- Downsizing– Cells of smaller size will draw less current. But the transition of cells should not become worse.
- The number of power switches can be increased to reduce IR drop
- It should be made sure that all the power pins of macros are properly connected to the power rails.

Note:

- For accurate dynamic analysis VCD files (switching activity file) with SDF (standard delay format) is better.
- Glitches produced from combinational circuit may act as instantaneous switch. Reducing them will decrease the pessimism of dynamic IR drop analysis.

- IR drop analysis is done in RC worst corner (corner having more resistance of rails) and FF process, high voltage and high temp corner (PVT corner) because current is drawn more in this corner.

A.5 Design Rule Check (DRC)

The Design Rule Check (DRC) is a check for certain layout rules, to ensure design will be manufactured reliably. It is a part of the PDK that determines whether the layout of a chip satisfies a series of recommended parameters called design rules. Design rules are set of parameters provided by semiconductor manufacturers to the designers, in order to verify the correctness of a mask set. It varies based on semiconductor manufacturing process. These rule set describes certain restrictions in geometry and connectivity to ensure that the design has sufficient margin to take care of any variability in manufacturing process.

Design rule checks are nothing but physical checks of metal width, pitch and spacing requirement for the different layers with respect to different manufacturing process. If we give physical connection to the components without considering the DRC rules, then it will lead to failure of functionality of chip, so all DRC violations has to be cleaned up.

Here are some basic and common types of DRC rules

- Minimum width
- Minimum spacing
- Minimum area
- Wide metal jog
- Misaligned via wire
- Special notch spacing
- End of line spacing

After the completion of physical connection, we check each and every polygon in the design, based on the design rules and reports all the violations. This whole process is called Design Rule Check.

A.6 Layout vs. Schematic (LVS)

DRC only checks if the given layout complies with the fabrication unit's design requirements. It does not guarantee layout functionality. As a result, the concept of LVS was born. This appendix focuses on how LVS works and the most common challenges that LVS users confront.

A.6.1. How LVS works

Inputs needed to perform LVS are:

- (.v) netlist of the design
- GDS layout database of the design
- LVS rule deck

Note: .v and GDS should be of same stage

The LVS rule deck is a collection of code written in SVRF (Standard Verification Rule Format) or TCL Verification Format (TVF). It instructs the tool on how to extract devices and IC connectivity. It contains the layer definition, which is used to identify the layers in the layout file and match them to their GDS locations. It also has definitions for device structures.

A.6.2. Steps of LVS check

1. **Extraction**: The tool accepts a GDSII file with all of the layers and uses a polygon-based technique to identify components such as transistors, diodes, capacitors, and resistors, as well as connectivity information between devices in the layout. All of the device layers, device terminals, device sizes, nets, vias, and pin positions are described and given a unique identifier.
2. **Reduction:** All the defined information is extracted in the form of netlist.
3. **Comparison:** Using the LVS rule deck, the extracted layout netlist is compared to the netlist of the same stage. The number of instances, nets, and ports are compared at this point. All mismatches are reported, including shorts and openings, pin mismatches, and so on. Topology and size mismatch are also checked by the tools.

A.6.3. LVS flow

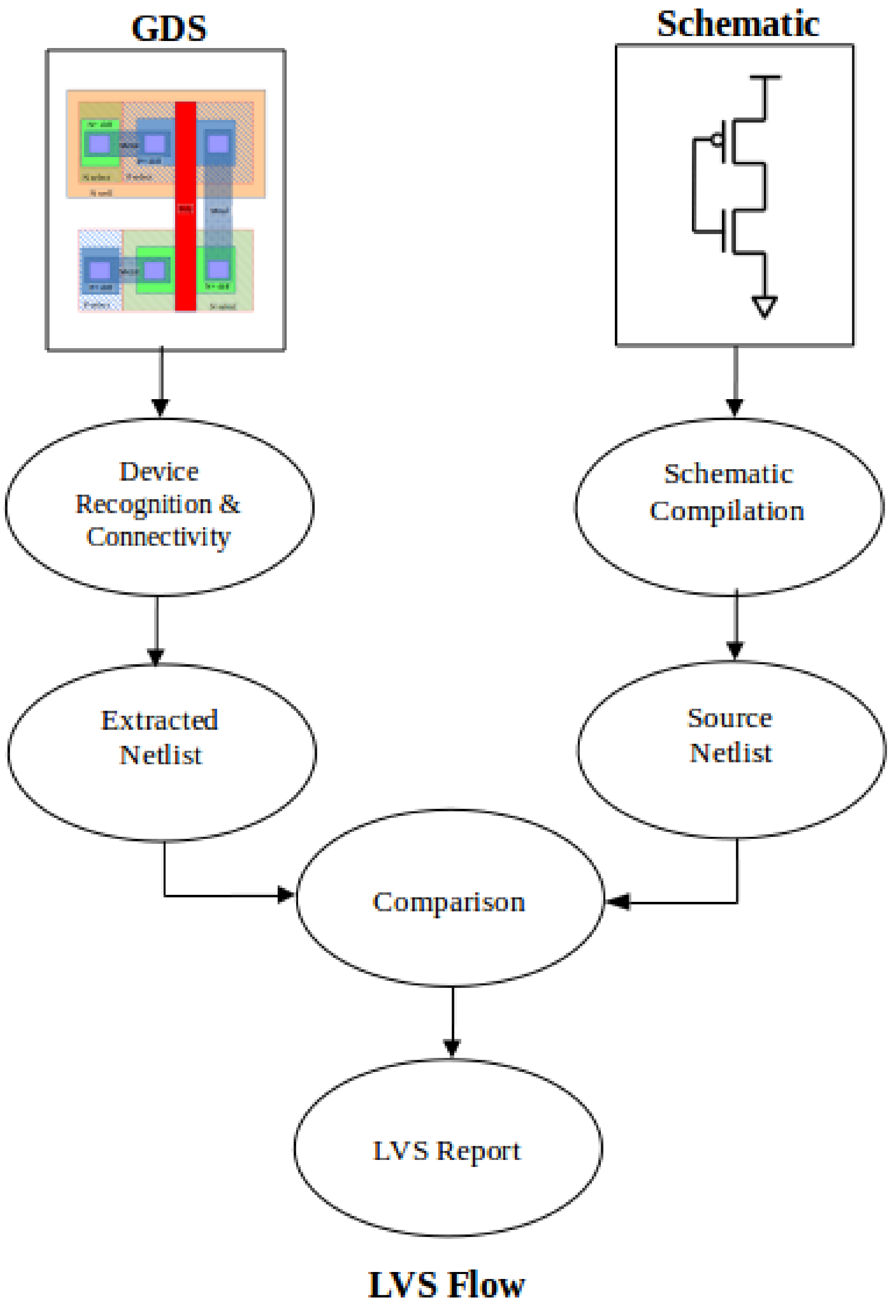


Figure 85: The LVS flow

A.6.4. Commonly faced LVS issues

LVS check includes following comparisons:

- Number of devices in schematic and its layout
- Type of devices in schematic and its layout
- Number of nets in schematic and its layout

A.6.5. Typical errors which can occur during LVS checks

1. Shorts: Shorts are formed, if two or more wires which should not be connected together are connected.
2. Opens: Opens are formed, if the wires or components which should be connected together are left floating or partially connected.
3. Component mismatch: Component mismatch can happen, if components of different types are used (e.g, LVT cells instead of HVT cells).

Appendix B

Scripts of Flat Flow

B.1 Synthesis script

```
set design cv32e40p_core

set_app_var   search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front
_End/Liberty/NLDM"

set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"

set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"

sh rm -rf work

sh mkdir -p work

define_design_lib work -path ./work

set hdlin_sverilog_std              2009

analyze           -library           work           -format           verilog
/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Back_
End/virtuoso/NangateOpenCellLibrary/CLKGATETST_X4/functional/verilog.v

  analyze -library work -format sverilog ../rtl/cv32e40p_register_file_ff.sv

  analyze -library work -format sverilog ../rtl/include/cv32e40p_apu_core_pkg.sv

  analyze -library work -format sverilog ../rtl/include/cv32e40p_pkg.sv

  analyze -library work -format sverilog ../rtl/include/cv32e40p_fpu_pkg.sv

  analyze -library work -format sverilog ../rtl/cv32e40p_alu.sv
analyze -library work -format sverilog ../rtl/cv32e40p_alu_div.sv

analyze -library work -format sverilog ../rtl/cv32e40p_ff_one.sv

analyze -library work -format sverilog ../rtl/cv32e40p_popcnt.sv

analyze -library work -format sverilog ../rtl/cv32e40p_compressed_decoder.sv

analyze -library work -format sverilog ../rtl/cv32e40p_controller.sv

analyze -library work -format sverilog ../rtl/cv32e40p_cs_registers.sv

analyze -library work -format sverilog ../rtl/cv32e40p_decoder.sv

analyze -library work -format sverilog ../rtl/cv32e40p_int_controller.sv

analyze -library work -format sverilog ../rtl/cv32e40p_ex_stage.sv

analyze -library work -format sverilog ../rtl/cv32e40p_hwloop_regs.sv

analyze -library work -format sverilog ../rtl/cv32e40p_id_stage.sv

analyze -library work -format sverilog ../rtl/cv32e40p_if_stage.sv

analyze -library work -format sverilog ../rtl/cv32e40p_load_store_unit.sv

analyze -library work -format sverilog ../rtl/cv32e40p_mult.sv

analyze -library work -format sverilog ../rtl/cv32e40p_prefetch_buffer.sv

analyze -library work -format sverilog ../rtl/cv32e40p_prefetch_controller.sv

analyze -library work -format sverilog ../rtl/cv32e40p_obi_interface.sv

analyze -library work -format sverilog ../rtl/cv32e40p_aligner.sv

analyze -library work -format sverilog ../rtl/cv32e40p_sleep_unit.sv

analyze -library work -format sverilog ../rtl/cv32e40p_core.sv

analyze -library work -format sverilog ../rtl/cv32e40p_apu_disp.sv

analyze -library work -format sverilog ../rtl/cv32e40p_fifo.sv
```

```
    analyze -library work -format sverilog ../rtl/cv32e40p_clock_gate.sv

analyze -library work -format sverilog  ../rtl/${design}.sv

elaborate $design -lib work

current_design

report_hierarchy > ./report/synth_hier.rpt

check_design

source ../constraints/cv32e40p_core.sdc

link

compile_ultra -timing_high_effort_script -retime

compile_ultra -timing_high_effort_script -retime -incremental

#report_auto_ungroup

report_area > ./report/synth_area.rpt

report_power > ./report/synth_power.rpt

report_cell > ./report/synth_cells.rpt

report_qor  > ./report/synth_qor.rpt

report_resources > ./report/synth_resources.rpt

report_timing -max_paths 10 > ./report/synth_timing.rpt

write_sdc  output/${design}.sdc

define_name_rules  no_case -case_insensitive

change_names -rule no_case -hierarchy

change_names -rule verilog -hierarchy

set verilogout_no_tri   true
set verilogout_equation  false

write -hierarchy -format verilog -output output/${design}.sv

write -f ddc -hierarchy -output output/${design}.ddc

dc_shell-t  | tee ./log/syn.log

exit
```

B.2 PnR scripts

```
#############################################

########### 1. DESIGN SETUP #################

#############################################

set design cv32e40p_core

sh rm -rf $design

set sc_dir "/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12"

set_app_var search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front
_End/Liberty/NLDM \

                                /home/mohamed/Desktop/johnson/rtl"

set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"

set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"

create_mw_lib   ./${design} \

             -technology $sc_dir/tech/techfile/milkyway/FreePDK45_10m.tf \
                    -mw_reference_library $sc_dir/lib/Back_End/mdb \
                    -open
set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"
set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"
set tech2itf "$sc_dir/tech/rcxt/FreePDK45_10m.map"
set_tlu_plus_files -max_tluplus $tlupmax \
               -min_tluplus $tlupmin \
                      -tech2itf_map $tech2itf
import_designs  ../syn/output/${design}.v \
             -format verilog \
                    -top ${design} \
                    -cel ${design}
source  ../syn/output/${design}.sdc
save_mw_cel -as ${design}_1_imported
##############################################
########### 2. Floorplan ####################
##############################################
## Create Starting Floorplan
############################
create_floorplan -core_utilization 0.25 \
      -core_aspect_ratio 2 \
          -start_first_row -flip_first_row \
        -left_io2core 12.4 -bottom_io2core 12.4 -right_io2core 12.4 -top_io2core 12.4

## CONSTRAINTS

##############

## Here, We define more constraints on your design that are related to floorplan stage.

report_ignored_layers

remove_ignored_layers -all

set_ignored_layers -max_routing_layer metal8

## Initial Virtual Flat Placement

#################################

## Use the following command with any of its options to meet a specific target

set_app_var  placer_max_cell_density_threshold 0.3

create_fp_placement  -timing_driven  -no_hierarchy_gravity  -effort  high  -congestion  -
incremental all

save_mw_cel -as ${design}_2_fp

##################################################

########### 3. POWER NETWORK ####################

##################################################

## Defining Logical POWER/GROUND Connections

############################################

derive_pg_connection          -power_net VDD                \

                              -ground_net VSS       \

                              -power_pin VDD                \
                              -ground_pin VSS

## Define Power Ring

####################

set_fp_rail_constraints  -set_ring -nets {VDD VSS}  \

                  -horizontal_ring_layer { metal9 } \

                  -vertical_ring_layer { metal10 } \

                              -ring_spacing 0.8 \

                              -ring_width 5 \

                              -ring_offset 0.8 \

                              -extend_strap core_ring

## Define Power Mesh

####################

set_fp_rail_constraints -add_layer  -layer metal10 -direction vertical   -max_strap 128 -
min_strap 20 -min_width 2.5 -spacing minimum

set_fp_rail_constraints -add_layer  -layer metal9  -direction horizontal -max_strap 128 -
min_strap 20 -min_width 2.5 -spacing minimum

set_fp_rail_constraints -add_layer  -layer metal8  -direction vertical   -max_strap 128 -
min_strap 20 -min_width 2.5 -spacing minimum

set_fp_rail_constraints -set_global

## Creating virtual PG pads

###########################

# you can create them with gui. Preroute > Create Virtual Power Pad

set die_llx [lindex [lindex [ get_attribute [get_die_area] bbox] 0] 0]
set die_lly [lindex [lindex [ get_attribute [get_die_area] bbox] 0] 1]

set die_urx [lindex [lindex [ get_attribute [get_die_area] bbox] 1] 0]

set die_ury [lindex [lindex [ get_attribute [get_die_area] bbox] 1] 1]

for {set i "[expr $die_llx + 20]"} {$i < "[expr $die_urx - 40]"} {set i [expr $i + 80]} {

        create_fp_virtual_pad -net VSS -point "{$i $die_lly}"

        create_fp_virtual_pad -net VDD -point "{[expr $i + 40] $die_lly}"

        create_fp_virtual_pad -net VSS -point "{$i $die_ury}"

        create_fp_virtual_pad -net VDD -point "{[expr $i + 40] $die_ury}"

}

for {set i "[expr $die_lly + 20]"} {$i < "[expr $die_ury - 40]"} {set i [expr $i + 80]} {

        create_fp_virtual_pad -net VSS -point "{$die_llx $i}"

        create_fp_virtual_pad -net VDD -point "{$die_llx [expr $i + 40]}"



        create_fp_virtual_pad -net VSS -point "{$die_urx $i}"

        create_fp_virtual_pad -net VDD -point "{$die_urx [expr $i + 40] }"

}

synthesize_fp_rail  -nets {VDD VSS} -synthesize_power_plan -target_voltage_drop 22 -
voltage_supply 1.1 -power_budget 500

## Analyze IR-drop; Modify power network constraints and re-synthesize, as needed.

## Max IR is 2% of Nominal Supply. In our case, 0.02 x 1.1v= 22mv

commit_fp_rail

set_preroute_drc_strategy -max_layer metal8
```

```
preroute_standard_cells -fill_empty_rows -remove_floating_pieces

## If you want to remove power and recreate it

#remove_net_shape  [get_net_shapes -of_objects [get_nets -all "VSS VDD"]]

#remove_via  [get_vias -of_objects [get_nets -all "VSS VDD"]]

## MAy need => remove_fp_virtual_pad -all

## Analyze IR-drop; Modify power network constraints and re-synthesize, as needed.

analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1

## Final Floorplan Assessment

#create_fp_placement -incremental all; # Updates fp placement after PG mesh creation.

#### Analyze Congestion

#### Analyze Timing

## Add Well Tie Cells

####################

add_tap_cell_array -master   TAP \

                        -distance 30 \

                        -pattern  stagger_every_other_row

save_mw_cel -as ${design}_3_power

report_timing -max_paths 10 > ./reports/timing_CU_power.rpt

############################################

########### 4. Placement ####################

############################################

puts "start_place"
```

```
## CHECKS

#########

set_ignored_layers -max_routing_layer metal8

report_ignored_layers ; # To Make sure they are as wanted.

check_physical_design -stage pre_place_opt

check_physical_constraints

## INITIAL PLACEMENT

####################

## Initial Placement can be done using the following command using any of its target
options

place_opt

## OPTIMIZATION

###############

# psynopt -area_recovery |-power| |-congestion|

psynopt -congestion


## FINAL ASSESSMENT

###################

check_legality

## If no legalized cells => legalize_placement -effort high -incremental

# Check Congestion
# Check Timing

 report_design_physical -utilization > ./reports/utilization_placement.rpt

# DEFINING POWER/GROUND NETS AND PINS

derive_pg_connection     -power_net VDD              \

                         -ground_net VSS      \

                         -power_pin VDD              \

                         -ground_pin VSS

## Tie fixed values

set tie_pins [get_pins -all -filter "constant_value == 0 | | constant_value == 1 && name !~
V* && is_hierarchical == false "]

derive_pg_connection          -power_net VDD              \

                         -ground_net VSS      \

                         -tie

connect_tie_cells -objects $tie_pins \

            -obj_type port_inst \

                  -tie_low_lib_cell  LOGIC0_X1 \

                  -tie_high_lib_cell LOGIC1_X1

puts "finish_place"

save_mw_cel -as ${design}_4_placed

report_timing -max_paths 10 > ./reports/timing_CU_placement.rpt

report_area > ./reports/area_placement.rpt
```

```
############################################

########### 5. CTS      ####################

############################################

puts "start_cts"

## CHECKS

#########

check_physical_design -stage pre_clock_opt

check_clock_tree

report_clock_tree

## CONSTRAINTS

##############

## Here, We define more constraints on your design that are related to CTS stage.

set_driving_cell -lib_cell BUF_X16 -pin Z [get_ports clk_i]

#### Set Clock Exceptions

### Set Clock Control/Targets

set_clock_tree_options \

        -clock_trees clk_i \

            -target_early_delay 0.1 \

            -target_skew 0.5 \

            -max_capacitance 300 \

            -max_fanout 10 \

            -max_transition 0.150
```

```
set_clock_tree_options -clock_trees clk_i \
                    -buffer_relocation true \
                    -buffer_sizing true \
                    -gate_relocation true \
                    -gate_sizing true
## Selection of CTS cells
set_clock_tree_references -references [get_lib_cells */CLKBUF*]
### Set Clock Physical Constraints
## Clock Non-Default Ruls (NDR) - Set it to be double width and double spacing
define_routing_rule my_route_rule  \
  -widths   {metal3 0.14 metal4 0.28 metal5 0.28 metal6 0.28 metal7 0.8} \
  -spacings {metal3 0.14 metal4 0.28 metal5 0.28 metal6 0.28 metal7 0.8}
set_clock_tree_options -clock_trees clk_i \
                  -routing_rule my_route_rule  \
                         -layer_list "metal3 metal4 metal5 metal6 metal7"
## To avoid NDR at clock sinks
set_clock_tree_options -use_default_routing_for_sinks 1
report_clock_tree -settings
## Clock Tree : Synhtesis, Optimization, and Routing
##################################################
## The 3 steps can be done with the combo command clock_opt. But below, we do them
```

```
individually.

compile_clock_tree  -clock_trees clk_i

## 1- CTS

clock_opt -only_cts -no_clock_route

## analyze

   report_design_physical -utilization

   report_clock_tree -summary ; # reports for the clock tree, regardless of relation between
FFs

   report_clock_tree

   report_clock_timing -type summary ; # reports for the clock tree, considering relation
between FFs

   report_timing        ; #slack (VIOLATED)   -0.02

   report_timing -delay_type min     ;      #report_worest_hold_violation   la2eto 0.01 met

   report_constraints -all_violators -max_delay -min_delay

   # Check Congestion

   # Check Timing

## 2- CTO

## To Consider Hold Fix -- Design Dependent

clock_opt -only_psyn -no_clock_route

#analyze

## 3- Clock Tree Routing

route_group -all_clock_nets

#analyze
```

```
clock_opt -only_psyn -congestion

clock_opt -only_psyn

route_group -all_clock_nets

set_propagated_clock [get_clocks clk_i]

## If any issue at analysis, update CT constraints

###############################################

# DEFINING POWER/GROUND NETS AND PINS

derive_pg_connection    -power_net VDD          \

                        -ground_net VSS     \

                        -power_pin VDD          \

                        -ground_pin VSS

save_mw_cel -as ${design}_5_cts

puts "finish_cts"

report_area > ./reports/area_CU_CTS.rpt

report_cell > ./reports/cells_CU_CTS.rpt

report_qor  > ./reports/qor_CU_CTS.rpt

report_resources > ./reports/resources_CU_CTS.rpt

report_timing -max_paths 10 > ./reports/timing_CU_CTS.rpt

report_power > ./reports/ power_CU_CTS.rpt

########################################

########### 6. Routing   ####################

########################################
```

```
## Before starting to route, you should add spare cells

insert_spare_cells -lib_cell {NOR2_X4 NAND2_X4} \

                    -num_instances 20 \

                    -cell_name SPARE_PREFIX_NAME \

                    -tie

set_dont_touch  [all_spare_cells] true

set_attribute [all_spare_cells]  is_soft_fixed true

##############################################

puts "start_route"

get_utilization -flat

#set_dont_touch [get_nets -all VDD]

#set_dont_touch [get_nets -all VSS]

check_physical_design -stage pre_route_opt; # dump check_physical_design result to file
./cpd_pre_route_opt_*/index.html

all_ideal_nets

all_high_fanout -nets -threshold 100

check_routeability

set_delay_calculation_options -arnoldi_effort low

set_route_options -groute_timing_driven true \

                -groute_incremental true \

                -track_assign_timing_driven true \
                -same_net_notch check_and_fix
set_si_options -route_xtalk_prevention true\
              -delta_delay true \
              -min_delta_delay true \
              -static_noise true\
              -timing_window true
route_auto -effort high
save_mw_cel -as ${design}_30_after_route_auto
verify_lvs -max_error 1000
verify_zrt_route
route_opt
save_mw_cel -as ${design}_31_after_route_opt
verify_lvs -max_error 1000
verify_zrt_route
route_opt -incremental -effort high
route_opt -incremental -effort high
route_opt -incremental -effort high
save_mw_cel -as ${design}_32_now
focal_opt -setup_endpoints all
focal_opt -drc_pins  all
focal_opt -drc_nets  all
route_opt -incremental -effort high
route_opt -incremental -effort high

route_opt -incremental -effort high

derive_pg_connection    -power_net VDD              \

                        -ground_net VSS       \

                        -power_pin VDD              \

                        -ground_pin VSS

#report_noise

#report_timing -crosstalk_delta

report_area > ./reports/area_route.rpt

report_cell > ./reports/cells_route.rpt

report_qor > ./reports/qor_route.rpt

report_resources > ./reports/resources_route.rpt

report_timing -max_paths 10 > ./reports/timing_route.rpt

report_power > ./reports/ power_route.rpt

save_mw_cel -as ${design}_6_routed

puts "finish_route"

#############################################

########### 7. Finishing ####################

#############################################

insert_stdcell_filler -cell_without_metal {FILLCELL_X32 FILLCELL_X16 FILLCELL_X8
FILLCELL_X4 FILLCELL_X2 FILLCELL_X1} \

        -connect_to_power VDD -connect_to_ground VSS
```

```
derive_pg_connection     -power_net VDD              \
                              -ground_net VSS       \
                              -power_pin VDD                \
                              -ground_pin VSS
save_mw_cel -as ${design}_7_finished
save_mw_cel -as ${design}
##############################################
########### 8. Checks and Outputs ############
##############################################
verify_zrt_route
verify_lvs -ignore_floating_port -ignore_floating_net \
        -check_open_locator -check_short_locator
set_write_stream_options  -map_layer
$sc_dir/tech/strmout/FreePDK45_10m_gdsout.map \
                   -output_filling fill \
                              -child_depth 20 \
                              -output_outdated_fill  \
                              -output_pin  {text geometry}
write_stream -lib $design \
              -format gds\
                     -cells $design\
                     ./output/${design}.gds
define_name_rules  new_verilog  -special  verilog  -target_bus_naming_style  {%s[%d]}  -
```

```
check_internal_net_name -check_bus_indexing

change_names -rule new_verilog -hierarchy

set verilogout_no_tri  true

set verilogout_equation  false

write_sdc ./output/${design}.sdc

write_verilog -pg -no_physical_only_cells ./output/cv32e40p_core_icc.v

write_verilog -no_physical_only_cells ./output/cv32e40p_core_icc_nopg.v

extract_rc

write_parasitics -output {./output/cv32e40p_core.spef}

close_mw_cel

close_mw_lib

exit
```

B.3 STA-Max script

```
set design cv32e40p_core

set  link_path  "*
/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_
End/Liberty/NLDM/NangateOpenCellLibrary_ss0p95vn40c.db"

read_verilog "../../../pnr/output/${design}_icc.v"

current_design $design

link

source ../../../pnr/output/${design}.sdc
```

```
read_parasitics ../../../pnr/output/${design}.spef.max

#read_parasitics -format spef ../../rcxt/cmax/cv32e40p_core_cmax_tm40.spef

set_propagated_clock [get_clocks clk_i]

update_timing

report_timing -delay_type max

fix_eco_timing -type setup -method size_cell -buffer_list {BUF_X1 BUF_X2 BUF_X3 BUF_X4}

write_changes -format icctcl -output ./ecol.tcl

save_session ${design}_max.session

report_constraint -all_violators -significant_digits 4 > ./${design}.max_constr.rpt

report_timing -delay_type max -nworst 40 -significant_digits 4 > ./${design}.max_timing.rpt

write_sdf ./${design}.max.sdf
```

B.4 STA-Min script

```
set design cv32e40p_core

set link_path "* /home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLDM/NangateOpenCellLibrary_ff1p25vn40c.db"

read_verilog "../../../pnr/output/${design}_icc.v"

current_design $design

link

source ../../../pnr/output/${design}.sdc
```

```
read_parasitics ../../../pnr/output/${design}.spef.min

#read_parasitics -format spef ../../rcxt/cmax/cv32e40p_core_cmax_tm40.spef

set_propagated_clock [get_clocks clk_i]

update_timing

report_timing -delay_type min

save_session ${design}_min.session

report_constraint -all_violators -significant_digits 4 > ./${design}.min_constr.rpt

report_timing -delay_type min -nworst 40 -significant_digits 4 >
./${design}.min_timing.rpt

write_sdf ./${design}.min.sdf
```

Appendix C

Scripts of Hierarchical Flow

C.1 Synthesis script

```
dc_shell-t | tee ./log/syn.log

set design cv32e40p_core

set_app_var  search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Fro
nt_End/Liberty/NLDM"

set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"

set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"

sh rm -rf work
sh mkdir -p work

define_design_lib work -path ./work

set hdlin_sverilog_std          2009

   analyze -library work -format sverilog ../rtl/cv32e40p_register_file_ff.sv

   analyze -library work -format sverilog ../rtl/include/cv32e40p_apu_core_pkg.sv

   analyze -library work -format sverilog ../rtl/include/cv32e40p_pkg.sv

   analyze -library work -format sverilog ../rtl/include/cv32e40p_fpu_pkg.sv

   analyze -library work -format sverilog ../bhv/ include/cv32e40p_tracer_pkg.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_alu.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_alu_div.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_ff_one.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_popcnt.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_compressed_decoder.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_controller.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_cs_registers.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_decoder.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_int_controller.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_ex_stage.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_hwloop_regs.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_id_stage.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_if_stage.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_load_store_unit.sv
   analyze -library work -format sverilog ../rtl/cv32e40p_mult.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_prefetch_buffer.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_prefetch_controller.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_obi_interface.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_aligner.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_sleep_unit.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_apu_disp.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_fifo.sv

   analyze -library work -format sverilog ../bhv/cv32e40p_sim_clock_gate.sv

   analyze -library work -format sverilog ../bhv/cv32e40p_wrapper.sv

analyze -library work -format sverilog  ../rtl/${design}.sv

puts "elaborate_stage"

elaborate $design -lib work

current_design

report_hierarchy > ./report/synth_hier.rpt

check_design

source ../constraints/cv32e40p_core.sdc

link

uniquify

compile_ultra -no_autoungroup -timing_high_effort_script -retime

report_timing -delay_type max

compile_ultra -no_autoungroup -timing_high_effort_script -retime -incremental
compile_ultra -no_autoungroup -timing_high_effort_script -retime -incremental

compile_ultra -no_autoungroup -timing_high_effort_script -retime -incremental

report_area > ./report/synth_area.rpt

report_power > ./report/synth_power.rpt

report_timing -max_paths 10 > ./report/synth_timing.rpt

write_sdc  output/${design}.sdc

define_name_rules  no_case -case_insensitive

change_names -rule no_case -hierarchy

change_names -rule verilog -hierarchy

set verilogout_no_tri   true

set verilogout_equation  false

write -hierarchy -format verilog -output output/${design}.v

write -f ddc -hierarchy -output output/${design}.ddc

exit
```

C.2 PnR scripts

```
#############################################

########### 1. DESIGN SETUP #################

#############################################

icc_shell -output_log_file ./log/pnr.log

set design cv32e40p_core

sh rm -rf $design

set sc_dir "/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12"

set_app_var  search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front
_End/Liberty/NLDM"

set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"

set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"

create_mw_lib   ./${design} \

            -technology $sc_dir/tech/techfile/milkyway/FreePDK45_10m.tf \

                  -mw_reference_library $sc_dir/lib/Back_End/mdb \

                  -open

set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"

set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"

set tech2itf "$sc_dir/tech/rcxt/FreePDK45_10m.map"

set_tlu_plus_files -max_tluplus $tlupmax \

              -min_tluplus $tlupmin \
                    -tech2itf_map $tech2itf

import_designs  ../syn/output/${design}.v \

          -format verilog \

                -top ${design} \

                -cel ${design}

source  ../constraints/cv32e40p_core.sdc

set_propagated_clock [get_clocks clk_i]

save_mw_cel -as ${design}_1_imported

#############################################

########### 2. Floorplan ####################

#############################################

## Create Starting Floorplan

###########################

puts "start floorplan"

create_floorplan -core_utilization 0.25 -core_aspect_ratio 2 \

        -start_first_row -flip_first_row \

        -left_io2core 12.4 -bottom_io2core 12.4 -right_io2core 12.4 -top_io2core 12.4

report_ignored_layers

remove_ignored_layers -all

set_ignored_layers -max_routing_layer metal8

## Initial Virtual Flat Placement

################################
```

```
set_app_var placer_max_cell_density_threshold 0.3

 create_fp_placement

 report_fp_placement

puts "end floorplan"

save_mw_cel -as ${design}_2_fp



##################################################

########### 3. POWER NETWORK ####################

##################################################

puts "start powergrid"

## Defining Logical POWER/GROUND Connections

############################################

derive_pg_connection        -power_net VDD          \

                    -ground_net VSS     \

                    -power_pin VDD          \

                    -ground_pin VSS

## Define Power Ring

####################

set_fp_rail_constraints  -set_ring -nets {VDD VSS}  \

            -horizontal_ring_layer {  metal9} \

            -vertical_ring_layer { metal10 } \

                    -ring_spacing 0.8 \
                              -ring_width 5 \

                              -ring_offset 0.8 \

                              -extend_strap core_ring



## Define Power Mesh

####################

set_fp_rail_constraints -add_layer  -layer metal10 -direction vertical   -max_strap 128 -
min_strap 20 -min_width 2.5 -spacing minimum

set_fp_rail_constraints -add_layer  -layer metal9  -direction horizontal -max_strap 128 -
min_strap 20 -min_width 2.5 -spacing minimum

set_fp_rail_constraints -add_layer  -layer metal8  -direction vertical   -max_strap 128 -
min_strap 20 -min_width 2.5 -spacing minimum

set_fp_rail_constraints -set_global

set die_llx [lindex [lindex [ get_attribute [get_die_area] bbox] 0] 0]

set die_lly [lindex [lindex [ get_attribute [get_die_area] bbox] 0] 1]

set die_urx [lindex [lindex [ get_attribute [get_die_area] bbox] 1] 0]

set die_ury [lindex [lindex [ get_attribute [get_die_area] bbox] 1] 1]



for {set i "[expr $die_llx + 20]"} {$i < "[expr $die_urx - 40]"} {set i [expr $i + 80]} {

         create_fp_virtual_pad -net VSS -point "{$i $die_lly}"

         create_fp_virtual_pad -net VDD -point "{[expr $i + 40] $die_lly}"



         create_fp_virtual_pad -net VSS -point "{$i $die_ury}"
```

```
        create_fp_virtual_pad -net VDD -point "{[expr $i + 40] $die_ury}"

}



for {set i "[expr $die_lly + 20]"} {$i < "[expr $die_ury - 40]"} {set i [expr $i + 80]} {

        create_fp_virtual_pad -net VSS -point "{$die_llx $i}"

        create_fp_virtual_pad -net VDD -point "{$die_llx [expr $i + 40]}"



        create_fp_virtual_pad -net VSS -point "{$die_urx $i}"

        create_fp_virtual_pad -net VDD -point "{$die_urx [expr $i + 40] }"

}

synthesize_fp_rail   -nets {VDD  VSS} -synthesize_power_plan -target_voltage_drop 22 -
voltage_supply 1.1 -power_budget 500

analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1

commit_fp_rail

set_preroute_drc_strategy -max_layer metal8

preroute_standard_cells -fill_empty_rows -remove_floating_pieces

## Add Well Tie Cells

####################

add_tap_cell_array -master   TAP \

                 -distance 30 \

                 -pattern  stagger_every_other_row

puts "end powergrid"
```

```
save_mw_cel -as ${design}_3_power

##############################################

########### 4. Placement #####################

##############################################

puts "start_place"

report_ignored_layers

check_physical_design -stage pre_place_opt

check_physical_constraints

## INITIAL PLACEMENT

place_opt

## check Congestion -> open global congestion map

# check timing

report_timing -delay_type max

report_timing -delay_type min

## OPTIMIZATION

###############

psynopt

##   ASSESSMENT

##################

check_legality

## check Congestion -> open global congestion map

# check timing
```

```
report_timing -delay_type max

report_timing -delay_type min

# DEFINING POWER/GROUND NETS AND PINS

derive_pg_connection    -power_net VDD              \

                        -ground_net VSS     \

                        -power_pin VDD              \

                        -ground_pin VSS

## Tie fixed values

###################

set tie_pins [get_flat_pins -all -filter "(constant_value == 0 || constant_value == 1) &&
name !~ V* && is_hierarchical == false "]



derive_pg_connection         -power_net VDD              \

                        -ground_net VSS     \

                        -tie

connect_tie_cells -objects $tie_pins \

           -obj_type port_inst \

                -tie_low_lib_cell  LOGIC0_X1 \

                -tie_high_lib_cell LOGIC1_X1

puts "finish_place"

save_mw_cel -as ${design}_4_placed

#############################################
```

```
########### 5. CTS      ####################

############################################



puts "start_cts"

## CHECKS

#########

check_physical_design -stage pre_clock_opt

check_clock_tree

report_clock_tree

set_driving_cell -lib_cell BUF_X16 -pin Z [get_ports clk_i]

### Set Clock Control/Targets

set_clock_tree_options \

            -clock_trees clk_i \

                  -target_early_delay 0.1 \

                  -target_skew 0.2 \

                  -max_capacitance 300 \

                  -max_fanout 10 \

                  -max_transition 0.150

set_clock_tree_options -clock_trees clk_i \

                  -buffer_relocation true \

                  -buffer_sizing true \

                  -gate_relocation true \
                    -gate_sizing true
## Selection of CTS cells
set_clock_tree_references -references [get_lib_cells */CLKBUF*]
### Set Clock Physical Constraints
define_routing_rule my_route_rule  \
  -widths   {metal3 0.14 metal4 0.28 metal5 0.28 metal6 0.28 metal7 0.8} \
  -spacings {metal3 0.14 metal4 0.28 metal5 0.28 metal6 0.28 metal7 0.8}
set_clock_tree_options -clock_trees clk_i \
                  -routing_rule my_route_rule  \
                         -layer_list "metal3 metal4 metal5 metal6 metal7"
set_clock_tree_options -use_default_routing_for_sinks 1
report_clock_tree -settings
## Clock Tree : Synhtesis, Optimization, and Routing
##################################################
compile_clock_tree


clock_opt -only_cts -no_clock_route
## analyze
   report_design_physical -utilization
   report_clock_tree -summary
   report_clock_tree
   report_clock_timing -type summary
   report_timing -delay_type max

   report_timing -delay_type min

   report_constraints -all_violators -max_delay -min_delay

## To Consider Hold Fix

  set_fix_hold [all_clocks]

  set_fix_hold_options -prioritize_tns -effort low

set_propagated_clock [all_clocks]

clock_opt -only_psyn -no_clock_route

route_group -all_clock_nets

derive_pg_connection    -power_net VDD            \

                        -ground_net VSS     \

                        -power_pin VDD            \

                        -ground_pin VSS

puts "end_CTS"

save_mw_cel -as ${design}_5_cts

#############################################

########### 6. ROUTING     #####################

#############################################

puts "start_route"

check_physical_design -stage pre_route_opt

all_ideal_nets

all_high_fanout -nets -threshold 100
```

```
check_routeability

set_delay_calculation_options -arnoldi_effort low

set_route_options -groute_timing_driven true \

                  -groute_incremental true \

                  -track_assign_timing_driven true \

                  -same_net_notch check_and_fix


set_si_options -route_xtalk_prevention true\

               -delta_delay true \

               -min_delta_delay true \

               -static_noise true\

               -timing_window true

 set_fix_hold [all_clocks]

   set_prefer -min  [get_lib_cells "*/BUF_X2 */BUF_X1"]

   set_fix_hold_options -preferred_buffer -effort low

set_propagated_clock [all_clocks]

route_auto

# check short nets

verify_lvs -max_error 1000

# check timing

report_timing -delay_type max

report_timing -delay_type min
```

```
# check max IR drop

analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1

save_mw_cel -as ${design}_6_routed

route_opt

# check short nets

verify_lvs -max_error 1000

# check timing

report_timing -delay_type max

report_timing -delay_type min

# check max IR drop

analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1

save_mw_cel -as ${design}_6_routed2

route_opt -incremental -effort high

# check short nets

verify_lvs -max_error 1000

# check timing

report_timing -delay_type max

report_timing -delay_type min

# check max IR drop

analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1

save_mw_cel -as ${design}_6_routed7
```

```
route_opt -incremental -effort high

# check short nets

verify_lvs -max_error 1000

# check timing

report_timing -delay_type max

report_timing -delay_type min

# check max IR drop

analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1

save_mw_cel -as ${design}_6_routed10

derive_pg_connection     -power_net VDD              \

                         -ground_net VSS       \

                         -power_pin VDD              \

                         -ground_pin VSS

save_mw_cel -as ${design}_6_routed4

puts "finish_route"



#############################################

########### 7. Finishing ####################

#############################################

insert_stdcell_filler -cell_without_metal {FILLCELL_X32 FILLCELL_X16  FILLCELL_X8
FILLCELL_X4 FILLCELL_X2 FILLCELL_X1} \

        -connect_to_power VDD -connect_to_ground VSS
```

```
insert_zrt_redundant_vias
derive_pg_connection    -power_net VDD              \
                        -ground_net VSS       \
                        -power_pin VDD              \
                        -ground_pin VSS
create_port -direction inout VDD
connect_net VDD [get_ports VDD]
create_port -direction inout VSS
connect_net VSS [get_ports VSS]
save_mw_cel -as ${design}_7_finished
save_mw_cel -as ${design}


##############################################
########### 8. Checks and Outputs ############
##############################################
verify_lvs -ignore_floating_port -ignore_floating_net \
       -check_open_locator -check_short_locator
verify_lvs -max_error 1000
report_timing -delay_type max
report_timing -delay_type min
analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1
set_write_stream_options  -map_layer
```

```
$sc_dir/tech/strmout/FreePDK45_10m_gdsout.map \

                  -output_filling fill \

                            -child_depth 20 \

                            -output_outdated_fill  \

                            -output_pin  {text geometry}



write_stream -lib $design \

             -format gds\

                    -cells $design\

                    ./output/${design}.gds

define_name_rules  new_verilog  -special  verilog  -target_bus_naming_style  {%s[%d]}  -
check_internal_net_name -check_bus_indexing

change_names -rule new_verilog -hierarchy

set verilogout_no_tri   true

set verilogout_equation  false

write_verilog -pg -no_physical_only_cells ./output/${design}_icc.v

write_verilog -no_physical_only_cells ./output/${design}_icc_nopg.v

report_area > ./report/pnr_area.rpt

report_power > ./report/pnr_power.rpt

report_cell > ./report/pnr_cells.rpt

report_qor  > ./report/pnr_qor.rpt

report_resources > ./report/pnr_resources.rpt
```

```
report_timing -max_paths 10 > ./report/pnr_timing.rpt



extract_rc

write_parasitics -output {./output/hierarch.spef}

create_rail_setup

close_mw_cel

close_mw_lib

exit
```

C.3 STA-Max script

```
set  link_path  "$link_path
/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_
End/Liberty/NLDM/NangateOpenCellLibrary_ss0p95vn40c.db"

read_verilog "../../../pnr/output/cv32e40p_core_icc.v"

current_design cv32e40p_core

link

source ../../../constraints/cv32e40p_core.sdc

set_propagated_clock [get_clocks clk_i]

read_parasitics ../../../pnr/output/hierarch.spef.max

#read_parasitics ../../rcxt/cmax/cv32e40p_core_cmax_tm40.spef

#fix_eco_timing -type setup
```

```
#update_timing



save_session cv32e40p_core_max_rcxt.session

report_constraint -all_violators -significant_digits 4 > ./cv32e40p_core.max_constr.rpt

report_timing -delay_type max -nworst 40 -significant_digits 4 >
./cv32e40p_core.max_timing.rpt

write_sdf ./cv32e40p_core.max.sdf

exit
```

C.4 STA-Min script

```
Set  link_path  "$link_path
/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_
End/Liberty/NLDM/NangateOpenCellLibrary_ff1p25v0c.db"

read_verilog "../../../pnr/output/cv32e40p_core_icc.v"

current_design cv32e40p_core

link

source ../../../constraints/cv32e40p_core.sdc

read_parasitics ../../../pnr/output/hierarch.spef.min

#read_parasitics ../../rcxt/cmin/cv32e40p_core_cmin_t125.spef

#fix_eco_timing -type hold -buffer_list {BUF_X16} -methods { insert_buffer}

#update_timing

save_session cv32e40p_core_min_rcxt.session

report_constraint -all_violators -significant_digits 4 > ./cv32e40p_core.min_constr.rpt
```

```
report_timing -delay_type min -nworst 40 -significant_digits 4 >
./cv32e40p_core.min_timing.rpt

write_sdf ./cv32e40p_core.min.sdf

exit
```

Appendix D

Scripts of Topographical Flow

D.1 Synthesis 1st stage

```
set design cv32e40p_core

set_app_var search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front
_End/Liberty/NLDM"

set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"

set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"

sh rm -rf work

sh mkdir -p work

define_design_lib work -path ./work

set hdlin_sverilog_std          2009

   analyze -library work -format sverilog ../rtl/cv32e40p_register_file_ff.sv

   analyze -library work -format sverilog ../rtl/include/cv32e40p_apu_core_pkg.sv

   analyze -library work -format sverilog ../rtl/include/cv32e40p_pkg.sv

   analyze -library work -format sverilog ../rtl/include/cv32e40p_fpu_pkg.sv
   analyze -library work -format sverilog ../bhv/include/cv32e40p_tracer_pkg.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_alu.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_alu_div.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_ff_one.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_popcnt.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_compressed_decoder.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_controller.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_cs_registers.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_decoder.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_int_controller.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_ex_stage.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_hwloop_regs.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_id_stage.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_if_stage.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_load_store_unit.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_mult.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_prefetch_buffer.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_prefetch_controller.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_obi_interface.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_aligner.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_sleep_unit.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_core.sv
```

```
   analyze -library work -format sverilog ../rtl/cv32e40p_apu_disp.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_fifo.sv

   analyze -library work -format sverilog ../bhv/cv32e40p_sim_clock_gate.sv

   analyze -library work -format sverilog ../bhv/cv32e40p_wrapper.sv

elaborate $design -lib work

current_design

check_design

##in Topo

#change_names -rules verilog -hier

source ./cons/cv32e40p_core.sdc

link

compile_ultra -timing_high_effort_script

report_timing -delay_type max

compile_ultra -timing_high_effort_script -incremental

report_timing -delay_type max

compile_ultra -timing_high_effort_script -incremental

report_timing -delay_type max

report_area > ./report/synth_area_CU_topo1.rpt

report_cell > ./report/synth_cells_CU_topo1.rpt

report_qor  > ./report/synth_qor_CU_topo1.rpt

report_resources > ./report/synth_resources_CU_topo1.rpt

report_timing -max_paths 10 > ./report/synth_timing_CU_topo1.rpt
report_timing -delay_type min > ./report/synth_timing_CU_topo_hold1.rpt

report_power > ./report/power_CU_topo1.rpt

define_name_rules  no_case -case_insensitive

change_names -rule no_case -hierarchy

change_names -rule verilog -hierarchy

set verilogout_no_tri   true

set verilogout_equation  false

write -hierarchy -format verilog -output output/${design}_1.v

write -f ddc -hierarchy -output output/${design}_1.ddc

write_sdc -version 1.9 output/${design}_1.sdc

exit
```

D.2 PnR 1st stage

```
##############################################

########### 1. DESIGN SETUP ##################

##############################################

set design cv32e40p_core

sh rm -rf $design

set sc_dir "/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12"

set_app_var  search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front
_End/Liberty/NLDM \
                              /home/mohamed/Desktop/johnson/rtl"
set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"
set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"
create_mw_lib  ./${design} \
            -technology $sc_dir/tech/techfile/milkyway/FreePDK45_10m.tf \
                   -mw_reference_library $sc_dir/lib/Back_End/mdb \
                   -open
set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"
set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"
set tech2itf "$sc_dir/tech/rcxt/FreePDK45_10m.map"
set_tlu_plus_files -max_tluplus $tlupmax \
              -min_tluplus $tlupmin \
                     -tech2itf_map $tech2itf
import_designs  ../syn/output/${design}_1.ddc \
            -format ddc \
                   -top ${design} \
                   -cel ${design}
source  ../syn/cons/cv32e40p_core.sdc
set_propagated_clock [get_clocks clk_i]
save_mw_cel -as ${design}_1_imported


#############################################
```

```
########### 2. Floorplan ####################

##################################################

## Create Starting Floorplan

############################

create_floorplan -core_utilization 0.25 \

        -core_aspect_ratio 2 \

        -start_first_row -flip_first_row \

        -left_io2core 12.4 -bottom_io2core 12.4 -right_io2core 12.4 -top_io2core 12.4


set_ignored_layers -max_routing_layer metal8 -min_routing_layer metal1

create_fp_placement -timing -no_hierarchy_gravity -congestion

create_fp_placement

save_mw_cel -as ${design}_1_fp

##################################################

########### 3. POWER NETWORK ####################

##################################################


derive_pg_connection        -power_net VDD          \

                        -ground_net VSS     \

                        -power_pin VDD          \

                        -ground_pin VSS
```

```
set_fp_rail_constraints  -set_ring -nets {VDD VSS}  \

                    -horizontal_ring_layer { metal9 } \

                    -vertical_ring_layer { metal10 } \

                                -ring_spacing 0.8 \

                                -ring_width 5 \

                                -ring_offset 0.8 \

                                -extend_strap core_ring



set_fp_rail_constraints -add_layer  -layer metal10 -direction vertical   -max_strap 128 -
min_strap 20 -min_width 2.5 -spacing minimum

set_fp_rail_constraints -add_layer  -layer metal9  -direction horizontal -max_strap 128 -
min_strap 20 -min_width 2.5 -spacing minimum

set_fp_rail_constraints -add_layer  -layer metal8  -direction vertical   -max_strap 128 -
min_strap 20 -min_width 2.5 -spacing minimum

set_fp_rail_constraints -set_global

set die_llx [lindex [lindex [ get_attribute [get_die_area] bbox] 0] 0]

set die_lly [lindex [lindex [ get_attribute [get_die_area] bbox] 0] 1]

set die_urx [lindex [lindex [ get_attribute [get_die_area] bbox] 1] 0]

set die_ury [lindex [lindex [ get_attribute [get_die_area] bbox] 1] 1]



for {set i "[expr $die_llx + 20]"} {$i < "[expr $die_urx - 40]"} {set i [expr $i + 80]} {

          create_fp_virtual_pad -net VSS -point "{$i $die_lly}"

          create_fp_virtual_pad -net VDD -point "{[expr $i + 40] $die_lly}"
```

```
        create_fp_virtual_pad -net VSS -point "{$i $die_ury}"

        create_fp_virtual_pad -net VDD -point "{[expr $i + 40] $die_ury}"

}



for {set i "[expr $die_lly + 20]"} {$i < "[expr $die_ury - 40]"} {set i [expr $i + 80]} {

        create_fp_virtual_pad -net VSS -point "{$die_llx $i}"

        create_fp_virtual_pad -net VDD -point "{$die_llx [expr $i + 40]}"



        create_fp_virtual_pad -net VSS -point "{$die_urx $i}"

        create_fp_virtual_pad -net VDD -point "{$die_urx [expr $i + 40] }"

}

synthesize_fp_rail   -nets {VDD VSS} -synthesize_power_plan -target_voltage_drop 22 -
voltage_supply 1.1 -power_budget 500

commit_fp_rail

set_preroute_drc_strategy -max_layer metal8

preroute_standard_cells -fill_empty_rows -remove_floating_pieces

analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1

add_tap_cell_array -master   TAP \

                -distance 30 \

                -pattern  stagger_every_other_row

save_mw_cel -as ${design}_3_power
```

```
change_names -rule verilog -dont_touch .

write_def -output ./output/defff.def

exit
```

D.3 Synthesis topographical mode

```
set design cv32e40p_core

set sc_dir "/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12"

set_app_var  search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front
_End/Liberty/NLDM \

                                /home/mohamed/Desktop/johnson/rtl"



set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"

set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"

set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"

set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"

set tech2itf "$sc_dir/tech/rcxt/FreePDK45_10m.map"

set_tlu_plus_files -max_tluplus $tlupmax \

               -min_tluplus $tlupmin \

                      -tech2itf_map $tech2itf
```

```
open_mw_lib ../pnr/cv32e40p_core

sh rm -rf work

sh mkdir -p work

define_design_lib work -path ./work

set hdlin_sverilog_std              2009

   analyze -library work -format sverilog ../rtl/cv32e40p_register_file_ff.sv

   analyze -library work -format sverilog ../rtl/include/cv32e40p_apu_core_pkg.sv

   analyze -library work -format sverilog ../rtl/include/cv32e40p_pkg.sv

   analyze -library work -format sverilog ../rtl/include/cv32e40p_fpu_pkg.sv

   analyze -library work -format sverilog ../bhv/ include/cv32e40p_tracer_pkg.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_alu.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_alu_div.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_ff_one.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_popcnt.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_compressed_decoder.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_controller.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_cs_registers.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_decoder.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_int_controller.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_ex_stage.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_hwloop_regs.sv

   analyze -library work -format sverilog ../rtl/cv32e40p_id_stage.sv
```

```
    analyze -library work -format sverilog ../rtl/cv32e40p_if_stage.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_load_store_unit.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_mult.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_prefetch_buffer.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_prefetch_controller.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_obi_interface.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_aligner.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_sleep_unit.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_core.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_apu_disp.sv

    analyze -library work -format sverilog ../rtl/cv32e40p_fifo.sv

    analyze -library work -format sverilog ../bhv/cv32e40p_sim_clock_gate.sv

    analyze -library work -format sverilog ../bhv/cv32e40p_wrapper.sv

elaborate $design -lib work

current_design

check_design

##in Topo

change_names -rule verilog -hierarchy

source ./cons/cv32e40p_core.sdc

link

extract_physical_constraints ../pnr/output/defff.def

compile_ultra -spg -timing_high_effort_script
compile_ultra -spg -timing_high_effort_script -incremental

compile_ultra -spg -timing_high_effort_script -incremental

compile_ultra -spg -timing_high_effort_script -retime

compile_ultra -spg -timing_high_effort_script -retime -incremental

report_area > ./report/synth_area_CU_topo2.rpt

report_cell > ./report/synth_cells_CU_topo2.rpt

report_qor  > ./report/synth_qor_CU_topo2.rpt

report_resources > ./report/synth_resources_CU_topo2.rpt

report_timing -max_paths 10 > ./report/synth_timing_CU_topo2.rpt

report_timing -delay_type min > ./report/synth_timing_CU_topo_hold2.rpt

report_power > ./report/power_CU_topo2.rpt




define_name_rules  no_case -case_insensitive

change_names -rule no_case -hierarchy

change_names -rule verilog -hierarchy

set verilogout_no_tri   true

set verilogout_equation  false

write -hierarchy -format verilog -output output/${design}_2.v

write -f ddc -hierarchy -output output/${design}_2.ddc

write_sdc -version 1.9 output/${design}_2.sdc

write_floorplan -all ./output/yara.fp
```

D.4 PnR 2nd stage

```
###############################################

########### 1. DESIGN SETUP ##################

###############################################

set design cv32e40p_core

set sc_dir "/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12"

set_app_var  search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front
_End/Liberty/NLDM \

                                  /home/mohamed/Desktop/johnson/rtl"

set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"

set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"

open_mw_lib cv32e40p_core

set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"

set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"

set tech2itf "$sc_dir/tech/rcxt/FreePDK45_10m.map"



set_tlu_plus_files -max_tluplus $tlupmax \

                -min_tluplus $tlupmin \

                        -tech2itf_map $tech2itf

import_designs  ../syn/output/${design}_2.ddc \
            -format ddc \
                  -top ${design} \
                  -cel ${design}
source  ../syn/cons/cv32e40p_core.sdc
save_mw_cel -as ${design}_1_imported
read_floorplan ../syn/output/yara.fp.objects
read_floorplan ../syn/output/yara.fp
set_ignored_layers -max_routing_layer metal8 -min_routing_layer metal1
set_preroute_drc_strategy -max_layer metal8
preroute_standard_cells -fill_empty_rows -remove_floating_pieces


##############################################
########### 4. Placement #####################
##############################################
puts "start_place"
set_app_var placer_max_cell_density_threshold 0.3
place_opt -spg -effort high -cts -congestion
refine_placement -congestion_effort high
psynopt -congestion
check_legality
derive_pg_connection     -power_net VDD              \
                         -ground_net VSS       \
                                -power_pin VDD                \
                                -ground_pin VSS
puts "finish_place"
save_mw_cel -as ${design}_4_placed
#############################################
########### 5. CTS      #####################
#############################################
puts "start_cts"
set_driving_cell -lib_cell BUF_X16 -pin Z [get_ports clk_i]
set_clock_tree_options \
            -clock_trees clk_i \
                  -target_early_delay 0.1 \
                  -target_skew 0.25 \
                  -max_capacitance 300 \
                  -max_fanout 10 \
                  -max_transition 0.15
set_clock_tree_options -clock_trees clk_i \
                  -buffer_relocation true \
                  -buffer_sizing true \
                  -gate_relocation true \
                  -gate_sizing true
set_clock_tree_references -references [get_lib_cells */CLKBUF*]
```

```
define_routing_rule my_route_rule  \
  -widths   {metal3 0.14 metal4 0.28 metal5 0.28} \
  -spacings {metal3 0.14 metal4 0.28 metal5 0.28}
set_clock_tree_options -clock_trees clk_i \
                    -routing_rule my_route_rule  \
                            -layer_list "metal3 metal4 metal5"
set_clock_tree_options -use_default_routing_for_sinks 1
report_clock_tree -settings
compile_clock_tree  -clock_trees clk_i
clock_opt -only_cts -no_clock_route
report_timing
save_mw_cel -as ${design}_CTS1
set_fix_hold [all_clocks]
set_fix_hold_options -prioritize_tns
set_propagated_clock [all_clocks]
clock_opt -only_psyn -no_clock_route
save_mw_cel -as ${design}_CTS2
route_group -all_clock_nets
save_mw_cel -as ${design}_CTS3
derive_pg_connection    -power_net VDD              \
                                  -ground_net VSS       \
                                  -power_pin VDD                  \
                        -ground_pin VSS

save_mw_cel -as ${design}_5_cts

puts "finish_cts"

##############################################

########### 6. Routing   ####################

##############################################



## Before starting to route, you should add spare cells

insert_spare_cells -lib_cell {NOR2_X4 NAND2_X4} \

                  -num_instances 20 \

                  -cell_name SPARE_PREFIX_NAME \

                  -tie

set_dont_touch  [all_spare_cells] true

set_attribute [all_spare_cells]  is_soft_fixed true

##############################################

puts "start_route"

check_physical_design -stage pre_route_opt; # dump check_physical_design result to file
./cpd_pre_route_opt_*/index.html

all_ideal_nets

all_high_fanout -nets -threshold 100

check_routeability

set_delay_calculation_options -arnoldi_effort low
```

```
set_route_options -groute_timing_driven true \
                -groute_incremental true \
                -track_assign_timing_driven true \
                -same_net_notch check_and_fix


set_si_options -route_xtalk_prevention true\
              -delta_delay true \
              -min_delta_delay true \
              -static_noise true\
              -timing_window true
route_auto -effort high
save_mw_cel -as ${design}_route_auto
verify_lvs -max_error 1000
report_timing -delay_type max
report_timing -delay_type min
route_opt -effort high
verify_lvs -max_error 1000
report_timing -delay_type max
report_timing -delay_type min
save_mw_cel -as ${design}_route_opt1
route_opt -effort high
route_opt -incremental -effort high ///do it five times
```

```
verify_lvs -max_error 1000

report_timing -delay_type max

report_timing -delay_type min

save_mw_cel -as ${design}_route_inc_effort

psynopt  -only_hold_time

save_mw_cel -as ${design}_psyn

route_zrt_eco -open_net_driven true

verify_lvs -max_error 1000

report_timing -delay_type max

report_timing -delay_type min

save_mw_cel -as ${design}_zrt_eco

verify_zrt_route

route_zrt_detail -initial_drc_from_input true

save_mw_cel -as ${design}_zrt_detail

verify_lvs -max_error 1000

report_timing -delay_type max

report_timing -delay_type min

route_zrt_detail -incremental true -initial_drc_from_input true

save_mw_cel -as ${design}_zrt_detail_2

verify_lvs -max_error 1000

report_timing -delay_type max

report_timing -delay_type min
```

```
derive_pg_connection    -power_net VDD              \

                          -ground_net VSS       \

                          -power_pin VDD                \

                          -ground_pin VSS

save_mw_cel -as ${design}_6_routed

save_mw_cel -as ${design}

puts "finish_route"

verify_zrt_route

verify_lvs -ignore_floating_port -ignore_floating_net \

       -check_open_locator -check_short_locator



set_write_stream_options -map_layer
$sc_dir/tech/strmout/FreePDK45_10m_gdsout.map \

                -output_filling fill \

                          -child_depth 20 \

                          -output_outdated_fill  \

                          -output_pin  {text geometry}

write_stream -lib $design \

            -format gds\

                  -cells $design\

                  ./output/${design}.gds

define_name_rules new_verilog -special verilog -target_bus_naming_style {%s[%d]} -
check_internal_net_name -check_bus_indexing
```

```
change_names -rule new_verilog -hierarchy

set verilogout_no_tri  true

set verilogout_equation  false

write_verilog -pg -no_physical_only_cells ./output/${design}_icc.v

write_verilog -no_physical_only_cells ./ output/${design}_icc_nopg.v

set write_sdc_output_lumped_net_capacitance false

set write_sdc_output_net_resistance false

write_sdc -version 1.9 output/conss2.sdc

write -format ddc -hier -output output/ ddc_netlist_core2.ddc

extract_rc

write_parasitics -output {./ output/cv32e40p_core1.spef}

create_rail_setup

close_mw_cel

close_mw_lib

exit
```