ASIC Implementation of PULPissimo Microcontroller

By

Ahmed Hamouda Abdelhafeez

Ahmed saied

Hossam Adel Elsayed

Mohamed Adel

Osama Mostafa

A Thesis Submitted to the

Faculty of Engineering at Cairo University

in Partial Fulfillment of the

Requirements for the Degree of

BACHELOR OF SCIENCE

in

Electronics and Electrical Communications Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

2022

ASIC Implementation of PULPissimo Microcontroller

By

Ahmed Hamouda Abdelhafeez

Ahmed saied

Hossam Adel Elsayed

Mohamed Adel

Osama Mostafa

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of

BACHELOR OF SCIENCE

in

Electronics and Electrical Communications Engineering

Under the supervision of

Dr. Hassan Mostafa

Associate Professor
Electronics and Communications Department
Faculty of Engineering, Cairo university

Dr. Amin Nassar

Emeritus Professor
Electronics and Communications Department
Faculty of Engineering, Cairo university

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2022

Contents

List of Figures:

List of Tables

1. Introduction

1.1. PULP:

The Parallel Ultra Low Power (PULP) platform attempts to bridge the gap between academic research and the development of industrially relevant microcontroller systems specifically tailored for low power processing. It is a young project launched in 2013 as a collaboration between the Integrated Systems Laboratory (IIS) at ETH Zürich, Switzerland and Energy-efficient Embedded Systems (EEES) group at the University of Bologna, Italy.

1.2. The Need to low power designs:

The design of power-efficient system on chip (SoC) is of significance importance to the development of the applications with a built-in battery that targets operations with a large-number of computations to increase the life-time of the applications, therefore increases the reliability of the designs.

The Internet of Things (IoT) applications is an example of these previously mentioned applications and the PULP designs are aimed for these kind of applications.

1.3. PULPissimo compared to other PULP products:

PULP products can be summarized as following:

1) Multi-core microcontroller:
    - PULP microcontroller.
2) Single-core microcontroller:
    - PULPino microcontroller.
    - PULPissimo microcontroller.

The previously mentioned summary shown, the PULPissimo is like PULPino as both are a single-core microcontroller, however the PULPissimo is significantly advanced version compared to the PULPino in terms of complexity and completeness and it is used as a main System-on-Chip controller for all recent multi-core PULP chips.

The PULPissimo is an open source advanced microcontroller System on Chip (SoC), readily available as RTL as part of the open source initiative PULP platform.

1.4. Thesis Objective

RTL-to-GDSII Flow plays an important role in the development process of electronic chips over the world. Due to the huge competition in this field and huge customer demands, the technology in this field is developing at a fast pace towards smaller, faster and more complex devices which is making it harder for a newcomer to cope up with this field without learning and understanding the basics first. CAD tools nowadays offer more than one ASIC design style to satisfiy huge customer demands and improve QoR and TTR of ASIC designs. Objective of this thesis is to go through the core ASIC implementation design steps, i.e. Logic synthesis and Placement and Routing using Synopsys Computer Aided Design (CAD) tools and SAED32nm CMOS technology library.

1.5. Thesis Map

After introducing the main goal of this thesis, in chapter two we will present the architecture, of the PULPissimo SoC focusing on its key relevant features of relevance to our design. In chapter three we focus on our backend design of the chip in three implementation approaches called hierarchal, topographical and flat flows. We present our results and discuss them in chapter 4 and lay the ground for future work in chapter 5. We supplement all of this with appendix giving necessary ASIC background.

2. PULPissimo Overview

2.1 Main blocks in the PULPissimo:

The main blocks in the PULPissimo can be summarized as following:

1) Either the RI5CY core or the Ibex one as main core
2) Autonomous Input/Output subsystem (uDMA)
3) New memory subsystem (interleaved and private memories).
4) Support for Hardware Processing Engines (HWPEs)
5) New simple interrupt controller
6) Peripherals

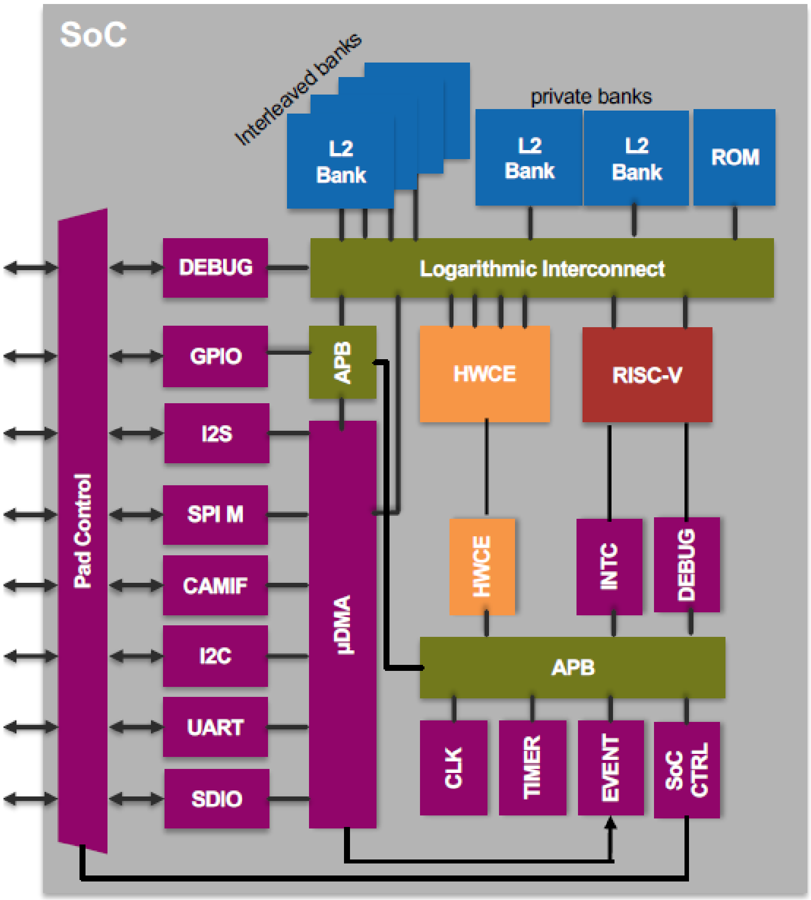


Figure 1 simplified block diagram of PULPissmo SoC

And they are discussed in the below sections

2.1.1 The RI5CY and Ibex cores:

1) The zero-riscy core:

- In-order, single-issue core with 2 pipeline stages.
- It has full support for the base integer instruction set (RV32I) and compressed instructions (RV32C).
- It can be configured to have multiplication instruction set extension (RV32M) and the reduced number of registers extension (RV32E).
- It has been designed to target ultra-low-power and ultra-low-area constraints.

2) The RISCY core:

- In-order, single-issue core with 4 pipeline stages and it has an instruction per cycle (IPC) close to 1.
- Full support for the base integer instruction set (RV32), compressed instructions (RV32C) and multiplication instruction set extension (RV32M).
- It can be configured to have single-precision floating-point instruction set extension (RV32F).
- It implements several ISA extensions such as: hardware loops, post-incrementing load and store instructions, bit-manipulation instructions, MAC operations, support fixed-point operations, packed-SIMD instructions and the dot product.
- It has been designed to increase the energy efficiency of in ultra-low-power signal processing applications.

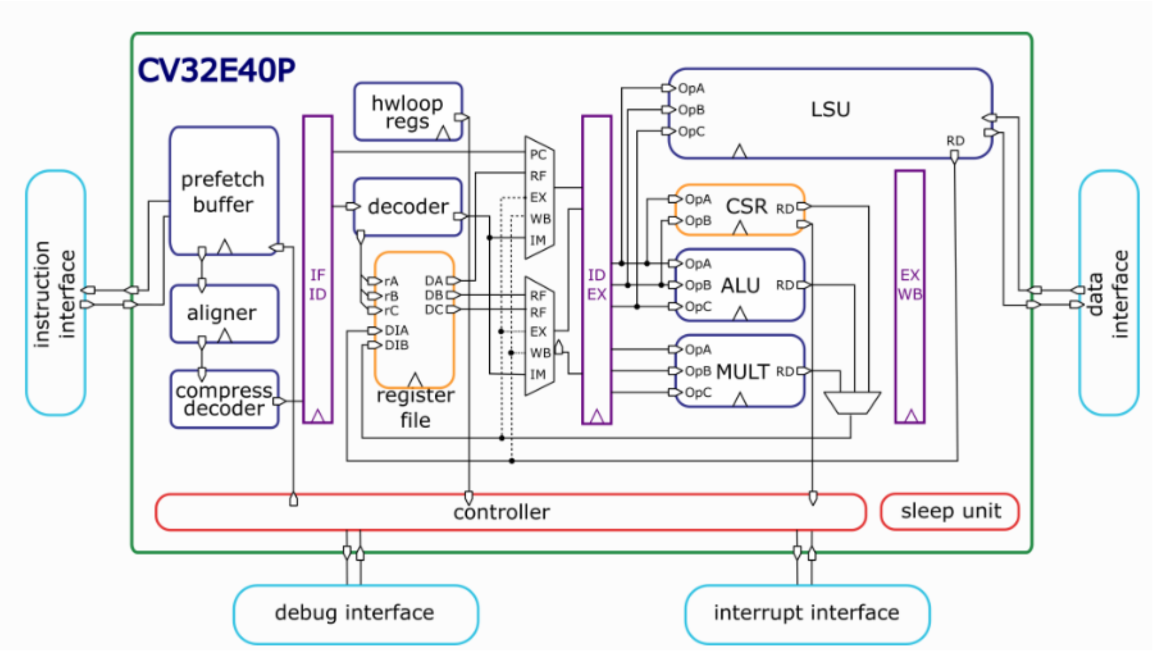


Figure 2: block diagram of the RISCY core at the heart of PULPissimo SoC

From the previous compared, it appears that the RISCY core is better than the zero-riscy core, therefore it is chosen in the design of this thesis.

2.1.2 Autonomous Input/Output subsystem (uDMA):

PULPissimo includes a new efficient I/O subsystem via a uDMA (pronounced "micro-DMA") which communicates with the peripherals And provides to them an autonomous access to the interleaved memory, therefore the peripherals can directly communicate with the interleaved memory without communicating with it through the core, which reduces the latency and increase the overall performance.

2.1.3 Interleaved and private memories:

The interleaved memory can be accessed by core, APB, uDMA and debug units through a logarithmic interconnect which provides a low-latency access for the interleaved and private memories. This memory is made in interleaved configuration to minimize conflicts during parallel accesses of different masters (e.g., core and accelerator). And it consists of 4 memory banks, each one of them is size of $2^{15} * 32$ which is a 128KB memory size.

The private memory can be accessed by the core only, therefore it is called a private memory. And it consists of 2 memory banks, each one of them is size of $8192\ (2^{13}) * 32$ which is a 32KB memory size.

2.1.4 Support for Hardware Processing Engines (HWPEs):

The HWPE has its own master ports to memories within the system. So the HWPE is programmed by the core through the APB through its configuration interface, where the core tells the HWPE to start address, or to find, the data to process and where the results should be written to, and then the HWPE fetches the data from the memory independently without any interaction from the core. The core will be notified as soon as the operation is finished through a certain event line.

2.1.5 New simple interrupt controller:

The interrupt controller has 32 interrupt lines only. And as the SoC has many interrupt lines, than the 32 interrupt lines that are in the interrupt controller, an another block called SoC event generator receives several interrupt lines from the peripherals and HWPE and maps them to 2 lines of the 32 interrupt lines in the interrupt controller.

2.1.6 Peripherals:

The main peripherals in the PULPissimo are:

1) SPI (flash memory): The SPI stands for Serial Peripheral Interface. It’s a simple serial protocol that can talk to a variety of devices, including serial flash devices. The SPI Flash is simply the cheapest, simplest way of building off-chip non-volatile flash memory at the moment.

2) I2S: is an electrical serial bus interface standard used for connecting digital audio devices together. It is used to communicate PCM audio data between integrated circuits in an electronic device. And it consists of a SCK (Serial Clock), which is the clock signal also referred as BCLK (Bit Clock Line), FS (Frame Select) which is used to discriminate Right or Left Channel data also referred WS (Word Select) and SD (Serial Data) which is the serial data to be transmitted.

3) Camera Interface (CPI): In camera peripheral, RGB565 camera are used and it is especially (cheap) screens used with embedded devices do not provide 24 bit color-depth. Only provides 2^16=65 536 colors.

4) I2C: alternatively known as I2C or IIC, is a synchronous, multi-controller/multi-target, packet switched, single-ended, serial communication bus

5) UART: A universal asynchronous receiver-transmitter is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable.

6) JTAG: named after the Joint Test Action Group which codified it. And It specifies the use of a dedicated debug port implementing a serial communications interface for low-overhead access without requiring direct external access to the system address and data buses.

7) 4 Timers

8) ROM

2.2 Main domains in PULPissimo:

Figure (3) shows a simplified block diagram showing the main modules of the chip. The top module of the PULPissimo is a wrapper that includes three submodules, called domains and a set of different high level control signals. The 3 domains are called SoC domain, safe domain and pad frame.

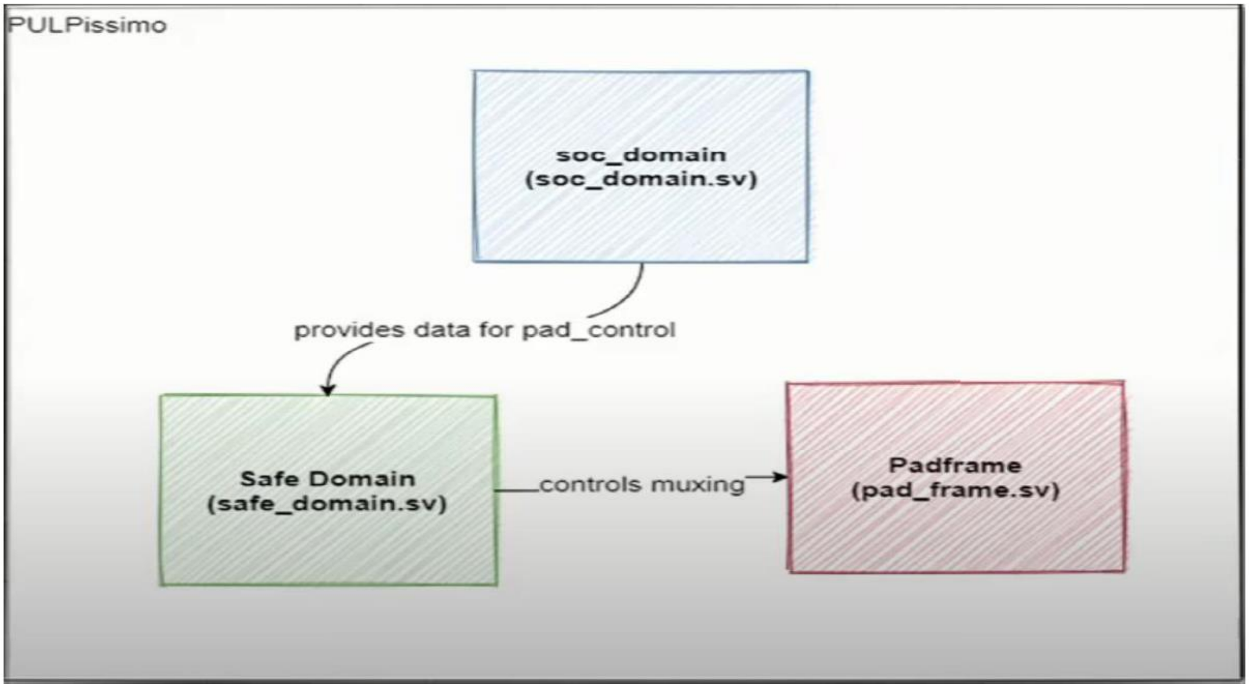


Figure 3: the three main domains of PULPissimo

And the Reasons behinds this segmentation are as follows:

1) To enable the usage of different voltage levels.
2) To turn off the power from some parts of the system during the idle duration, to save power, without affecting the essential operations of the core.

The three domains are discussed in the below sections.

2.2.1 SoC domain:

The SoC domain is what wraps most of the chip's core logic, it contains key functional modules such as the RISC-V core, the interleaved and private memory banks, the debug modules and others. The figure below (figure 4) gives an abstract view of the SoC domain submodules and their interconnectivity. We can see that the SoC domain consists of 5 major modules: soc_interconnect_wrap, fc_subsystem, boot_rom, L2_ram_multibank and soc_peripherals. For the rest of this subsection we will dig deeper into each of these components.

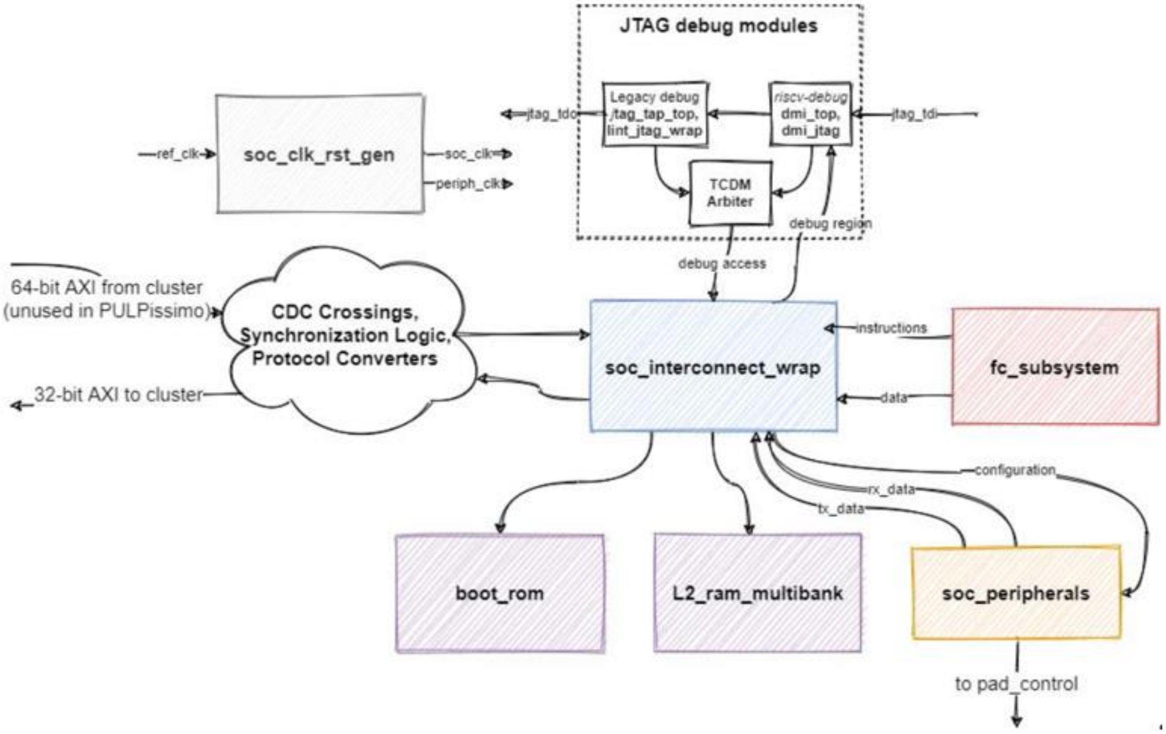


Figure 4: Main blocks in the SoC domain

These blocks are discussed in the below sections.

2.2.1.1 The Fabric Control subsystem:

The FC, short for Fabric Control, subsystem is what wraps the RISC-V core of the SoC, in PULPissimo this is the place where the hardware processing engine would be plugged in if needed. The reason why they are placed in the same module even though they do not interact with each other directly but through the SoC interconnect is to simplify voltage gating as these are the main power intensive modules in the design. In our implementation we did not consider hardware accelerators. The FC subsystem also includes an interrupt controller that handles the passing of interrupts from the SoC peripherals sub-domain (which will be explained shortly) and the RISC core.

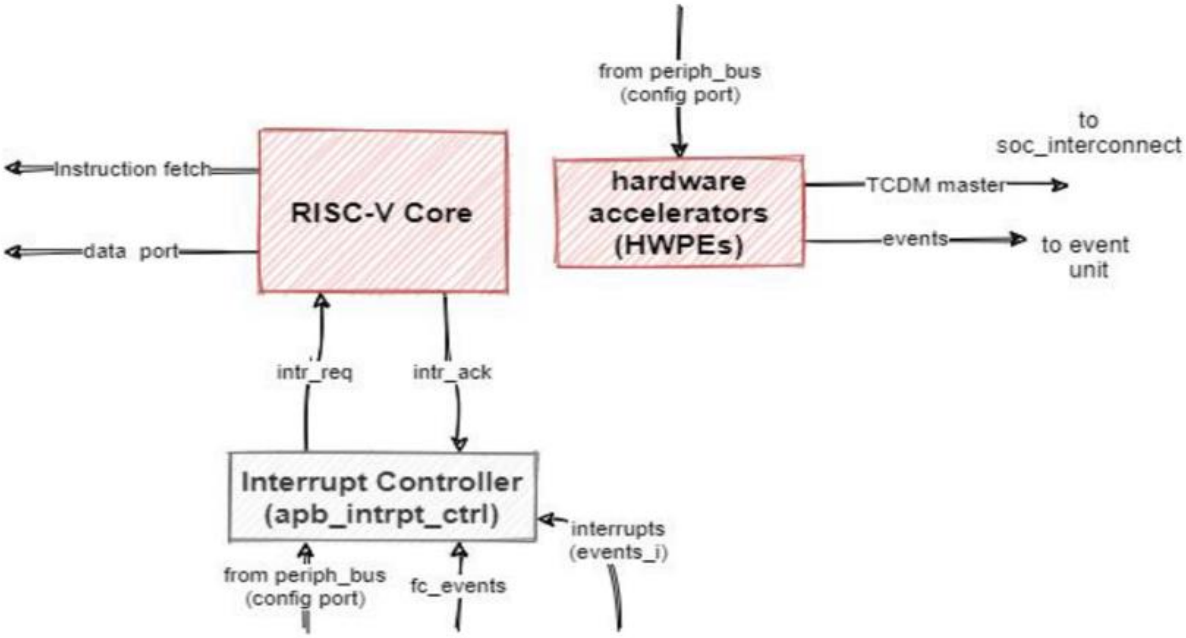


Figure 5: schematic showing the FC subsystem

2.2.1.2 SoC peripherals:

This is where the different peripheral the SoC could support are to be found it contains modules such as:

1) **Hardware event controller**: which takes as input any of the different interrupts the peripherals (or the hardware processing engine, if it exists) can send and maps it to one of the 32 interrupt lines of our RISC-V core through the interrupt controller.

2) **GPIO controller:** provides basic gpio functionalities like reading or setting their different values.

3) **µDMA subsystem:** this allows all peripheral, wrapped within this module like I2C, I2S, camera interface, etc to send or receive data directly to or from the main memory, bypassing the CPU minimize the workload on the processor improving the speed.

4) **peripheral bus wrap:** which just de-multiplexes the transactions to the different peripherals.

5) **SoC Control Registers (apb_soc_control):** A register file with an APB interface that stores configuration signals that are global to the whole SoC.

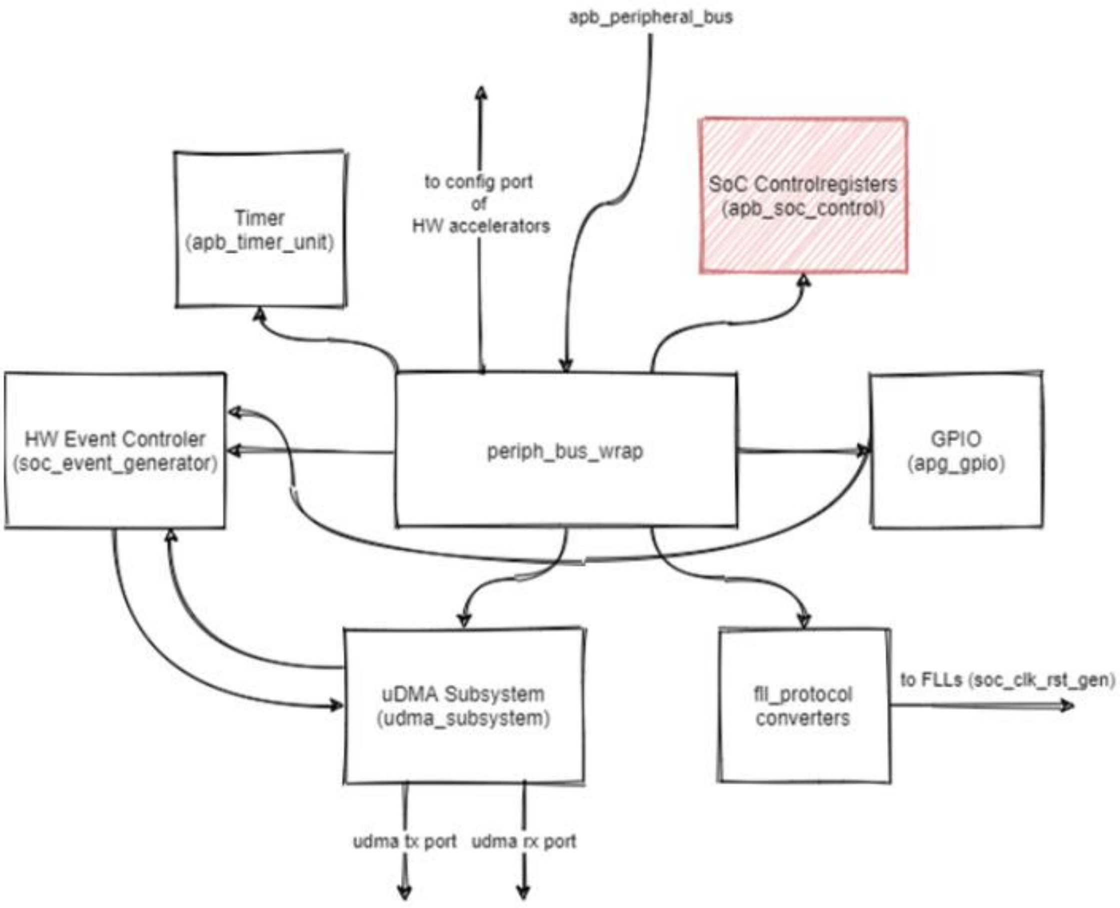


Figure 6: schematic showing the soc peripherals module

2.2.1.3 L2 Ram multibank:

This acts as a wrapper for memory banks of both interleaved and private memories. All memory banks are made of a SRAM IP that is provided by the vendor of used technology library instead of latch and flip-flop to implement these memories in reasonable area with a reasonable power consumption.

The memories are slaves of the system bus, which is based on a single-cycle latency logarithmic interconnect.

2.2.1.4 SoC Interconnect wrap:

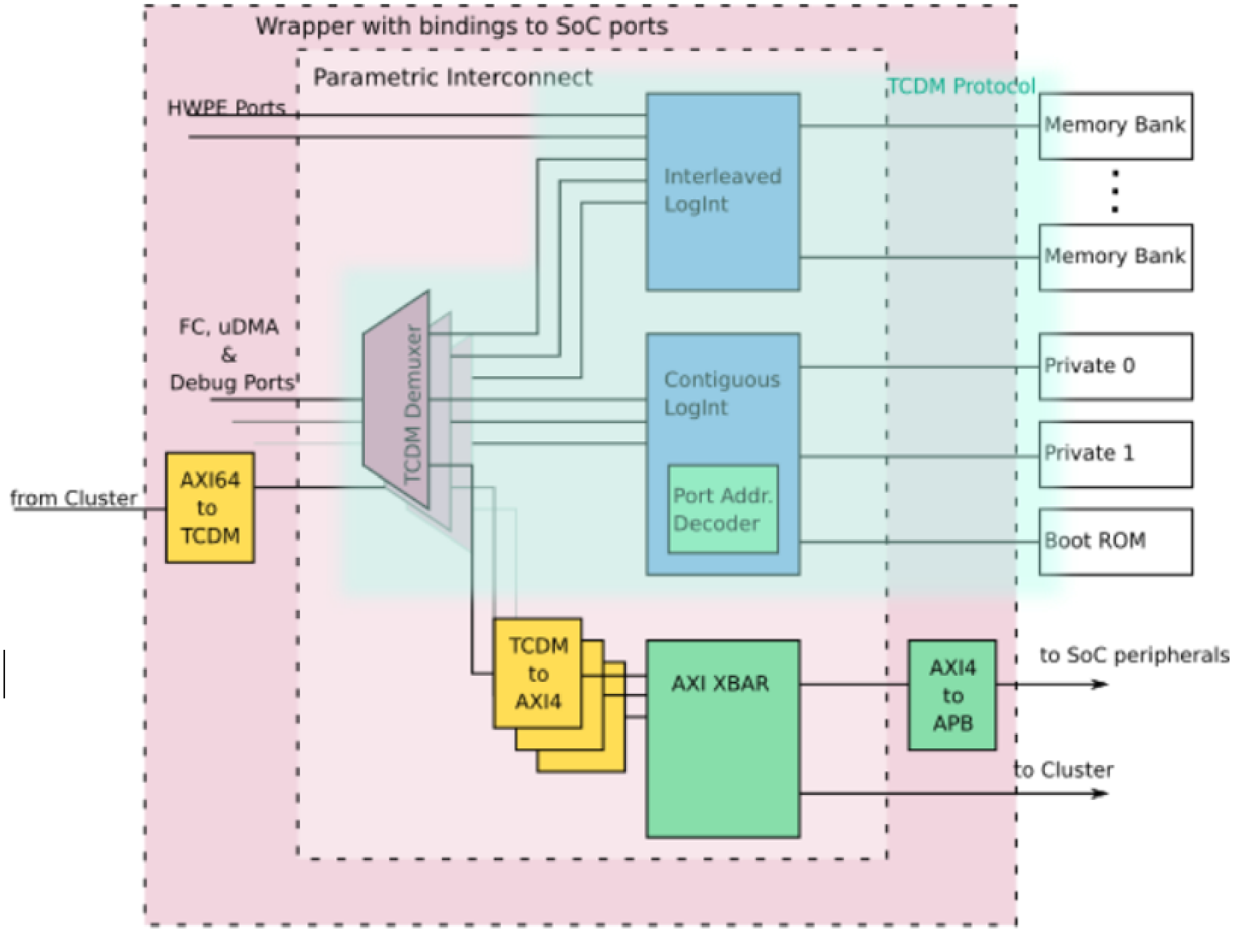


Figure 7: schematic showing SoC interconnect wrap

The SoC Interconnect wrap arbitrates three masters FC, uDMA and Debug Ports. It has three TCDM de-multiplexers, one for each master, which de-multiplexes signals (transactions) from masters, according to their addresses, to three different domains the first is the interleaved region, the second is the contiguous region and the third is the AXI crossbar domain. And it can summarized in points as following:

- Blocks in blue color are Logarithmic interconnects that uses TCDM protocol that allows single cycle latency memory access, but it’s not the best option for accessing peripherals having variable delays.

- The Interleaved Region uses Interleaved logarithmic Interconnect (LogInt) to access an interleaved memory banks where memory is divided into number of separate banks (modules) over which sequential words of data are interleavingly mapped, this would reduce memory conflicts between different masters accessing the memory.

- The contiguous region uses Contiguous Logarithmic Interconnect to access three contiguous memory banks where words are mapped sequentially into them. Only the Core, which is included in FC master, is allowed to access those memory banks, this would improve the performance of the core. Each of the three banks has a certain type of data to be stored in it, where Bank 0 is for Instructions, Bank 1 is for the stack and Bank 2 is for Boot ROM.

- AXI Crossbar which is used for accessing peripherals having variable delays instead of the LogInt, it has three TCDM to AXI blocks one for each master. It has two output ports the first is for SoC Peripherals through APB, while the second port is for the Cluster domain.

2.2.1.5 Boot ROM:

Besides the RAM, PULPissimo also includes a Read-only-memory (ROM) that has been implemented to store the boot instructions responsible for setting the system upon reset.

2.2.1.6 Debug blocks:

The PULPissimo contains two debug units which are a RSICY debug module and legacy debug module.

1) The **RSIC-V debug** module consists of 2 units: debug module called "dm_top" module and a debug module interface module called "dmi_jtag" module, where

   The "dm_top" module allows:

   - basic debug functionality of the core, like single stepping and break points.
   - Allows to access the system bus (the logarithmic interconnect).

   The "dmi_jtag" module provides a jtag access to the RSIC-V debug unit.

2) The **Legacy debug unit** consists of 2 modules that are called "jtag_tap_top" and "lint_jtag_wrap" modules and Reasons of this unit:

   - If something goes wrong with the RSIC-V debug unit, the legacy debug unit provides access to the system to be debugged.

   - It is faster with the memory transfer than the RSIC-V debug unit, through its jTAG interface.

2.2.1.7 SoC clock and reset generator:

The PULPissimo has three clock domains called SoC, peripheral and reference clock domains, where the SoC and peripheral clock domains are generated from a reference input frequency signal using 2 separate, internal phase lock loops (PLLs). And the reason behind using different clock domains is to allow the different domains to operate at different speeds which helps with the overall performance as the SoC is not as constrained by the slower peripheral clock and can have its own clock.

2.2.2 Safe domain:

This is the domain that contains every module that must be always powered on, the power of this domain cannot be turned off. And these modules are:

1) **The pad controller:** effectively a multiplexer between the different module output signals to the IO pads. Note that this control is needed because the IO pads of PULPissimo aren't as many as what the modules require, this is related to the rich set of peripherals that the system can support. So, we multiplex the outputs signals coming from the different PULPissimo modules, and connect the output of the MUX to the output pad.

2) **Reset Generator:** Synchronizes the reset signals to the ref. clock (32 KHz) and the resynchronized signals are only used with the Safe domain.

In principle, more logic that needs to be power gated could be added to this domain.

2.2.3 Pad frame:

This is where Input/Output (IO) pads are included. The signals connected to each pad are logic signals from the SoC to IO pads and vice versa, output driver enable, the actual pad signal, and additional Configuration signals. The table 1 below shows the different signals with their description and defined direction as they are defined in its SystemVerilog module;

Table 1 Padframe signals

Direction (Pad frame perspective)	**Name**	**Description**
Input	Oe_<padname>_i	Active high output driver enable.
Output	in_<padname>_o	Logic Signal from pad to SoC.
Input	Out_<padname>_i	Logic signal from SoC to pad.
Inout	pad_<padname>	The actual pad signal that is connected to the top level.
Input	Pad_cfg_i	Additional configuration signals for pads.

3. The actual implementation of the PULPissimo:

CAD tools offers many flows to implement the RTL-GDS (ASIC) flow. During this thesis, we will use three flows and compare the results among them. The parameter of comparison will be area, power and runtime. Here are the chosen flows:

3.1 Constraints of the three chosen flows:

The constraints mainly extracted from the used technology library in this thesis which is the **Synopsys SAED 32nm**. And they consist of electrical DRCs and timing constraints. We choose to design on the worst case process variation which is slow-slow at 0.9 voltage at temperature negative 40 degree of Celsius based on the trial and error method and some other criteria. And these constraints and the criteria, by which the constraints are chosen, are discussed the following sections.

3.1.1 Electrical DRC constraints:

The electrical DRC constraints extracted from the technology library:

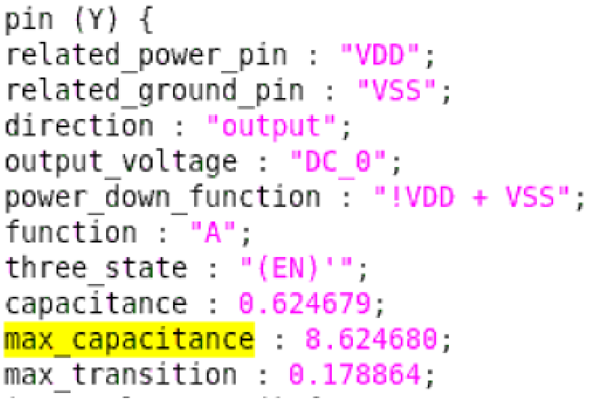
Table 2 DRC constraints of the PULPissimo implemented in the topographical flow

DRC constraint name	Value
Maximum capacitance	416 fF
Maximum fan-out number	20
Maximum transition	1.024 ns

3.1.1.1 Maximum capacitance and maximum transition:

Where the maximum capacitance was extracted from collecting all the values of the “max_capacitance” attribute for each pin of each version of the standard cells, which is done by the code in the appendix C part 1, choosing the maximum value among them. The figure (figure 28) below shows an example of the “max_capacitance” attribute in the Synopsys SAED 32nm technology library.

The maximum transition extraction is very similar to the maximum capacitance extraction as there is an attitude called “max_transtion” for each pin of each version of the standard cells, which is done by the code in the appendix C part 2, choosing the maximum value among them

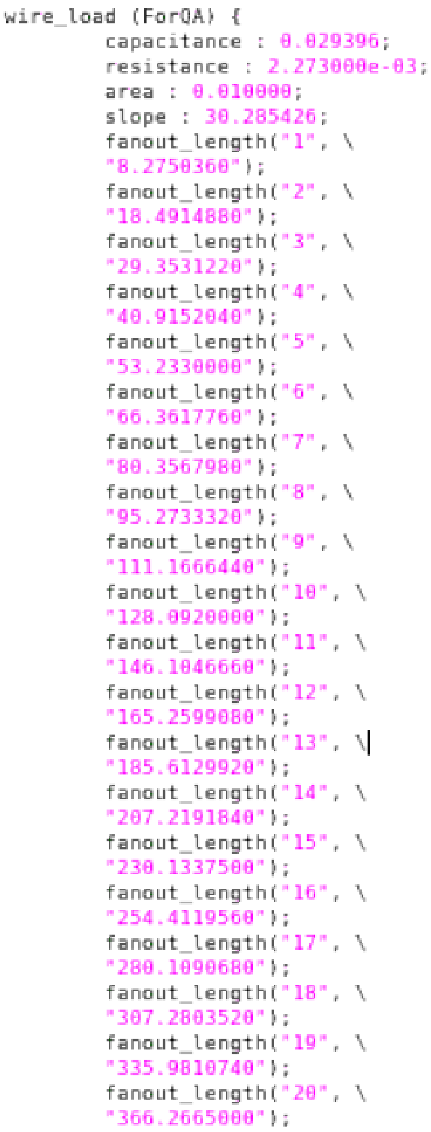


```
pin (Y) {
related_power_pin : "VDD";
related_ground_pin : "VSS";
direction : "output";
output_voltage : "DC_0";
power_down_function : "!VDD + VSS";
function : "A";
three_state : "(EN)'";
capacitance : 0.624679;
max_capacitance : 8.624680;
max_transition : 0.178864;
```

Figure 8: an example of the “max_capacitance” attribute

3.1.1.2 Maximum fan-out number:

The maximum fan-out number is chosen based on the attached wire load model (WLM) in the technology library, which is written in the following manner: “fanout_length function (fanout number, estimated net length), and the figure below shows an example of the wire load model.



```
wire_load (ForQA) {
        capacitance : 0.029396;
        resistance : 2.273000e-03;
        area : 0.010000;
        slope : 30.285426;
        fanout_length("1", \
        "8.2750360");
        fanout_length("2", \
        "18.4914880");
        fanout_length("3", \
        "29.3531220");
        fanout_length("4", \
        "40.9152040");
        fanout_length("5", \
        "53.2330000");
        fanout_length("6", \
        "66.3617760");
        fanout_length("7", \
        "80.3567980");
        fanout_length("8", \
        "95.2733320");
        fanout_length("9", \
        "111.1666440");
        fanout_length("10", \
        "128.0920000");
        fanout_length("11", \
        "146.1046660");
        fanout_length("12", \
        "165.2599080");
        fanout_length("13", \
        "185.6129920");
        fanout_length("14", \
        "207.2191840");
        fanout_length("15", \
        "230.1337500");
        fanout_length("16", \
        "254.4119560");
        fanout_length("17", \
        "280.1090680");
        fanout_length("18", \
        "307.2803520");
        fanout_length("19", \
        "335.9810740");
        fanout_length("20", \
        "366.2665000");
```

Figure 9: example of the wire load model

3.1.2 Timing constraints:

The PULPissimo has 3 different master clock domains, which are called “ref_clk, soc_clk and per_clk”, where the clock source of the reference clock (ref_clk) domain comes from outside the chip and enters the chip through a pad called “pad_xtal_in”. and the clock source of the two other clock domain, SoC clock domain and the peripheral clock domain, are generated by 2 different phase lock loop (PLL) IPs which are internally placed in the PULPissimo microcontroller and the reference clock source of the two PLL IPs is the same clock source of the reference clock domain.

The following table shows all timing constraints for each clock domain, which are used in this thesis to implement the PULPissimo microcontroller using Synopsys SAED 32nm technology library.

Table 3 the timing constraints of the PULPissimo implemented in the topographical flow

Clock domain name	**ref_clk**	**soc_clock**	**per_clk**	**SPIM_CLK (generated clock from per_clk)**
Clock period	56	14	28	20
Uncertainty value for setup checks	6	5	5	---
Uncertainty value for hold checks	1.15	0.88	2.22	---
Source Latency	3	--	2	---
Max. input/output delay	--	7	--	---
Min. input/output delay	--	7	--	---

3.1.2.1 Clock period:

In the implemented PULPissimo in this thesis, the clock period of the reference clock domain determines the other clock determines the clock period of the SoC and peripherals clock domains and that because the available phase locked loop (PLL) IP with the SAED 32nm technology library, which is used in this thesis, has three operating modes: normal, external feedback and bypass.

In the external feedback mode, the feedback input clock (FB_CLK) will be phase-aligned with reference input clock (REF_CLK). These aligned clocks will allow removing clock delay and skew between devices. And the frequency of the output clock signal can be the frequency of the reference clock Multiplied by a factor 4, 2 or 1. In Bypass mode, reference clock will be bypassed to out.

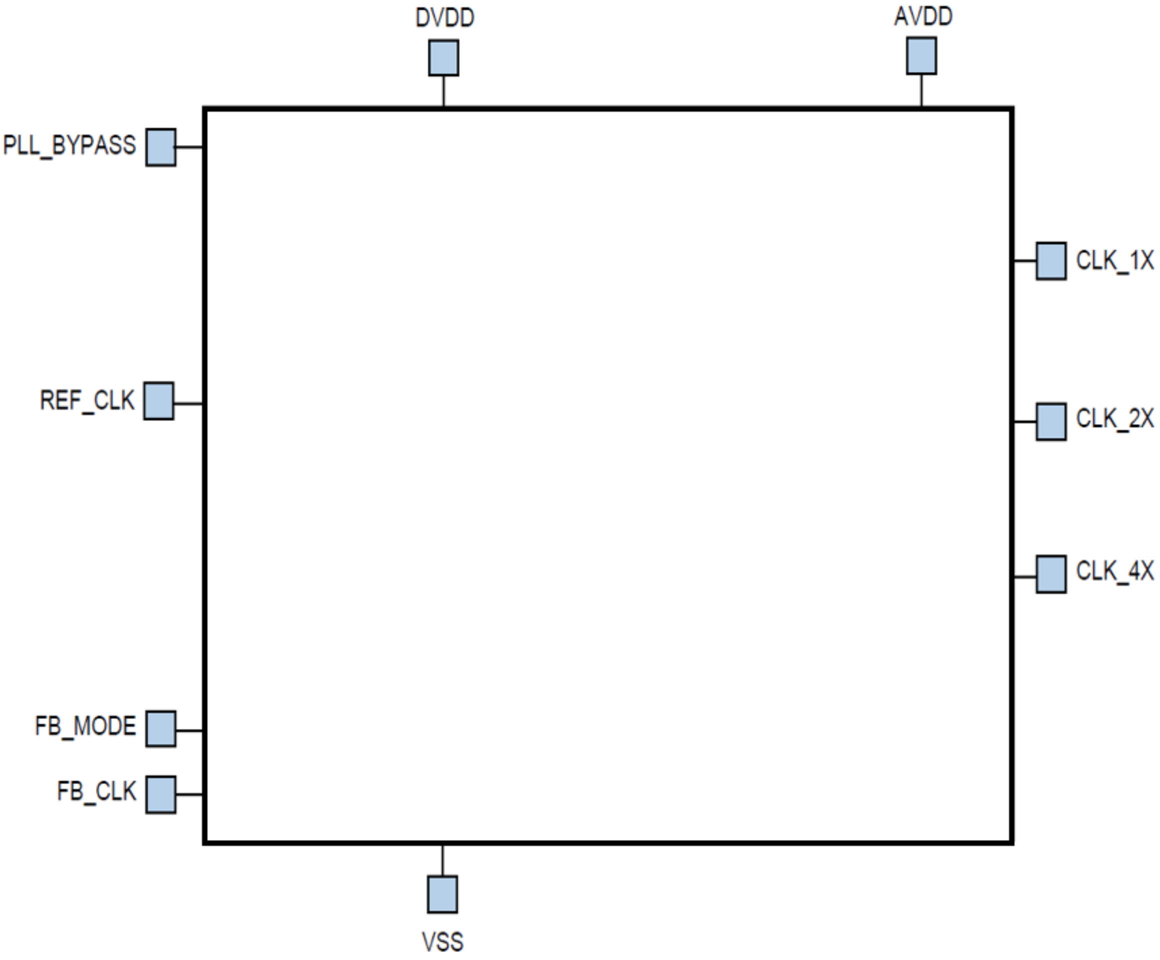


Figure 10: the SAED 32nm PLL IP as a black box

And as the SoC clock domain is the faster clock signal in the PULPissimo, we choose the PLL IP of that clock domain to work in the external feedback mode and multiplies the frequency of the reference clock by a factor 4, therefore it is equal to 14 ns which is 25% of the 56 ns reference clock.

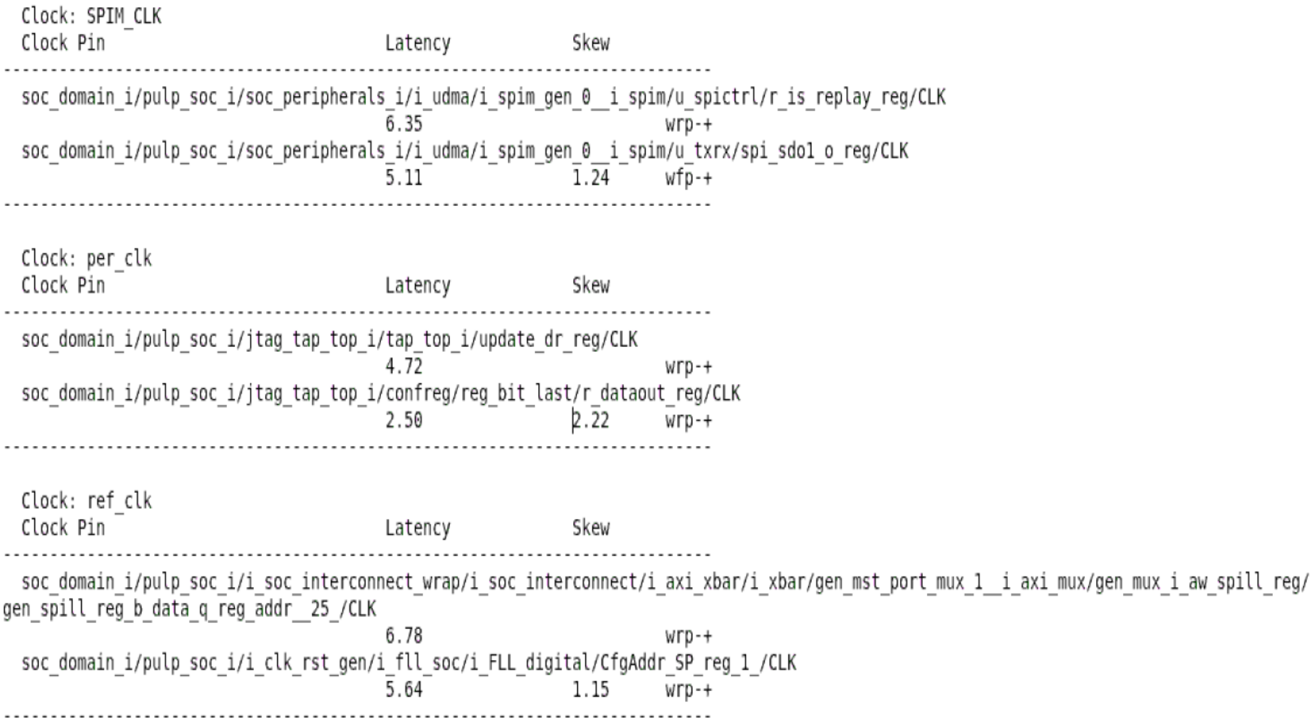
The peripheral clock signal is slower than the SoC clock signal, as it is used to trigger the PULPissimo peripherals that work on low speed signal, therefore we choose the PLL IP of that clock domain to work in the external feedback mode and multiplies the frequency of the reference clock by a factor 2, therefore it is equal to 28 ns which is 50% of the 56 ns reference clock.

3.1.2.2 Uncertainty value:

The uncertainty value of clock signal differs based on the stage at which the designer works, since before the clock tree synthesis, it is equal to the summation of the estimated skew, jitter and margin. And after the clock tree synthesis, the skew is computed, therefore the uncertainty becomes the summation of the jitter and margin. Usually, the margin in the uncertainty is increased to push the tool to the best optimization and to meet the timing constraints under conservative constraints. Therefore the designer can reduce the margin in the uncertainty as a way to solve the timing violation after the clock tree synthesis.

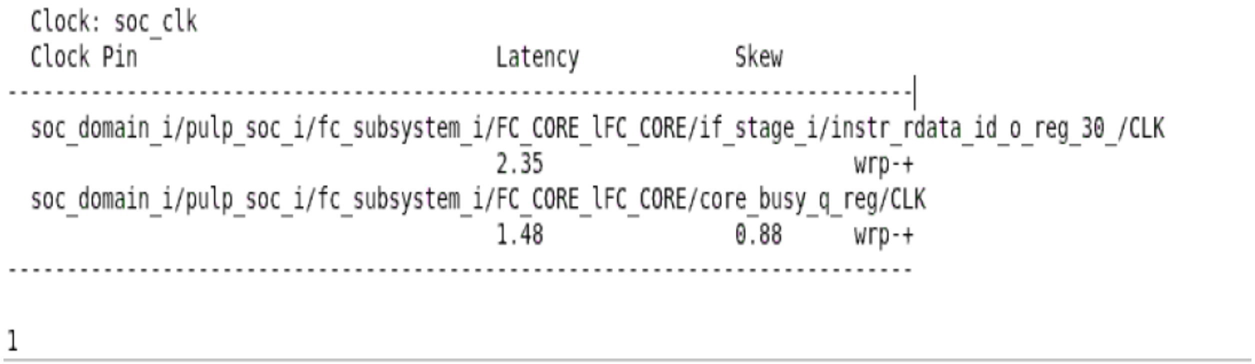
The uncertainty has two types, one is used for the setup timing checks and the other is used for the hold timing checks. And as the jitter doesn’t affect on the hold timing checks because all sequential elements in the same way, the uncertainty for the hold timing checks is more less than the uncertainty for the setup timing checks.

To choose the values of the 2 kinds of the uncertainty, we begin by choosing the uncertainty for setup timing checks of each clock domain to 10% of the clock period and the uncertainty for hold timing checks of each clock domain to 2.5% of the clock signal and these values are initial ones, and then we proceed the ASIC flow till we reach the clock tree synthesis to see the actual global skew of each clock domain, and finally we takes the uncertainty for the setup timing checks as rounding value of actual global skew multiplied by factor 3, and the uncertainty for the hold timing checks as the actual global skew. The figure below shows the global skew of each clock domain in the first pass of the hierarchal flow.



```
  Clock: SPIM_CLK
  Clock Pin                            Latency             Skew
-------------------------------------------------------------------------
  soc_domain_i/pulp_soc_i/soc_peripherals_i/i_udma/i_spim_gen_0__i_spim/u_spictrl/r_is_replay_reg/CLK
                                       6.35                         wrp-+
  soc_domain_i/pulp_soc_i/soc_peripherals_i/i_udma/i_spim_gen_0__i_spim/u_txrx/spi_sdo1_o_reg/CLK
                                       5.11                1.24     wfp-+
-------------------------------------------------------------------------

  Clock: per_clk
  Clock Pin                            Latency             Skew
-------------------------------------------------------------------------
  soc_domain_i/pulp_soc_i/jtag_tap_top_i/tap_top_i/update_dr_reg/CLK
                                       4.72                         wrp-+
  soc_domain_i/pulp_soc_i/jtag_tap_top_i/confreg/reg_bit_last/r_dataout_reg/CLK
                                       2.50                2.22     wrp-+
-------------------------------------------------------------------------

  Clock: ref_clk
  Clock Pin                            Latency             Skew
-------------------------------------------------------------------------
  soc_domain_i/pulp_soc_i/i_soc_interconnect_wrap/i_soc_interconnect/i_axi_xbar/i_xbar/gen_mst_port_mux_1__i_axi_mux/gen_mux_i_aw_spill_reg/
gen_spill_reg_b_data_q_reg_addr__25_/CLK
                                       6.78                         wrp-+
  soc_domain_i/pulp_soc_i/i_clk_rst_gen/i_fll_soc/i_FLL_digital/CfgAddr_SP_reg_1_/CLK
                                       5.64                1.15     wrp-+
-------------------------------------------------------------------------
```

Figure 10: global skew of the SPIM_CLK, per_clk and ref_clk clock domains in the initial running



```
  Clock: soc_clk
  Clock Pin                            Latency             Skew
-------------------------------------------------------------------------
  soc_domain_i/pulp_soc_i/fc_subsystem_i/FC_CORE_lFC_CORE/if_stage_i/instr_rdata_id_o_reg_30_/CLK
                                       2.35                         wrp-+
  soc_domain_i/pulp_soc_i/fc_subsystem_i/FC_CORE_lFC_CORE/core_busy_q_reg/CLK
                                       1.48                0.88     wrp-+
-------------------------------------------------------------------------

1
```

Figure 11: global skew of the soc_clk clock domain in the initial running

3.1.2.3 Clock latency:

The clock latency consists of:

1) The network latency which is the time taken by the clock signal to travel from the clock pad in the chip to a clock pin of a sequential element inside the chip.

2) The source latency which has 2 kind of definitions:

   4.3.1. Source latency of clock signal generate outside the chip and it is the time taken by the clock signal to travel from the clock pad in the chip.

   4.3.2. Source latency of clock signal generate inside the chip by a PLL IP and it is the time taken by the clock signal to travel from the clock pin of the PLL IP in the chip to the clock pin of the clock generating circuitry.

Based on the definition in 2).b, the master clock domain, that doesn’t have a generated clock domain, doesn’t have a latency. Therefore the SoC master clock domain doesn’t have a source latency value.

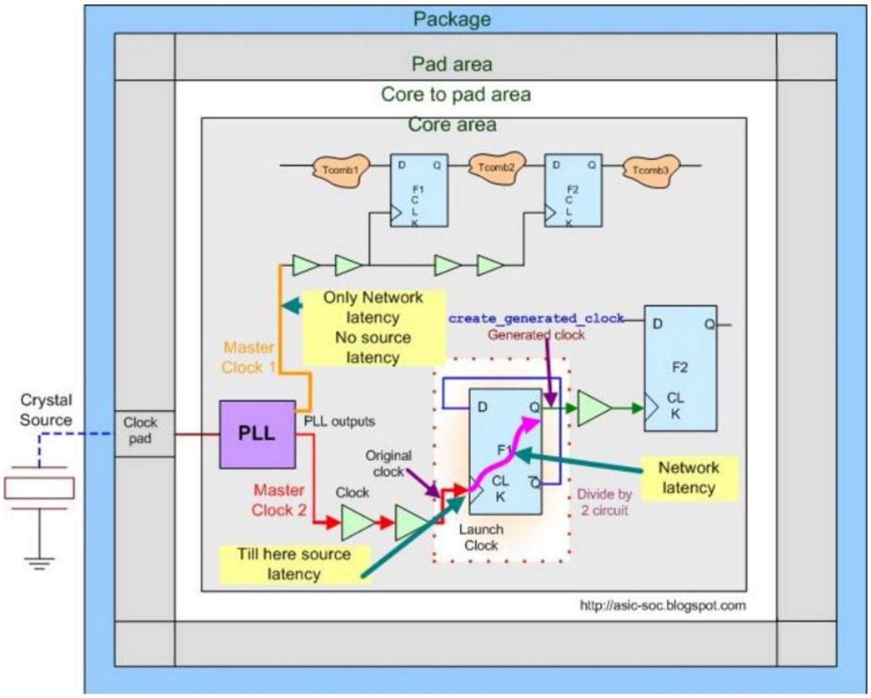


Figure 12: the difference between the two types of the source latency

The source latency of the reference clock domain is larger than the source latency of the peripheral clock domain because the path at which the reference clock signal travels, that consists of a wire on the PCB that contains the PULPissimo, is longest than the path at which the peripheral clock signal travels, that consists of a wire implemented inside the PULPissimo.

3.1.2.4 Input delay constraint:

The input delay constraint determine the division of the clock period between your chip and the other the chip that is connected to your chip as an input. All the input ports in the PULPissimo are associated with the peripheral clock domain only, therefore the peripheral clock domain is the only clock domain that has an input delay constraints and we specified 30% of the peripheral clock domain to the delay that is resulting from the logic outside the PULPissimo and 70% of the peripheral clock domain to inside the domain to the delay that is resulting from the logic inside the PULPissimo.

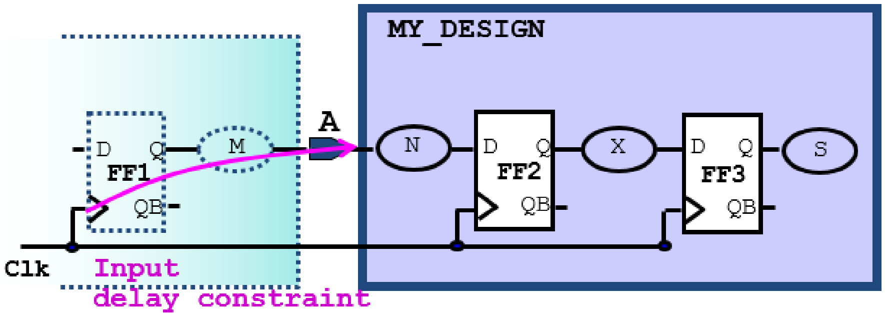


Figure 13: an example on an input delay constraint of the MY_DESIGN design

3.1.2.1 Output delay constraint:

The output delay constraint determine the division of the clock period between your chip and the other the chip that is connected to your chip as an output. All the output ports in the PULPissimo are associated with the peripheral clock domain only, therefore the peripheral clock domain is the only clock domain that has an output delay constraints and we specified 30% of the peripheral clock domain to the delay that is resulting from the logic outside the PULPissimo and

70% of the peripheral clock domain to inside the domain to the delay that is resulting from the logic inside the PULPissimo.

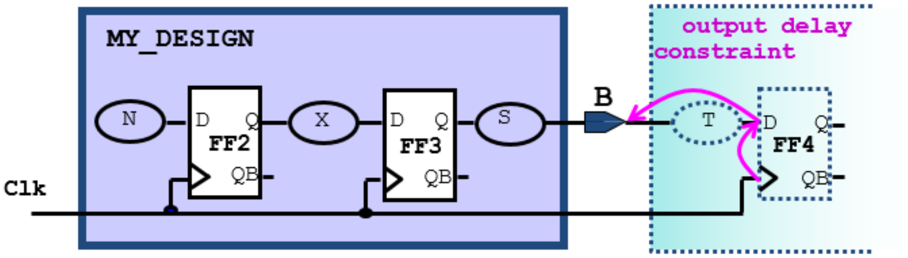


Figure 14: an example on an output delay constraint of the MY_DESIGN design

3.2 Instantiating of SAED 32nm IPs in the PULPissimo RTL:

3.2.1 Instantiating of SAED 32nm SRAM:

As the memories of the PULPissimo microcontroller are significantly large, they must be implemented using SRAM memory banks, otherwise they will degrade the area and power of the microcontroller and that will affect the main specification for which the PULPissimo is made. The SRAM that is used in this thesis is the single port SRAM SAED 32nm IP with a memory array dimensions 512 words and 32 bits per word.

The PULPissimo requires 3 different SRAM banks where:

- Four memory banks, each one of them has a memory array of 128 KB size. the four memory banks create the interleaved memory in the PULPissimo.

- Two memory banks, each one of them has a memory array of 32 KB size. the two memory banks create the private memory in the PULPissimo.

- One memory bank has a memory array of 2 KB size. the two memory banks create the private memory in the PULPissimo.

Table 4 table showing the used number of the SAED32nm SRAM IP in each memory bank

	Interleaved memory bank	**Private memory bank**	**ROM**
size	$2^{15} * 32$	$2^{15} * 32$	$2^{9} * 32$
IP instance number	64	16	4

3.2.2 Instantiating of SAED 32nm PLL IP:

We manually instantiated two instances of the SAED 32nm PLL IP that is available with the SAED 32nm technology library, where one is used for generating of the SoC clock domain from the reference clock signal and the other for generating of the peripheral clock domain from the reference clock signal.

3.2.3 Instantiating of SAED 32nm clock gating:

we manually instantiated integrated clock gating cells from the SAED 32 nm target library to avoid over constraining the clock gating design rules such clock gating setup or hold timing constraints, pre-mapped cells Design compiler (DC) takes its design rule constrains in its account.

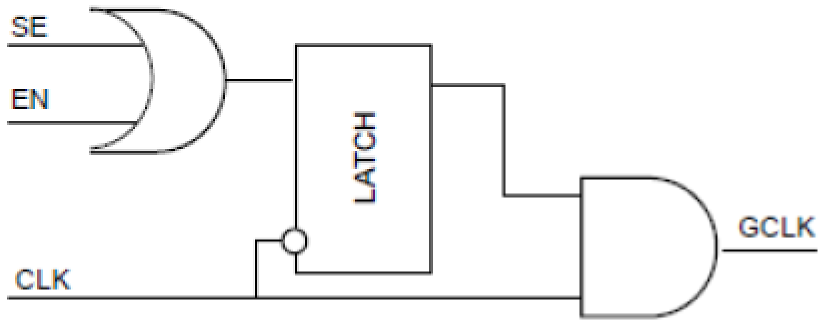


Figure 15: SAED 32 nm clock gating standard cell

3.2.4 Instantiating of SAED 32nm IO pads:

We manually instantiated 41 instances of SAED 32 nm IO pads in the pad frame domain, 4 SAED 32 nm corner pads and 3 filler pads to make the total number even, therefore makes the floor-planning more organized.

3.3 Logic synthesis:

3.3.1 Hierarchical flow:

In the hierarchical flow, the levels of the provided hierarchy in the RTL code are kept as they are during all different steps in the ASIC implementation flow. Keeping of the hierarchical levels can restrict the ASIC implementation tools, specially the design compiler tool, and prevent them from many optimizations step like boundary and partition optimization steps at which the design compiler tool can break the level of the hierarchy or the boundary of the module to reduce delay and optimize the timing paths that are extent more than one module.

We use the design compiler (DC) tool to perform the logic synthesis step. After reading the RTL code and applying the constraints, we used “ compile_ultra” command with the following options:

1) No autoungroup: to disable the default setting of the "compile_ultra" command that automatically break the levels of the hierarchy of the design to optimize the timing paths that are contained in more than one module or hierarchical level.

2) Scan: to take into the consideration the design of testability (DFT) in the beginning of the ASIC designing. Where the design of testability increase the fan-out of each net connect to a register, like the figure below shows.

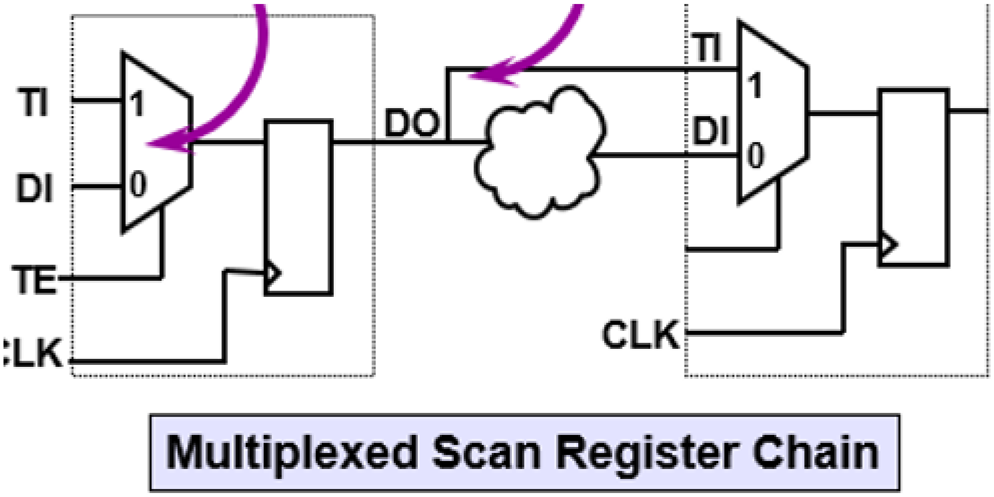


Figure 16: multiplexed scan register used for the DFT

3) Retime timing option: to be used with the critical timing paths to move their register in order to add more positive slack from the non-critical, neighboring timing paths. The –retime option is available starting with v2007.03. It performs "adaptive register retiming", which moves the logical location of registers up or down a timing path, to help improve local critical path timing without creating or worsening timing violations of surrounding paths. This will be discussed further in a later unit, in conjunction with the optimize_registers command.

4) Gate_clock: to use the instanced of the clock gating cells in the RTL code and add more.

And we done the incremental compilation step by high ultra-compilation efforts and some other techniques such as clock gating to improve (QoR) of timing, power and area.

3.3.2 Topographical flow:

In ultra-deep submicron designs, interconnect parasitics 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. In addition, the tool updates capacitances as synthesis progresses. That is, it considers the variation of net capacitances in the design by adjusting placement-derived net delays based on an updated "virtual layout" at multiple points during synthesis. This approach eliminates the need for over-constraining the design or using optimistic wire load models in synthesis. The accurate prediction of net capacitances drives Design Compiler to generate a netlist that is optimized for all design goals including area, timing, test, and power. It also results in a better starting point for physical implementation.

In order to meet the constraints, mainly the timing constraints, Synopsys offers this flow. Usually, the floor-planning step follows the synthesis step, this may lead to sub-optimal results, since the synthesis has no knowledge of the potential cell and macro placement.

And to overcome this issue we use the second pass flow, which is called topographical flow. And the main steps of the topographical flow can be summarized as the following:

1) Creating an initial netlist by synthesis.

2) Proceeding to the floor-planning step and once we reach a good floor-planning which achieve the following:
    - No congestions.
    -
    - Good and well-ordered pin and pad placement.

- Good and well-ordered macro placement.
- Good quality of results.

3) Backing and redo the logic synthesis step with the created floor-planning, but this time the synthesis step is aware of standard cells, pin, pads and macro placement which will potentially yield better results in terms of the timing (mainly the setup) , area and leakage power.

3.3.2.1 Inputs and outputs:

The second pass of the logic synthesis step in the topographical flow is done by the design compiler (DC) invoked in a topographical mode and the invocation is done by the command “dc_shell –topographical” command. And the tool takes the following files as inputs:

1) The gate-level netlist

2) The RTL code

3) The floor-planning described in the LEF file and FP floor-planning

This section describes the physical constraints imported from the DEF file with the extract_physical_constraints command, in the following subsections:

- Die Area
- Placement Area
- Macro Location and Orientation
- Hard, Soft, and Partial Placement Blockages
- Wiring Keepouts
- Placement Bounds
- Port Location
- Preroutes
- Site Array Information
- Vias
- Routing Tracks
- Keepout Margins

4) Constraints

5) Logic library

6) Physical library

The output of the design compiler is the topographical mode is a better synthesized gate-level netlist and the figure below shows these inputs and output.

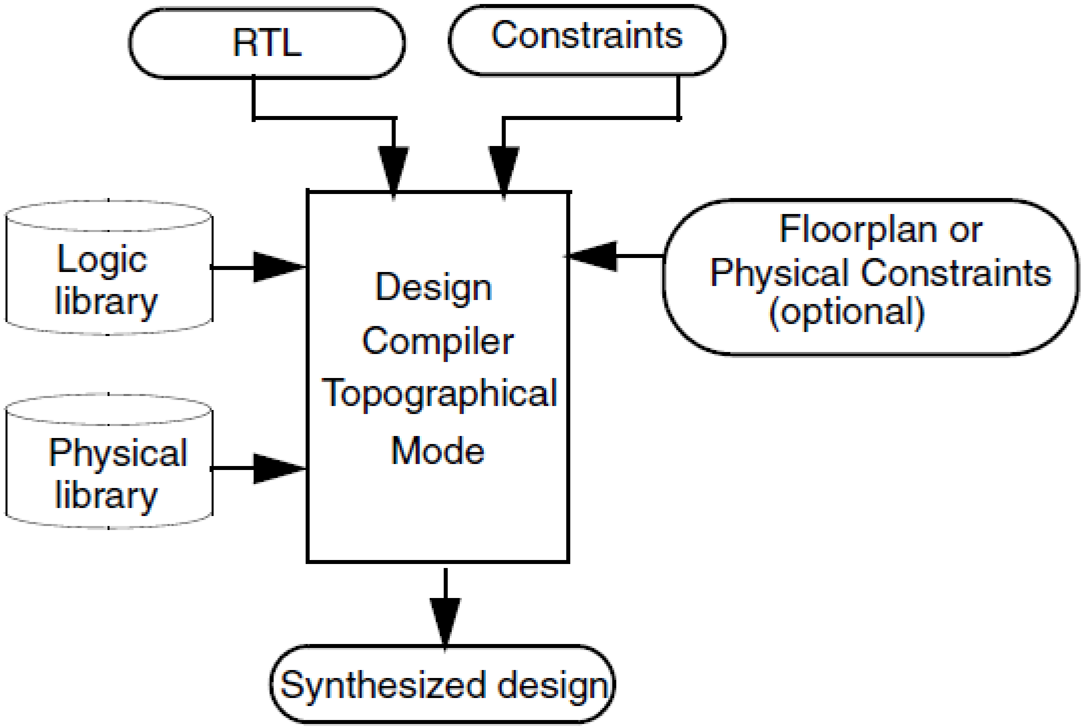

RTL
Constraints
Logic library
Physical library
Design Compiler Topographical Mode
Floorplan or Physical Constraints (optional)
Synthesized design

Figure 17: Inputs and Outputs in Design Compiler Topographical Mode

To visually inspect your extracted physical constraints, use the layout view in the Design Vision layout window. All physical constraints extracted from the DEF file are automatically added to the layout view.

3.3.2.2 Main steps:

To map a design to a new technology in the Design Compiler in the topographical mode, the following steps must be performed:

1) Add the original technology library to the link library:

```
dc_shell-topo>> lappend link_library tech_orig.db
```

2) Set up your Design Compiler environment for the new target technology.
   - Specify the logic libraries.
   - Specify the physical libraries.
3) Read in your mapped design:

```
dc_shell-topo >> read_verilog design.v
```

4) Run the “compile_ultra –incremental” command to translate the design to the new technology.

The steps of the PnR won’t be different than the ones previously described in the back-end part section.

We use the design compiler (DC) tool to perform the logic synthesis step. After reading the RTL code and applying the constraints, we used “ compile_ultra” command for the first and second pass in the topographical mode with the following options:

1) No autoungroup: to disable the default setting of the “compile_ultra” command that automatically break the levels of the hierarchy of the design to optimize the timing paths that are contained in more than one module or hierarchical level.

2) Scan: to take into the consideration the design of testability (DFT) in the beginning of the ASIC designing. Where the design of testability increase the fan-out of each net connect to a register, like the figure below shows.

3) Retime timing option: to be used with the critical timing paths to move their register in order to add more positive slack from the non-critical, neighboring timing paths. The –retime option is available starting with v2007.03. It performs “adaptive register retiming”, which moves the logical location of registers up or down a timing path, to help improve local critical path timing without creating or worsening timing violations of surrounding paths. This will be discussed further in a later unit, in conjunction with the optimize_registers command.

4) Gate_clock: to use the instanced of the clock gating cells in the RTL code and add more.

And we done the incremental compilation step by high ultra-compilation efforts and some other techniques such as clock gating to improve (QoR) of timing, power and area.

3.5.1 Flat flow:

In the flat flow, the levels of the provided hierarchy in the RTL code are broken during the first implementation step, which is the logic synthesis step, and remain broken during the rest of steps in the ASIC implementation flow.

By default, the high effort compilation removes levels of hierarchy as much as it can so we allow tool to remove levels of hierarchy which help it to optimize results.

We use the design compiler (DC) tool to perform the logic synthesis step. After reading the RTL code and applying the constraints, we used " compile_ultra" command with the following options:

5) Scan: to take into the consideration the design of testability (DFT) in the beginning of the ASIC designing. Where the design of testability increase the fan-out of each net connect to a register, like the figure below shows.

6) Retime timing option: to be used with the critical timing paths to move their register in order to add more positive slack from the non-critical, neighboring timing paths. The –retime option is available starting with v2007.03. It performs "adaptive register retiming", which moves the logical location of registers up or down a timing path, to help improve local critical path timing without creating or worsening timing violations of surrounding paths. This will be discussed further in a later unit, in conjunction with the optimize_registers command.

7) Gate_clock: to use the instanced of the clock gating cells in the RTL code and add more.

And we done the incremental compilation step by high ultra-compilation efforts and some other techniques such as clock gating to improve (QoR) of timing, power and area.

3.4 Design planning for the 3 different flows:

In this section we describe in more details the steps and design decisions that are made in the three different flows, which are flat, hierarchical and topographical, to implement the PULPissimo microcontroller.

3.4.1 Floor-planning:

In floor-planning stage, we start by defining die and core area by defining the height and width and the distance between the IO pads and the core boundaries, which define the aspect ratio and the utilization factor. Aspect ratio is the ratio between height and width, core utilization means the percentage of total cell area relative to the total area of the chip. We chose the following:

1) The core height to be 6200 um.
2) The core width to be 6200 um.
3) The distance between the IO pads boundaries and the core boundaries to be 100 um.

Which give us an aspect ratio equal to 1 a utilization factor equal to 0.401, so that we can be able to reduce the congestion issue. And he distance between the IO pads and the core boundaries are 100 um in all sides of the chip. Then we define placement constraints on the following:

1) Pins
2) IO pads
3) Macros including, the SRAM macro cells and the 2 PLLs

We define the metal layers that will be used for signal routing stage and the metal layers that will be used for routing of the power network, this is important at floor-planning stage so that PnR tool can perform global routing to estimate routing regions and potential congestion properly. The technology file provided with the PDK contains 9 routable layers, minimum routing layer is chosen to be metal layer 1 and maximum routing layer is metal layer 8. These layers will be used as the following:

- From Metal layer 1 to metal layer 7 will be used for signals routing.

- Metal layer 9 and Metal layer 8 for horizontal and vertical power straps and rings.
- From Metal layer 3 to metal layer 5 will be used for clock signal routing as higher metal layers have less resistance than lower layers which means lower interconnections propagation delay with non-default routing rules, which are doubling of minimum width and doubling of minimum spacing.

3.4.2 Virtual flat placement:

Virtual placement is then performed to allocate standard cell placement locations inside the chip core as previously explained and to place the macros, IO pads and the pins based on the previously defined constraints. The figure below shows the results of the floor-planning floorplan of the chip.

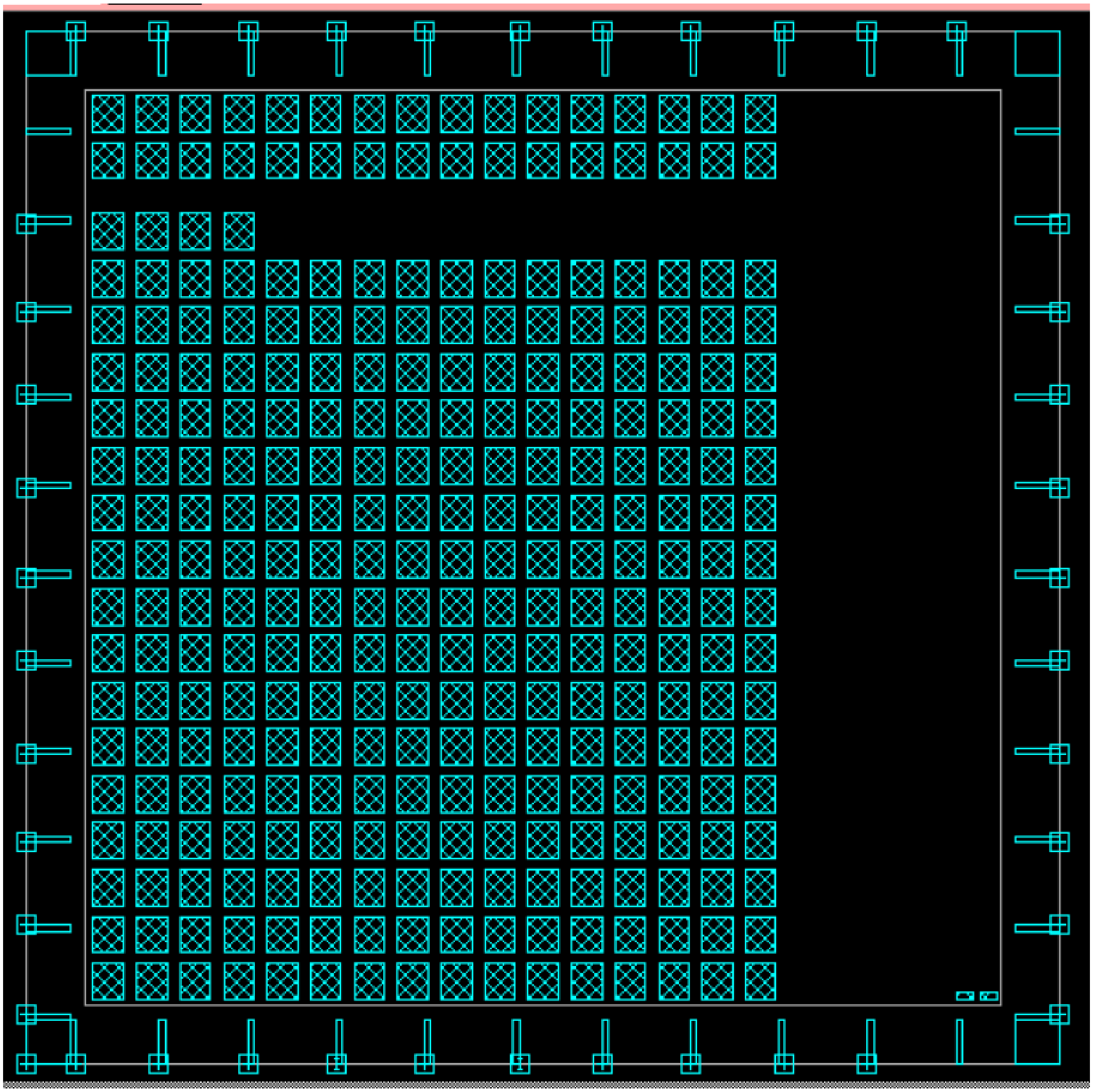


Figure 18: the floor-planning of the PULPissimo.

Where we choose to place macro cells as the following:

1) the private memory, which consists of 2 memory banks each one of them consists of 16 SRAM macro cell, in the upper right corner of the chip with a X offset equal to 90 um and a Y offset equal to 70 um

2) The ROM, which consists of the 4 SRAM macro cells, below the private memory and with the same X and Y offset as the private memory.
3) The interleaved memory, which consists of 4 memory banks each one of them consists of 64 SRAM macro cells, below the ROM and with the same X and Y offset as the private memory. Where each four rows represent a one memory bank of the interleaved memory.
4) The 2 PLLs are collecting together into one macro array and a 50 um separated distance between is chosen. And let the IC compiler tool to choose a suitable location to them in order to enhance the quality of results.
5) We set the keep-out margin, where the standard cells are prevent to be placed in the area of the keep-out margin, of all macro cells to 10 um from each sides of them. And that to reduce the congestion and provide more routing resources for the interconnects of the macro cell pins.

The reason behind collecting the SRAM macro cells of the private memory, ROM and the interleaved memory together in one place and reduce the separation distance between them as much as possible is reduce the interconnects be as much as possible, therefore better quality of results and because the number of interconnects among the SRAM macro cells and themselves is much larger than the number of the interconnects among the SRAM macro cells and other standard cells, therefore that placement may enhance the congestion as well. Figures below show the location of each macro.

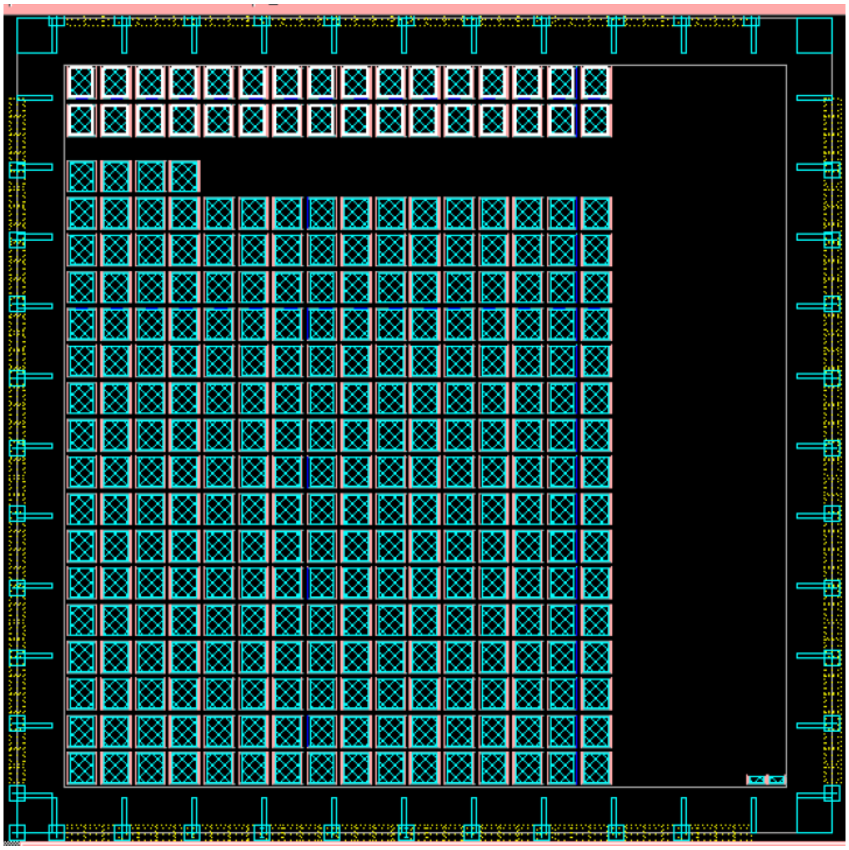


Figure 19: the highlighted 2 memory banks of the private memory

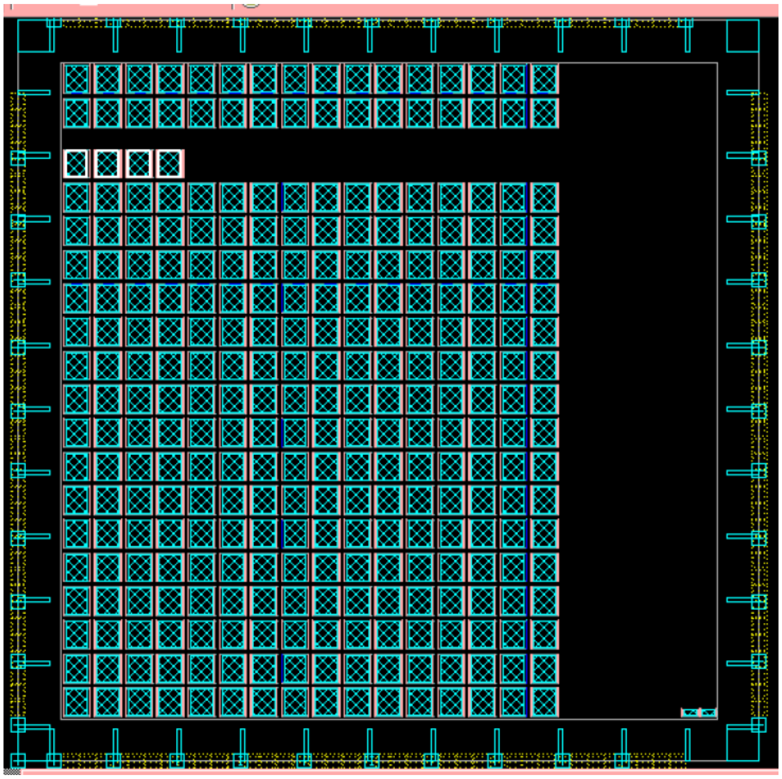


Figure 20: the highlighted ROM

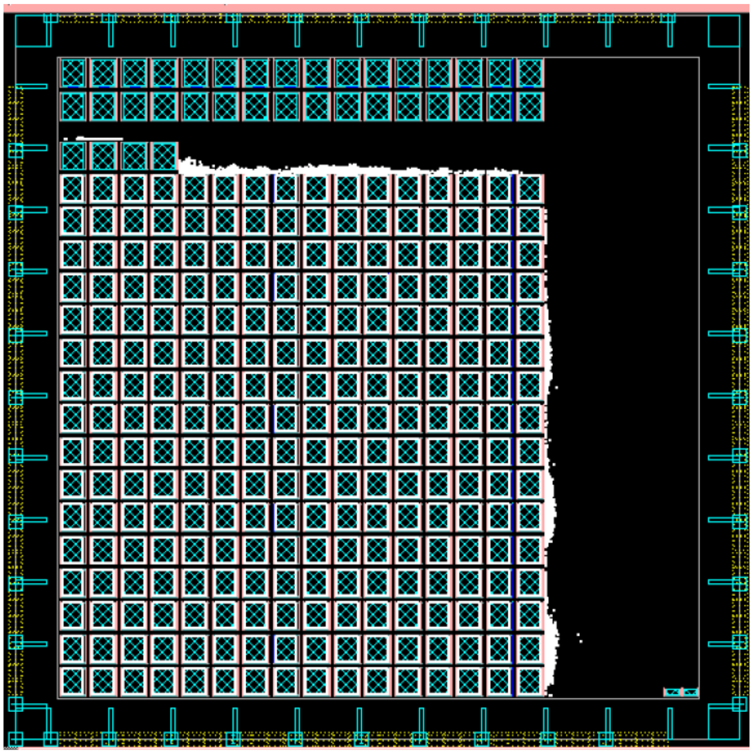


Figure 21: the highlighted 4 memory banks of the interleaved memory

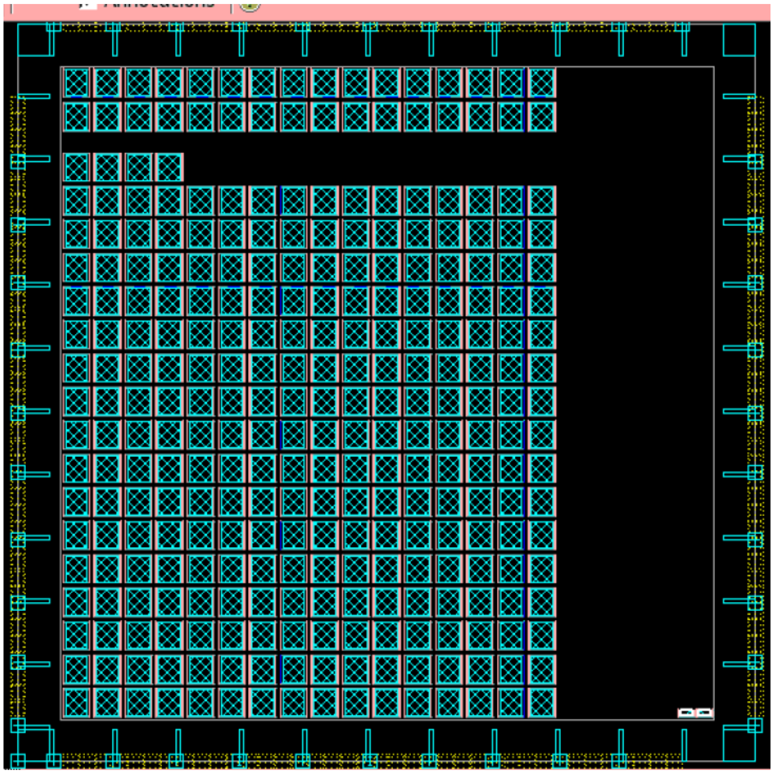
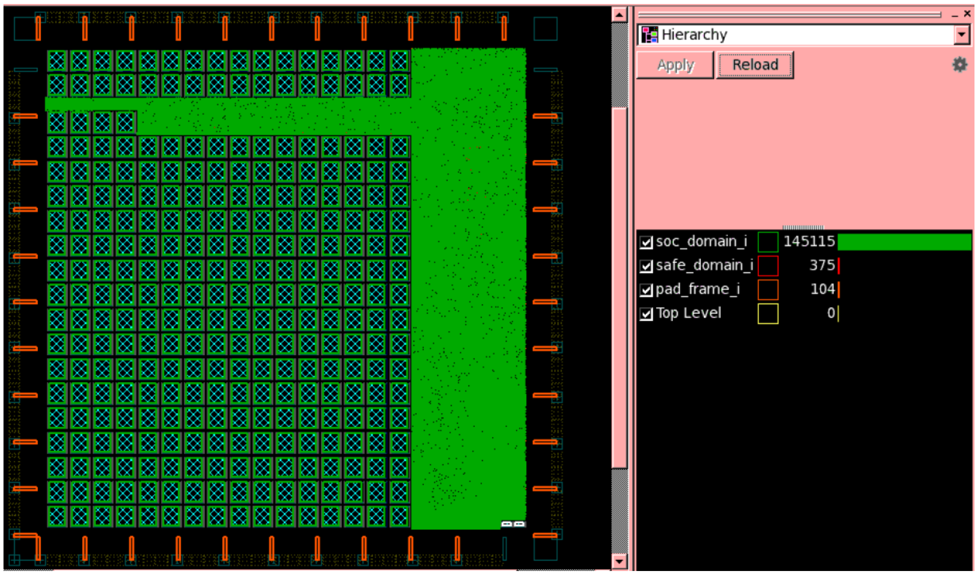


Figure 22: the highlighted 2 PLLs

The figure below shows the hierarchical virtual placement, where each domain of the 3 domains of the PULPissimo, SoC, pad frame and peripheral domains, is placed on one area as much as possible.



☑ soc_domain_i	□	145115
☑ safe_domain_i	□	375
☑ pad_frame_i	□	104
☑ Top Level	□	0

Figure 23: the hierarchical map shows the 3 main domain in the PULPissimo.

3.4.3 Reducing the congestion:

Reducing of the congestion in the design planning stage is very important to ensure that the core area contains an enough amount of routing resources, where the routing resources are the metal tracks in the metal layers, otherwise the IC compiler won't be able to complete the routing properly in the proceeding stages.

We use the trial and error method to determine the optimum offset in both X and Y directions between the SRAM macro cells in order to enhance the congestion as much as possible and keep the SRAM macro cells well-ordered, the figure below (figure 24) shows some of the trial and their results where in each trial we was increasing the offset in both X and Y directions.

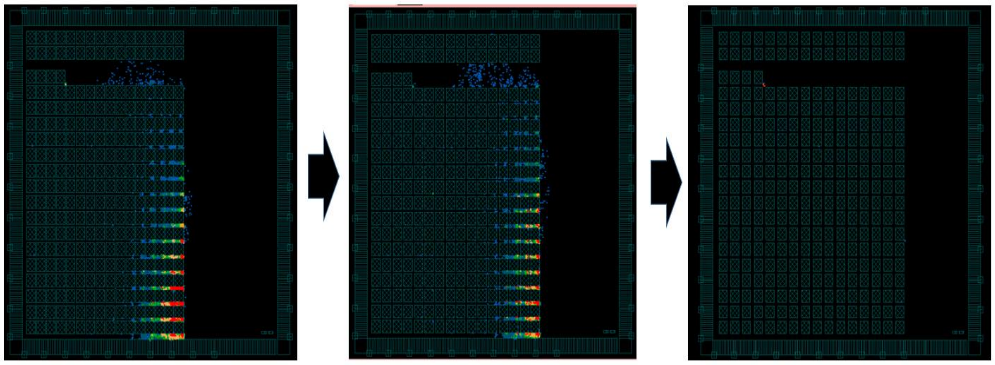


Figure 24: shows some of the trial and their results

The figures below shows the congestion map of our design, where almost all GRCs have available tracks more than the demand tracks.

3.4.4 Power tree synthesizing:

The power network consists of rings, straps and supply rails. We chose to implement the power network on the chip as the following:

1) The power and ground rings are been chosen to be implemented in metal layer 8 and 9 where the horizontal ring segments in the metal layer 9 and the vertical ring segments in the metal layer 8, where:
    - Their width is set to the maximum value which is 5um to increase the cross section at which the current flow, therefore their resistance will be decreased their resistance, therefore the IR drop on them will be decreased.
    - The separation distance between the power and ground rings is set to the minimum spacing distance in the metal layer 8 and 9, which is 0.5 um, to increase the number of the power and ground rings as much as possible, therefore reduce the IR drop in them.

The figure below shows the ring structure.

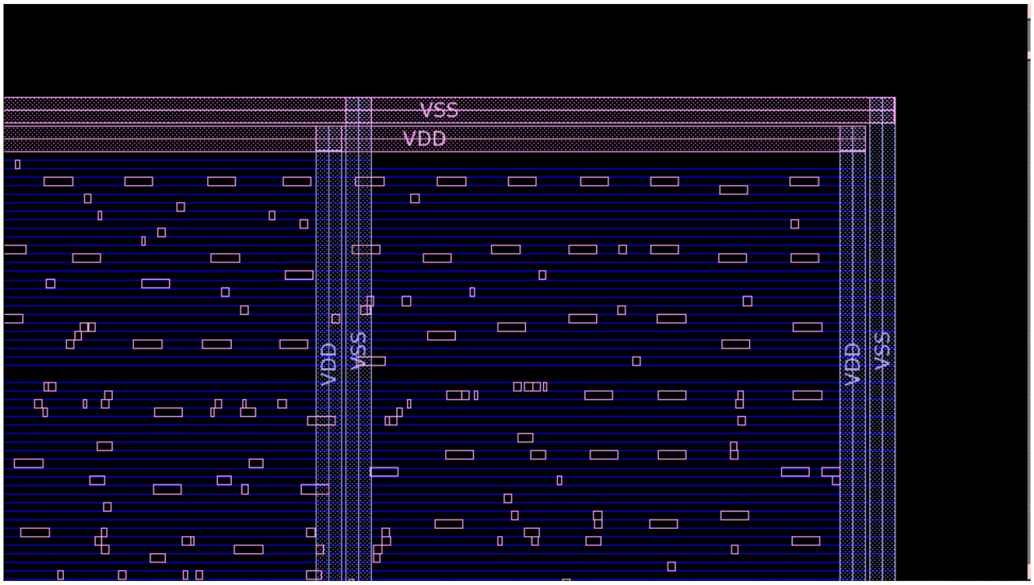


Figure 25: rings as a part of power network of the PULPissimo implemented in flat, hierarchical and topographical flows

2) The power and ground mesh, which consists of horizontal straps implemented in metal layer 8 and vertical straps implemented in metal layer 9, where:
    - Their width is set to the maximum value which is 5um to increase the cross section at which the current flow, therefore their resistance will be decreased their resistance, therefore the IR drop on them will be decreased.
    - The separation distance between in each group of power and ground straps is set to the minimum spacing distance in the metal layer 8 and 9, which is 0.5 um, to increase the number of the power and ground straps as much as possible, therefore reduce the IR drop in them.
    - The separation distance between each group of power and ground straps and the other is set to a distance which is 100 um to increase the number of the power and ground straps as much as possible, therefore reduce the IR drop in them.

The figure below shows the structure of the horizontal and vertical straps.

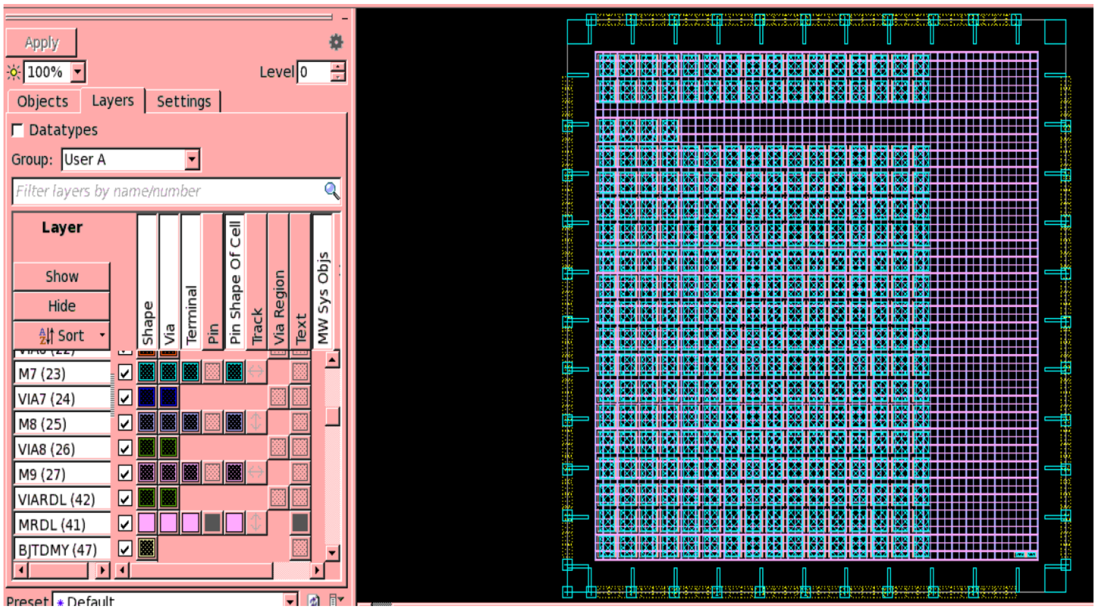


Figure 26: power network straps of the PULPissimo implemented in flat, hierarchical and topographical flows

3) We chose to build power and ground rings around each macro cell to make power and ground pins of the macro cells connected easily to power network and reduce the resistance of the interconnects that connects the power and ground pins of the macro cells to the power mesh in the metal layer 9 and 8. The setting of them are chosen to be as the following:
   - The horizontal segments are implemented in metal layer 5, as the pins of the SRAM macro cells are implemented in the same metal layer, and the vertical segments are implemented in metal layer 6.
   - Their width is set to the maximum value which is 5um.
   - The separation distance between the power and ground rings is set to 0.5 um.
   - The offset between the top side of the rings and the top side of SRAM macro cell is set to 2 um to reduce the length of the interconnects that connects the power and ground pins of the macro cells to the rings that are around the macro cells, therefore reduces their resistance and enhances the IR drop on them.
   - The offset between the all other sides of the rings and all other sides of SRAM macro cell is set to 15 um to increase the routing resources around all other pins of the SRAM macro cells, therefore reduce the congestion.

The figures below show the power and ground rings around SRAM and the PLL macro cells.

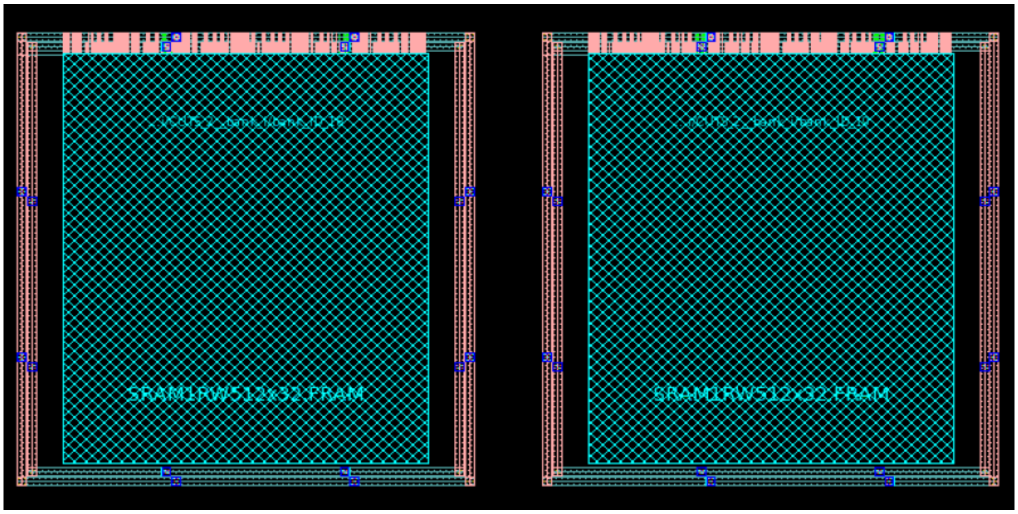


Figure 27: power and ground rings around SRAM macro cells in the PULPissimo implemented in flat, hierarchical and topographical flows

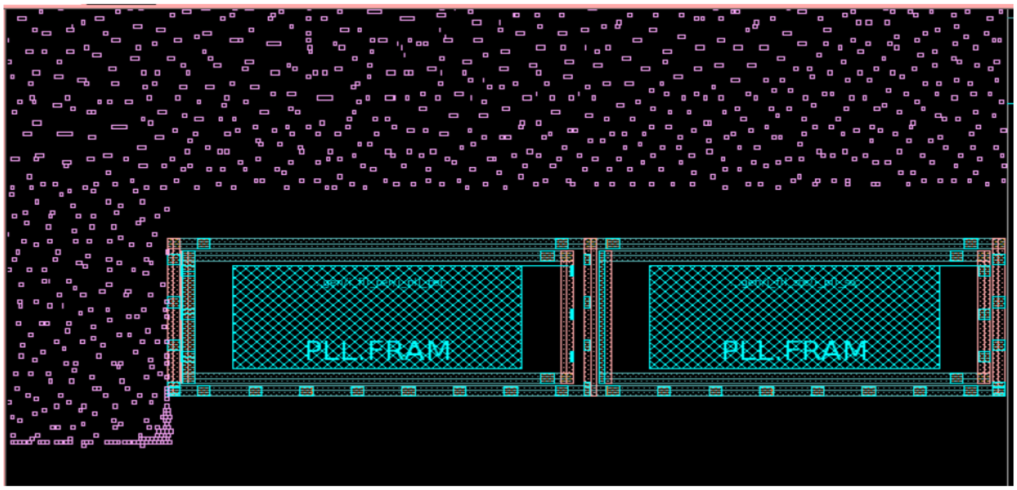


Figure 28: power and ground rings around PLL macro cells in the PULPissimo implemented in flat, hierarchical and topographical flows

4) The power supply are implemented in the metal layer 1, and we choose the disable the routing of the supply rails over the macro and fill each rows that with at least one placed standard cells.

3.4.5 Reducing delay:

The reducing delay is important before the extracting, but this is needed only when there are a large setup timing violations, where the worst negative slack (WNS) more than 20% of the clock period.

Fortunately, due to well-defined virtual placement, the WNS in the flat, hierarchical and topographical flows is in the SoC clock domain and it is less than or equal to 0.14% of the SoC clock period in. Therefore there is no need to reduce the delay.

3.4.6 Extracting:

The RC extracting is done to easily translate and pass the RC models of the power network and the estimated interconnects to other tools, like the design compiler tool in the topographical flow.

3.5 Placement of for the 3 different flows:

3.5.1 Setting of the non-default routing rules:

The setting of the non-default routing (NDR) rules of the clock signals should be done before the placement stage to enable the IC compiler tool to take them into the consideration, especially during the RC extracting of the clock nets that happens in the placement engine in the tool.

We choose the NDR rules of the clock nets, that will be implemented in the metal layer 3, 4 and 5, to be as the following:

1) The minimum spacing is equal to the minimum spacing of each metal layer in the used SAED 32nm technology library, to reduce the inter-wire capacitance that arise between the wires, therefore reduces the cross-talk between the wires and prevents swings in them. The non-default minimum spacing in the metal layer 3 is set to 0.112 and in the metal layer 4 and 5 is set to 0.224.
2) The minimum width is equal to the minimum width of each metal layer in the used SAED 32nm technology library, to increase the cross-section at which the current flows, therefore reduces the resistance of the clock nets.

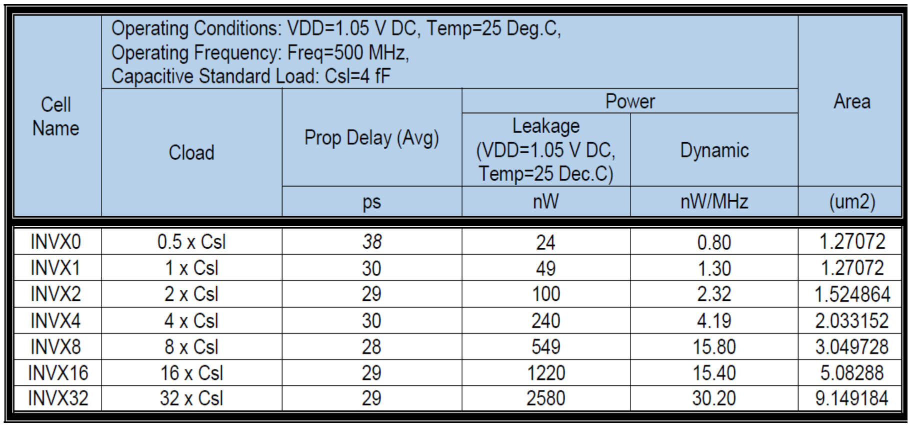
The non-default minimum width in the metal layer 3 is set to 0.112 and in the metal layer 4 and 5 is set to 0.224.

3.5.2 Setting of the options of the high fan-out synthesis:

The setting the options of the high fan-out synthesis (HFS) is important before the execute the main command of the placement stage, which is the “place_opt” command, because the main command won’t run the HFS engine unless these options are set. The HFS reduces the high fan-out number of the high fan-out nets by building a buffer or inverter tree at which the large net is divided into smaller nets with a much smaller fan-out number.

We chose the options of the high fan-out synthesis as the following:

1) The reference standard cells by which the buffer or inverter tree is built to be all inverters that are available with the used technology library which is SAED 32nm. We chose only the inverters to build the trees because the delay of the inverter in less than the buffers, but that may increase the congestion because the tool add more inverters and the available versions are described in the figure below.



Cell Name	Operating Conditions: VDD=1.05 V DC, Temp=25 Deg.C, Operating Frequency: Freq=500 MHz, Capacitive Standard Load: Csl=4 fF				Area
	Cload	Prop Delay (Avg)	Power		
			Leakage (VDD=1.05 V DC, Temp=25 Dec.C)	Dynamic	
		ps	nW	nW/MHz	(um2)
INVX0	0.5 x Csl	38	24	0.80	1.27072
INVX1	1 x Csl	30	49	1.30	1.27072
INVX2	2 x Csl	29	100	2.32	1.524864
INVX4	4 x Csl	30	240	4.19	2.033152
INVX8	8 x Csl	28	549	15.80	3.049728
INVX16	16 x Csl	29	1220	15.40	5.08288
INVX32	32 x Csl	29	2580	30.20	9.149184

Figure 29: inverters specifications that used as reference of HFS engine

2) Enable port punching
3) Enable boundary phase
4) Enable the HFS to work on the constant nets

3.5.3 Placement:

The main placement engine of the IC compiler tool is invoked by the “place_opt” command and we choose the following as options for this command:

1) Power.
2) Congestion.
3) Setting the effort to high to get best result, but the high effort will increase the CPU time which is acceptable to us.
4) Area recovery, where the main placement engine tries to re-size the standard cells to enhance the area.
5) Timing, where the main placement engine tries to fix the setup timing violations only and doesn’t fix the hold timing violations.

And before the executing of the “place_opt” command we reduce the keep-out margin to be 8 um for all the macro cells.

3.5.4 Reporting:

As the more important things to evaluate the placement stage are the congestion may the setup timing slack, we reported the congestion by executing the global route engine in the IC compiler by executing the “route_global” command and reported the quality of the results (QoRs) by executing the “report_qor” command and that to determine the options of the placement optimization engine that will be illustrated in the following the section.

3.5.5 Optimization of the placement:

The optimization of the placement in the IC compiler tool is executed by the “psynopt” command and the options of this command are chosen based on the results of the previous reporting step where:

1) if the design is congested, the designer should choose the “-congestion” option and if the design is still congested, the designer may delete the old results of the “place_opt” command, execute the “set placer_enable_enhanced_router TRUE” and then execute a new

"place_opt" command, that will invoke the real global routing engine during executing the new round of the "place_opt" command instead of the global routing estimator engine that is used the previous the "place_opt" therefore the results will be much better, but the CPU time will increase.

2) If the WNS of the timing setup checks isn't good or the DRC violations are a lot, the designer should execute the "psynopt –only_design_rule" command.

3.6 Clock tree synthesis:

3.6.1 Setting the clock tree options:

Before the beginning of the clock tree synthesis, the designer should set options of the clock tree synthesis engine and these options are divided into the following:

1) Targets, which are just to the tools which means that if the results of the clock tree synthesis doesn't meet these target, no violations are reports, but it is always nice to meet these targets. And the target that must be specified for each clock domain are:
    - Target skew which is the maximum global skew in the clock domain.
    - Target early delay which is the minimum insertion delay in the clock domain.
2) Constraints, if the clock tree engine couldn't meet these specified constraints, violations are reported and the CTS will fails. And they are the following
    - Maximum capacitance
    - Maximum fan-out
    - Maximum transition
3) References, which are the standard cells that the clock tree synthesis engine will use to build the clock tree of each clock domain.
4) The non-default routing rules by which the clock trees are routed.
5) The metal layer at which the the clock trees are routed.

We begin first running with an estimated skew in the uncertainty for the setup timing checks (which is equal to summation of estimated maximum global skew, jitter and margin) and the uncertainty of the hold timing checks (which is equal to the estimated maximum global skew only, as the the jitter doesn't affect on the hold timing checks) and proceeding the ASIC flow steps till we reach the clock tree synthesis step and synthesize all the clock domains in the PULPissimo and then report the maximum global skew of each clock domain. Then we go back to the beginning of the ASIC flow and set the uncertainty of both setup and hold timing checks as the following:

1) The uncertainty of the setup timing checks of each clock domain in the PULPissimo is set to be equal the maximum global skew that is resulted from the first running multiplied by factor 3.
2) The uncertainty of the hold timing checks of each clock domain in the PULPissimo is set to be equal the maximum global skew that is resulted from the first running.

To determine the clock tree targets and the constraints, we use the resulted maximum global skew that is resulted from the first running as initial point to begin with in the synthesizing clock domains in the PULPissimo. The following table summarized the clock tree synthesis options for each clock domain in the PULPissimo implemented in the three used ASIC approaches, hierarchical, topographical and flat.

Table 5 clock tree synthesis options for each clock domain in the PULPissimo implemented in hierarchical approach.

CTS option name	ref_clk	soc_clk	Per_clk	SPIM_CLK
Target skew (in ns)	1	0.75	2	1
Target early delay (in ns)	5.64	1.4	2.5	5.64
Maximum capacitance (in ff)	300	300	300	300
Maximum fan-out	10	10	10	10
Maximum transition (in ns)	0.2	0.2	0.2	0.2
Gate relocation optimization	TRUE	TRUE	TRUE	TRUE
Gate resizing optimization	TRUE	TRUE	TRUE	TRUE
Buffer relocation optimization	TRUE	TRUE	TRUE	TRUE
Buffer resizing optimization	TRUE	TRUE	TRUE	TRUE
Use default routing for sinking	TRUE	TRUE	TRUE	TRUE

Table 6 clock tree synthesis options for each clock domain in the PULPissimo implemented in topographical approach.

CTS option name	ref_clk	soc_clk	Per_clk	SPIM_CLK
Target skew (in ns)	1	0.75	2	1
Target early delay (in ns)	5.64	1.4	2.5	5.64
Maximum capacitance (in ff)	300	300	300	300
Maximum fan-out	10	10	10	10
Maximum transition (in ns)	0.2	0.2	0.2	0.2
Gate relocation optimization	TRUE	TRUE	TRUE	TRUE
Gate resizing optimization	TRUE	TRUE	TRUE	TRUE
Buffer relocation optimization	TRUE	TRUE	TRUE	TRUE
Buffer resizing optimization	TRUE	TRUE	TRUE	TRUE
Use default routing for sinking	TRUE	TRUE	TRUE	TRUE

Table 7 clock tree synthesis options for each clock domain in the PULPissimo implemented in topographical approach.

CTS option name	ref_clk	soc_clk	Per_clk	SPIM_CLK
Target skew (in ns)	1	0.75	2	1
Target early delay (in ns)	5.64	1.4	2.5	5.64
Maximum capacitance (in ff)	300	300	300	300
Maximum fan-out	10	10	10	10
Maximum transition (in ns)	0.2	0.2	0.2	0.2
Gate relocation optimization	TRUE	TRUE	TRUE	TRUE
Gate resizing optimization	TRUE	TRUE	TRUE	TRUE
Buffer relocation optimization	TRUE	TRUE	TRUE	TRUE

Buffer resizing optimization	TRUE	TRUE	TRUE	TRUE
Use default routing for sinking	TRUE	TRUE	TRUE	TRUE

3.6.2 Clock tree synthesizing:

After setting of the clock tree options, we follow a certain approach to efficiently handling multi-clock domain design:

1) Synthesizing clock domain without either fixing hold violations or routing it.
2) Reporting the actual global skew of the synthesized clock domain.
3) Removing the summation of the actual skew of the synthesized clock domain skew and a margin from the uncertainty of the setup timing checks.
4) Setting the hold timing checks of the synthesized clock domain to zero.
5) Fixing the hold timing violations of the synthesized clock domain.
6) Reporting the quality of results of the synthesized clock domain.
7) Perform more hold-fixing if the quality of results isn’t acceptable.
8) If the quality of results isn’t acceptable, reducing the maximum capacitance and maximum fan-out number of the synthesized clock domain, and then repeats all above steps.

We use the following command each clock domain in the PULPissimo to perform the previously mentioned approach:

1) The “clock_opt –only_cts –no_clock_route –clock_trees $clock_domain_name” command synthesizes the given clock domain without either fixing hold timing violations or routing it.
2) The “report_clock_trees –summary –clock_trees $clock_domain_name” command is used to report the actual global skew of the synthesized clock domain.
3) The “set_fix_hold $clock_domain_name” and “set_fix_hold_options –prioritize_tns –effort high” commands , and then executing the “clock_opt

–psyn_only –no_clock_route” command. By these 3 commands, we begin solve the hold timing violations of specified clock domain.

4) After synthesizing and fixing the hold violations of each clock domain, we begin to routing all clock signal simultaneously, by executing the “route_zrt_group -all_clock_nets -reuse_existing_global_route true” command.

3.7 Chip finishing:

Chip finishing consists of four basic steps:

1) Metal layers spreading and widening
2) Standard cell filling
3) Redundant vias insertion
4) Metal filling

3.8.1 Pad filling:

Pad filling is performed in the empty locations between the IO pads to make the chip uniform in density and to route the power and ground tracks properly.

3.8.2 Standard cell filling:

Standard cell filling is performed in the empty locations in the standard cell rows to make the chip uniform in density and to improve the yield of the chip. Some locations may be still empty because if filling occurred at such locations, it would lead to DRC violations.

3.8.3 Metal filling:

Metal filling is performed to protect metal interconnections in low metal density regions from over-etching during fabrication process. Timing driven metal filling specifically is performed to preserve timing on critical nets, metal fill near critical nets on the same layer, upper layer, and lower layer are removed or trimmed.

3.8.4 Verifying and reporting:

Verifying for design rules violations is done by executing the “verify_route” command which reports a summary for the DRC violations and shorts. And verifying for shorts and opens is done by executing the “verify_lvs” command which reports the shorts and opens in the design by performing layout versus schematic checks. The reporting is done by executing the “report_qor” command to report setup and hold worst negative slack and to setup and hold total negative slack, area of standard cells, IO pads and macros and the CPU time.

3.8.5 Outputs:

The outputs are GDS file, which contains the layout of the chip, gate-level netlist written in Verilog, constraints written in SDC format and the standard parasitic extraction format (SPEF) files, which allows the representation of parasitic information of a design(R, L, and C) in an ASCII, to be used in the static timing analysis in the Synopsys Prime-Time tool.

3.8 Static timing analysis:

After performing a clean place-and-route phase, the next necessary step is to make timing verification on a high precision static timing analysis tool as the timing aspect is a very sensitive aspect to take care of. Our STA tool that we have worked on is Synopsys Prime-Time, and we are going to discuss our work on it.

The target of using this tool in this stage is to verify timing on a spectrum of operating conditions that the design is expected to operate at. These operating conditions have three main dimensions which are temperature, operating voltage and manufacturing process. Another dimension that can be taken into account is MOSFET threshold voltage. We are going to spread them below.

In manufacturing process analysis, we can classify it to three elements: slow-slow, typical-typical, and fast-fast process. As illustrated from the naming, the slow-slow process has the highest standard cell delay among the others, and then the fast-fast process has the lowest one among them. With respect to the ambient temperature, we can say that the highest temperature-based cells have the highest delay, and the lowest temperature-based cells have the lowest delay.

Another affecting aspect is the operating voltage, it also has a significant impact on the propagation delay, since the highest voltage-based cells have the lowest delay, and the lowest voltage can contribute with a higher delay.

The last affecting factor is the MOSFET threshold voltage. There are three main types of threshold voltage based on MOS devices, and they are low voltage threshold (LVT), regular voltage threshold (RVT), high voltage threshold

(HVT). This factor is also significantly affecting the delay of the standard cells. The low threshold-based cells have the lowest delay, and the highest one has the highest propagation delay. In our case, we have only regular voltage threshold-based libraries. This type of libraries has a variety of the previously mentioned operating conditions.

Therefore, we can conclude that the best-case library has the lowest temperature, the highest voltage, and the fastest process, and the worst-case library has the inverse of these condition.

After recognizing the expected cases, we can go through our work, and handle all of these cases in order to meet the setup and hold timing requirements. One the significant problems that we have faced is that we have performed the synthesis process in the worst-case library. That implies good setup timing results, but deeply worst hold-timing results, since the timing paths is welloptimized for the worst case and have minimum possible delay so that these paths cannot hold enough time for data to be well-stabilized.

In order to solve this problem, we have to insert plenty of buffers to boost the delay, and that can be achieve in each case separately.

Recalling to the inputs of the tool, the tool needs to read the Verilog netlist and the operating library that we are going to analyze its case, in addition to the constraints file, and the parasitics file so that we can define the actual delays of the physical nets.

Till now, everything appears to be great, but a problem arises when setting up the environment to work, and this problem is that the tool calculates the propagation delay of the nets based on a non-real resistance and capacitance values, thus that can lead to non-real delay values. As discussed in the library section in the appendix, the standard cell library has a table to calculate the delay based on it. When the parasitic values exceed these values, the STA tool begins to make extrapolations based on the last parasitic values that defined in the table, and it takes this maximum value of defined delay and adds an extra 10% of delay. This can result in non-accurate delay calculations at all. That conclusion illustrated that the parasitic values, one way or another, is wrongly written in the file.

Therefore, we have resorted to another manipulating solution, and that solution was to work with DDC-based files. These files contain the design netlist information in addition to the calculated values of the parasitics, and also the design constraints are built-in in this file. To ensure that these values are valid, we have made a small comparison between the end-point path slack histogram in both layout tool and STA tool, and the result was approximately the same.

After finishing the setup successfully, we have stepped forward to analyze the design against the different cases. In most cases, the design had clean setup timing, and the problem was at the hold time. There are two was ways to solve such problems. The first one is to fix them in the layout tool and forward the design to layout tool to verify the results. The other one 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.

After experiencing the two method, we have decided to build our final results based on the second method. The PrimeTime STA tool offers a built-in synthesis engine to fix the setup and hold timing violations. We can take case study example, and we can consider all the other cases were treated by the same analogy.

For example, we have a certain library causes a certain hold time violation to the design. Therefore, we basically have inserted buffers with different sizes to recover the violation, but unfortunately, the STA cannot fix the setup and hold time violations simultaneously. Thus, a setup time violation can easily arise. After few iterations for solving the setup and hold violations, we can finally forward the changes file, which can be named “ECO netlist”, to the layout tool. Then, the layout have to do some modifications on the design such as performing eco routing to resolve the expected violations that can be resulted from the “ECO netlist”.

4) Results:

The following sections represent and discuss the results of implementing the PULPissimo microcontroller in the flat, hierarchial and topographical ASIC approaches using SAED 32nm technology library. The results are represented and discussed in the following order:

1) Logic synthesis
2) Design planning
3) Placement
4) Clock tree synthesis
5) Routing
6) Chip finishing
7) Static timing analysis

4.1 Logic synthesis results:

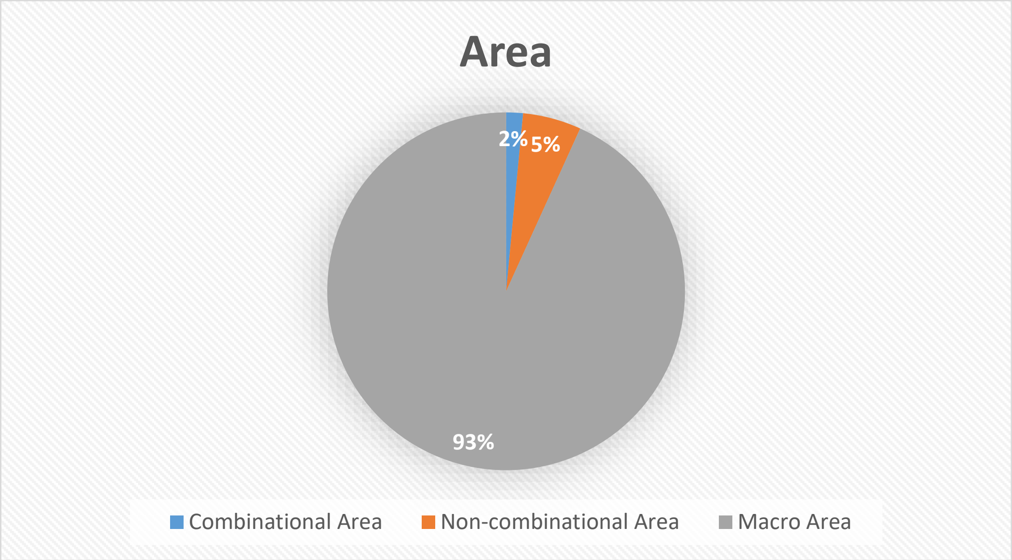
4.1.1 Flat flow:

4.1.1.1 Area results:

Table 8 : Flat Flow Area Results

	After Removing Hierarchy
Total Cell Area	16311413.9 μm^2
Net	1547462.2 μm^2
Total Design Area	17858876.2 μm^2

Combinational Area	247349.2 μm^2
Non-combinational Area	862207.2 μm^2
Macro Area	15201857.5 μm^2

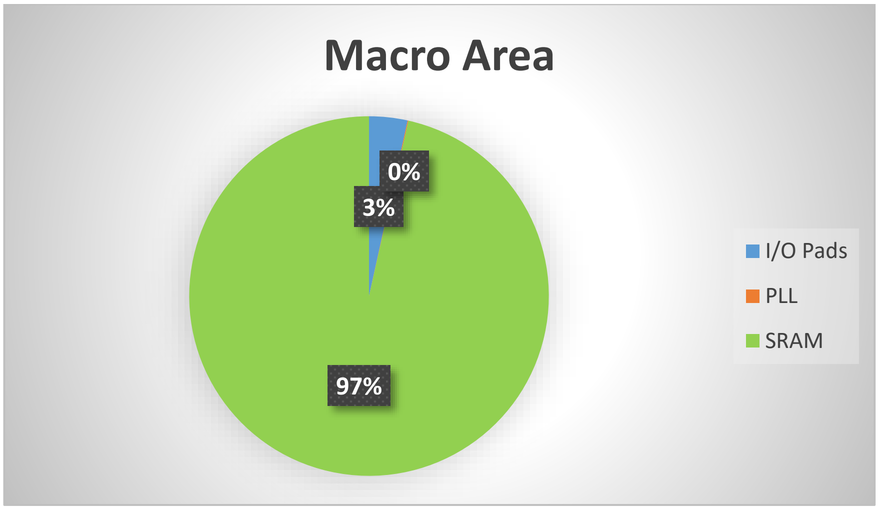


Macros take up 93% of the design area. Those macros are I/O Pads, SRAMs and PLL.

Table 9: Macro Area

I/O Pads	528000 μm^2
PLL	10873.333 μm^2

SRAM	14768716.43 μm²

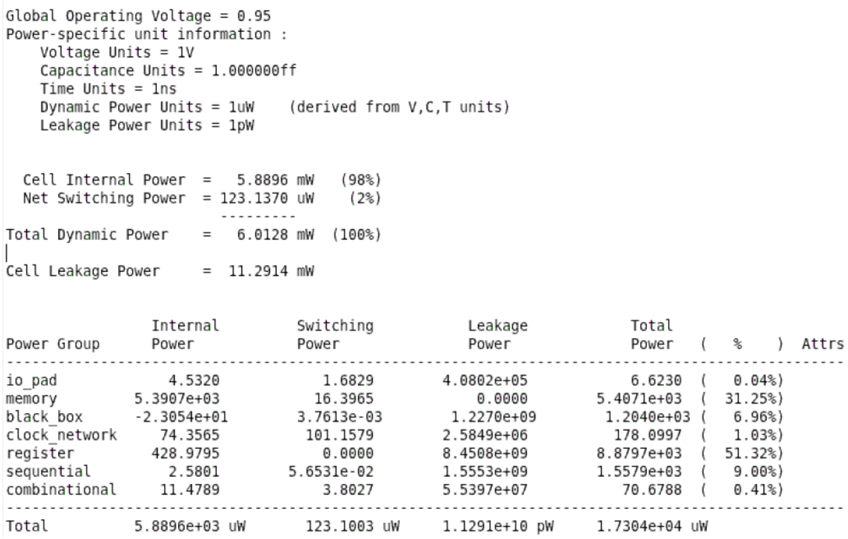


As shown in the above chart the SRAM takes up most of the Macro Area, this is because the design uses three types of memory banks Interleaved memory bank, Private memory bank and ROM bank. Those banks are large sized and due to the limited size of SRAM instance available in SAED 32nm library, which is $512 \times 32\ bits$, we used multiple instances to build one bank of each type

Table 10: Types of memory banks

Data / Memory Type	Number of Banks	Bank Size	Number of Instances used for each bank
Interleaved Memory	4	$2^{15} \times 32$ bits	64
Private Memory	2	$2^{13} \times 32$ bits	16
ROM	1	$2^{9} \times 32$ bits	4

4.1.1.2 Power results:

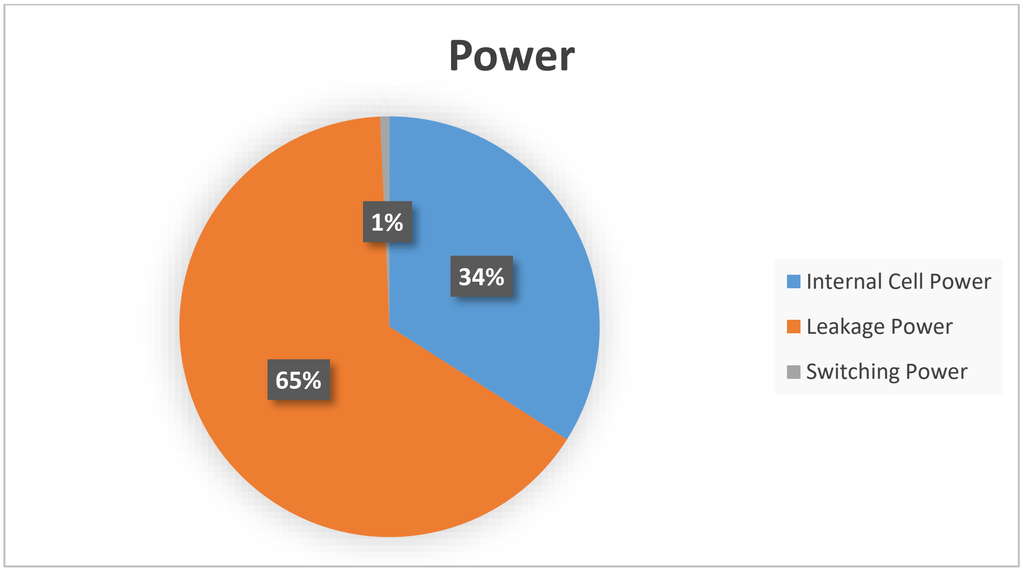


```
Global Operating Voltage = 0.95
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =   5.8896 mW   (98%)
  Net Switching Power  = 123.1370 uW    (2%)
                         ---------
Total Dynamic Power    =   6.0128 mW  (100%)

Cell Leakage Power     =  11.2914 mW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             4.5320           1.6829        4.0802e+05            6.6230  (   0.04%)
memory           5.3907e+03        16.3965            0.0000        5.4071e+03  (  31.25%)
black_box       -2.3054e+01       3.7613e-03       1.2270e+09        1.2040e+03 (   6.96%)
clock_network     74.3565         101.1579        2.5849e+06          178.0997  (   1.03%)
register         428.9795           0.0000        8.4508e+09        8.8797e+03  (  51.32%)
sequential         2.5801         5.6531e-02      1.5553e+09        1.5579e+03  (   9.00%)
combinational     11.4789           3.8027        5.5397e+07           70.6788  (   0.41%)
--------------------------------------------------------------------------------------------------
Total            5.8896e+03 uW      123.1003 uW     1.1291e+10 pW     1.7304e+04 uW
```

Figure 17: power report after synthesis (flat flow)

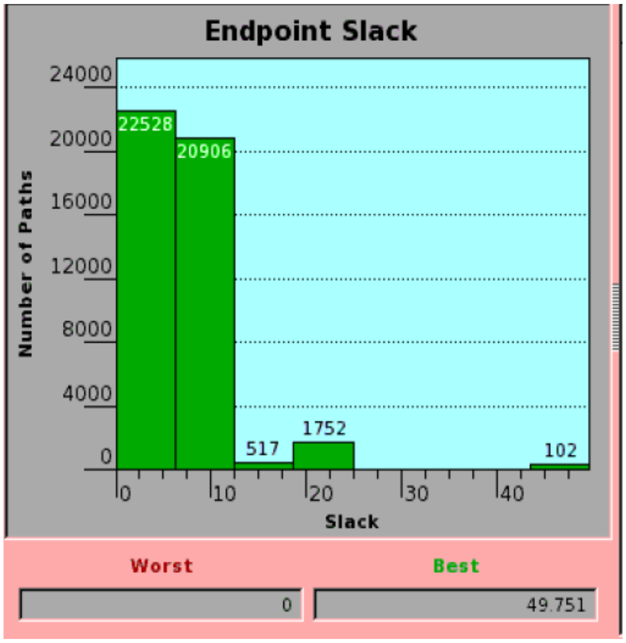
Table 11: Power Results

Internal Cell Power	5.88 mW
Leakage Power	11.291 mW
Switching Power	123.1 µW
Total Power	17.304 mW



The above results shows that most of power consumption is due to leakage power which is the static power consumed in the transistor due to constant current from VDD to Ground and it increases with shrinking the transistor size, that is why its value is high in our case. The second most type of consumed power is the internal power which is dissipated within the cell boundary in the form of dissipation during switching due to charging or discharging of any internal capacitors and the short circuit power which is caused by the flow of a current through a direct path between VDD and Ground during switching of both NMOS and PMOS transistors for a short period of time. The third type is Switching power which is dissipated in load capacitance during charging and discharging.

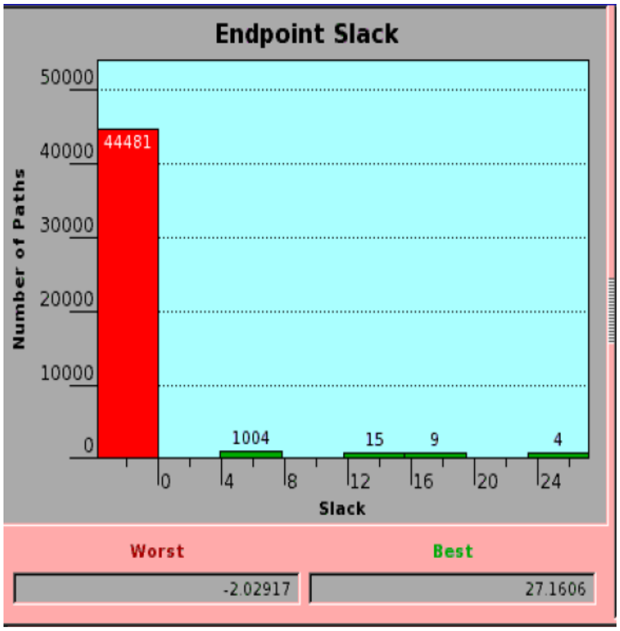
4.1.1.3 Timing results:



Worst | 0

Best | 49.751

Figure 18: Setup time histogram after synthesis (flat flow)

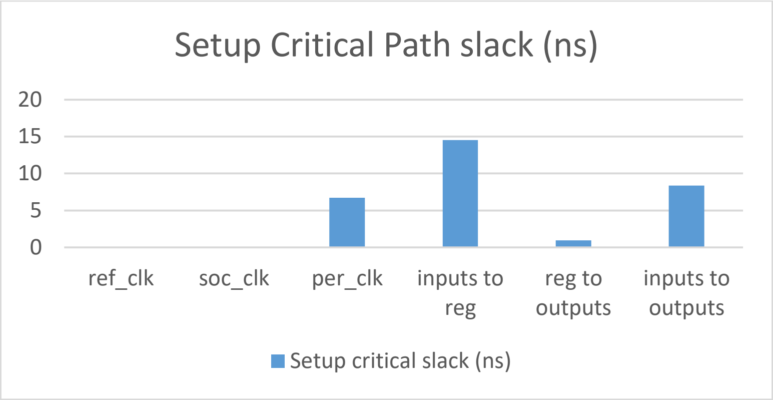


Worst | -2.02917

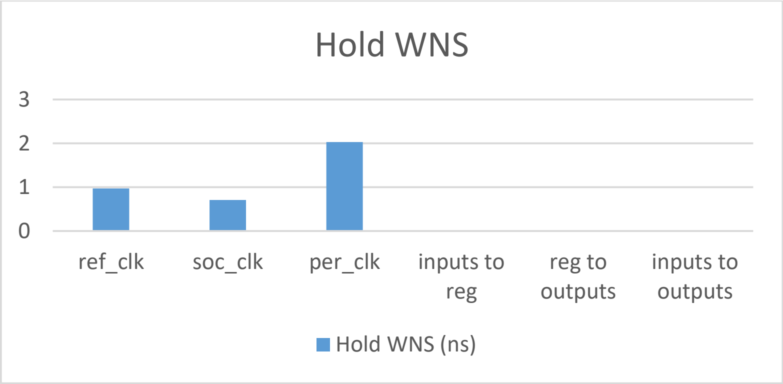
Best | 27.1606

Figure 19: Hold time histogram after synthesis (flat flow)

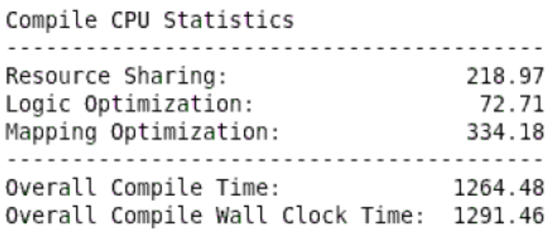
The chart below shows the worst critical slack in each clock domain among the setup timing checks:



The chart below shows the worst negative slack in each clock domain among the hold timing checks:



4.1.1.4 Run time:



```
Compile CPU Statistics
-----------------------------------------
Resource Sharing:                  218.97
Logic Optimization:                 72.71
Mapping Optimization:              334.18
-----------------------------------------
Overall Compile Time:             1264.48
Overall Compile Wall Clock Time:  1291.46
```

Figure 20: CPU time of synthesis (flat flow)

Total Compile time equals 1264.48 seconds which equals 21.07 minutes.

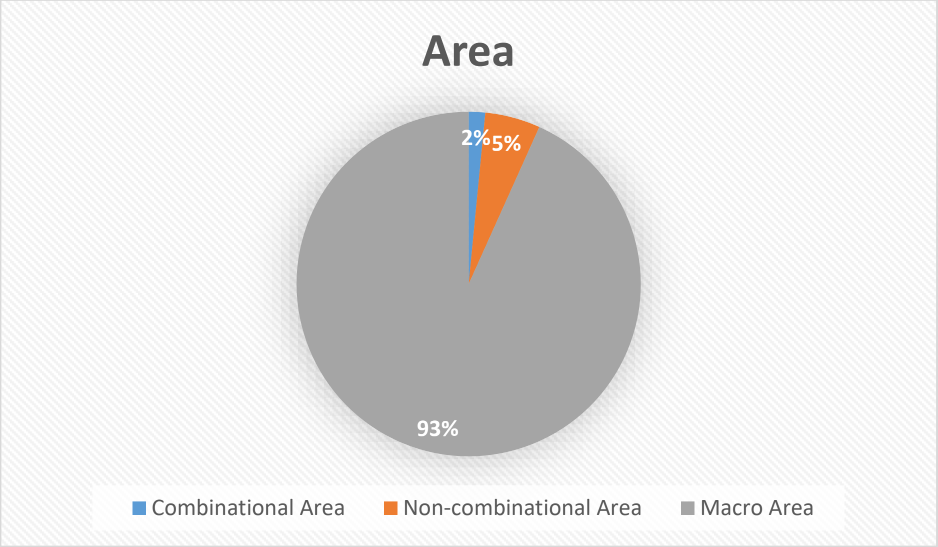
4.1.2 Hierarchical flow:

4.1.2.1 Area:

Table 12 : Hierarchical Flow Area Results

	After Removing Hierarchy
Total Cell Area	16300430 µm^2
Net Area	334048 µm^2
Total Design Area	16634478 µm^2

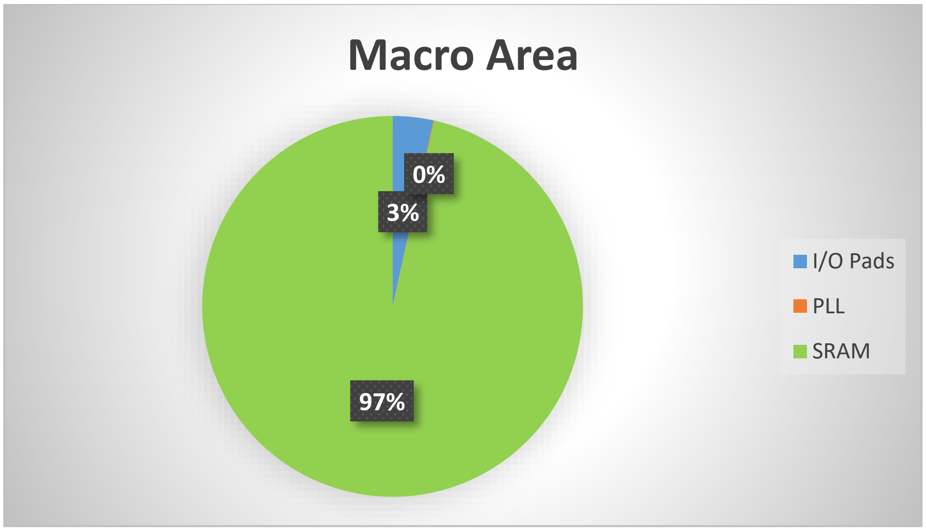
Combinational Area	252014 µm^2
Non-combinational Area	846558 µm^2
Macro Area	15201857.5 µm^2



Macro cells take up 93% of the design area. Those macro cells are I/O Pads, SRAMs and PLL.

Table 13: Macro Types

I/O Pads	528000 μm^2
PLL	10873.333 μm^2
SRAM	14768716.43 μm^2

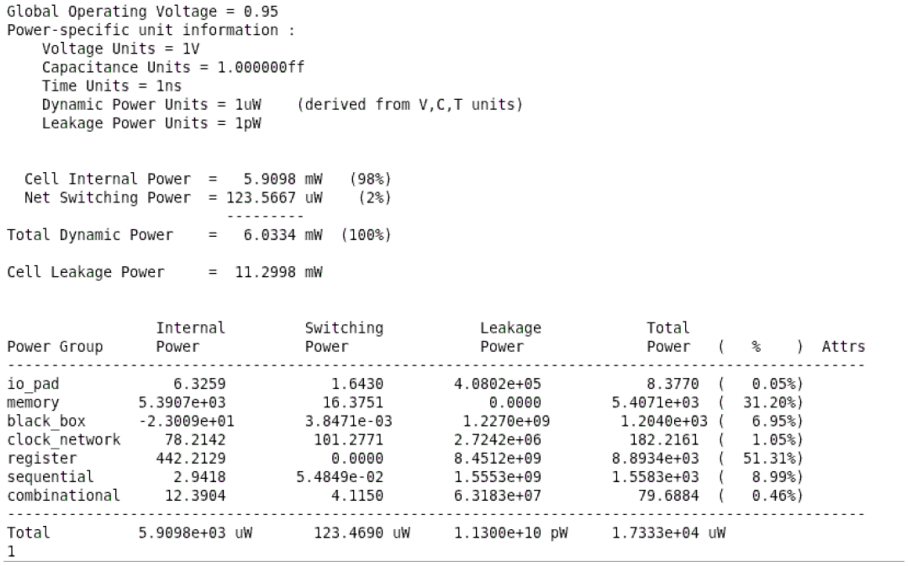


As shown in the above chart the SRAM takes up most of the Macro Area, this is because the design uses three types of memory banks interleaved memory bank, Private memory bank and ROM bank. Those banks are large sized and due to the limited size of SRAM instance available in SAED 32nm library, which is $512 \times 32\ bits$, we used multiple instances to build one bank of each type.

Table 14: Memory Types

Data Memory Type	Number of Banks	Bank Size	Number of Instances used for each bank
Interleaved Memory	4	$2^{15} \times 32$ bits	64
Private Memory	2	$2^{13} \times 32$ bits	16
ROM	1	$2^{9} \times 32$ bits	4

4.1.2.1 Power Results:

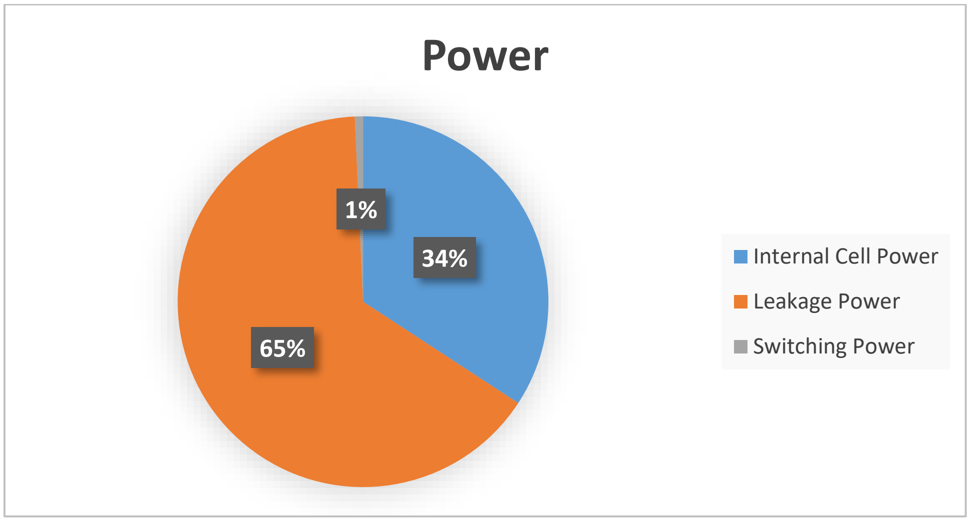


```
Global Operating Voltage = 0.95
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =   5.9098 mW   (98%)
  Net Switching Power  = 123.5667 uW    (2%)
                         ---------
Total Dynamic Power    =   6.0334 mW  (100%)

Cell Leakage Power     =  11.2998 mW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             6.3259            1.6430        4.0802e+05             8.3770  (   0.05%)
memory          5.3907e+03          16.3751            0.0000        5.4071e+03  (  31.20%)
black_box      -2.3009e+01        3.8471e-03        1.2270e+09        1.2040e+03 (   6.95%)
clock_network     78.2142          101.2771        2.7242e+06          182.2161  (   1.05%)
register         442.2129            0.0000        8.4512e+09        8.8934e+03  (  51.31%)
sequential         2.9418        5.4849e-02        1.5553e+09        1.5583e+03  (   8.99%)
combinational     12.3904            4.1150        6.3183e+07           79.6884  (   0.46%)
--------------------------------------------------------------------------------------------------
Total           5.9098e+03 uW      123.4690 uW     1.1300e+10 pW     1.7333e+04 uW
1
```

Figure 21: power report after logic synthesis (hierarchical flow)

Table 15: Power Results

Internal Cell Power	5.94 mW
Leakage Power	11.3 mW
Switching Power	123.46 µW
Total Power	17.33 mW



The above results shows that most of power consumption is due to leakage power which is the static power consumed in the transistor due to constant current from VDD to Ground and it increases with shrinking the transistor size, that is why its value is high in our case. The second most type of consumed power is the internal power which is dissipated within the cell boundary in the form of dissipation during switching due to charging or discharging of any internal capacitors and the short circuit power which is caused by the flow of a current through a direct path between VDD and Ground during switching of both NMOS and PMOS transistors for a short period of time. The third type is Switching power which is dissipated in load capacitance during charging and discharging.

4.1.2.2 Timing Results:

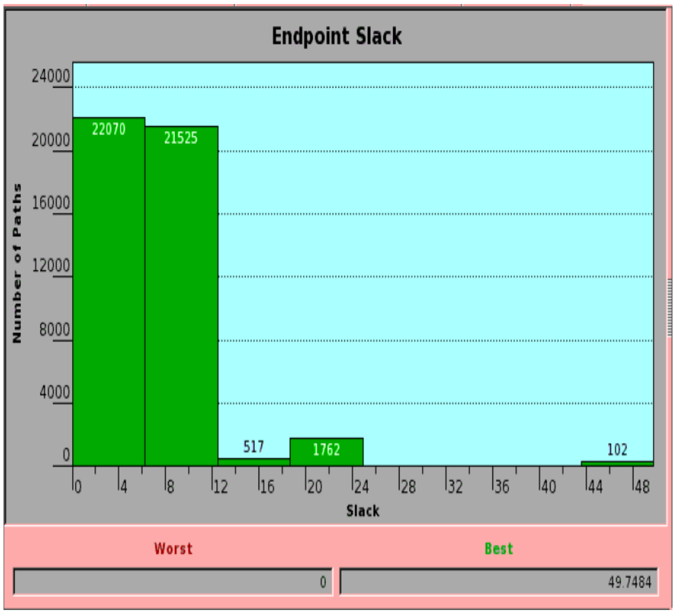


Figure 2: Setup Time Histogram after Logic Synthesis (Hierarchical Flow)

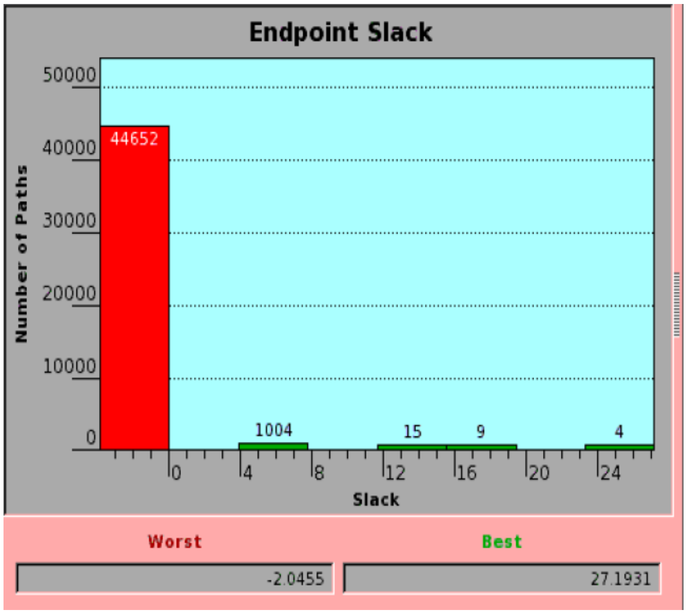
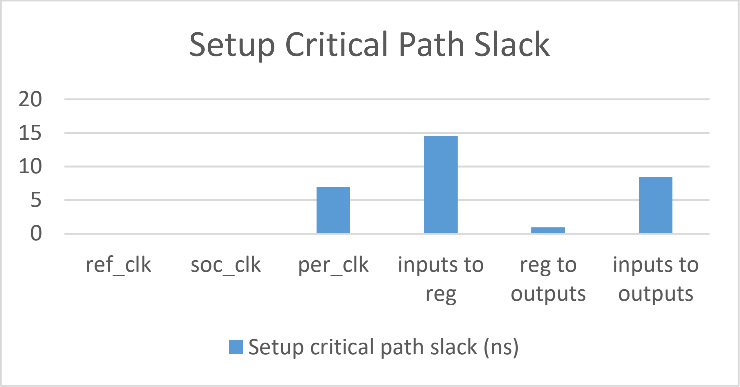
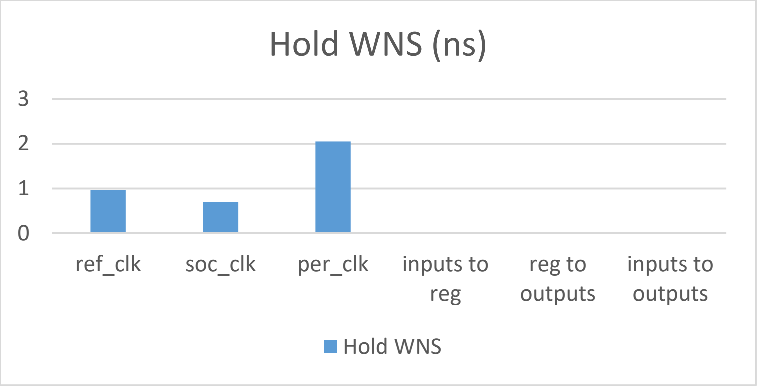


Figure 22: Hold Time Histogram after Logic Synthesis (Hierarchical Flow)

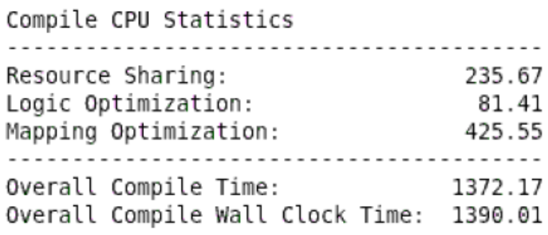
The chart below shows the worst negative slack in each clock domain among the hold timing checks:



The chart below shows the worst negative slack in each clock domain among the hold timing checks:



4.1.2.3 Run Time:



```
Compile CPU Statistics
-----------------------------------------
Resource Sharing:                  235.67
Logic Optimization:                 81.41
Mapping Optimization:              425.55
-----------------------------------------
Overall Compile Time:             1372.17
Overall Compile Wall Clock Time:  1390.01
```

Figure 23: CPU time of synthesis (hierarchal flow)

Total Compile time equals 1390.01 seconds which equals 23.16 minutes.

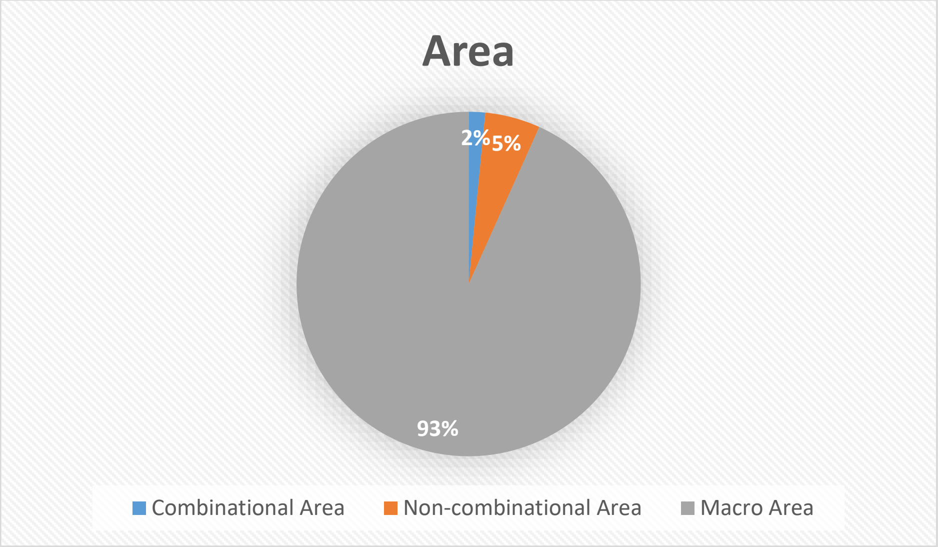
4.1.3 The second pass in Topographical flow:

4.1.2.2 Area:

Table 16 : Topographical Flow Area Results

	After Removing Hierarchy
Total Cell Area	16462564.8 μm^2
Net Area	0 μm^2
Total Design Area	16462564.8 μm^2

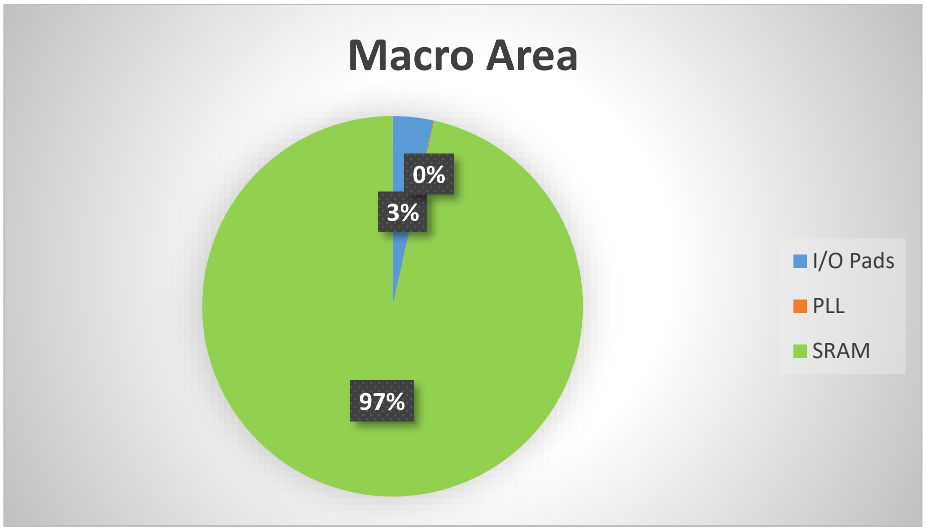
Combinational Area	411211.6 μm^2
Non-combinational Area	849495.8 μm^2
Macro Area	15201857.5 μm^2



Macro cells take up 93% of the design area. Those macro cells are I/O Pads, SRAMs and PLL.

Table 17: Macro Types

I/O Pads	528000 μm^2
PLL	10873.333 μm^2
SRAM	14768716.43 μm^2

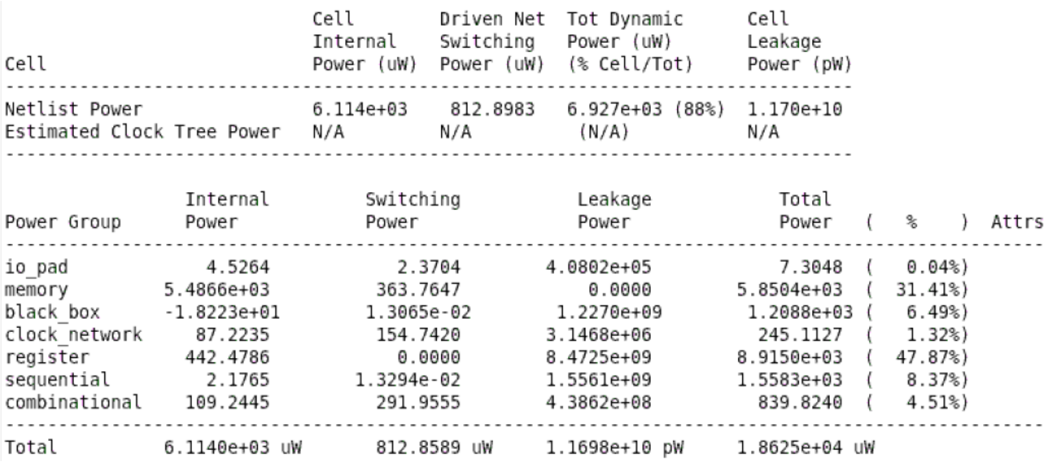


As shown in the above chart the SRAM takes up most of the Macro Area, this is because the design uses three types of memory banks Interleaved memory bank, Private memory bank and ROM bank. Those banks are large sized and due to the limited size of SRAM instance available in SAED 32nm library, which is $512 \times 32\ bits$, we used multiple instances to build one bank of each type.

Table 18: Memory Types

Data / Memory Type	Number of Banks	Bank Size	Number of Instances used for each bank
Interleaved Memory	4	$2^{15} \times 32$ bits	64
Private Memory	2	$2^{13} \times 32$ bits	16
ROM	1	$2^{9} \times 32$ bits	4

4.1.2.4 Power Results:

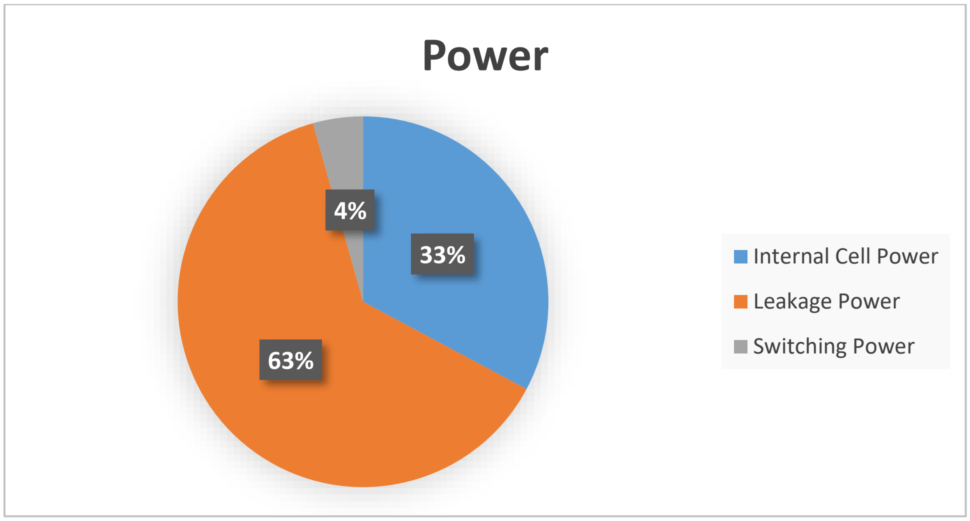


```
                             Cell        Driven Net  Tot Dynamic      Cell
                             Internal    Switching   Power (uW)       Leakage
Cell                         Power (uW)  Power (uW)  (% Cell/Tot)     Power (pW)
--------------------------------------------------------------------------------
Netlist Power                6.114e+03    812.8983   6.927e+03 (88%)  1.170e+10
Estimated Clock Tree Power   N/A         N/A          (N/A)           N/A
--------------------------------------------------------------------------------

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             4.5264            2.3704        4.0802e+05            7.3048  (   0.04%)
memory         5.4866e+03          363.7647            0.0000        5.8504e+03  (  31.41%)
black_box     -1.8223e+01        1.3065e-02        1.2270e+09        1.2088e+03 (   6.49%)
clock_network     87.2235          154.7420        3.1468e+06          245.1127  (   1.32%)
register         442.4786            0.0000        8.4725e+09        8.9150e+03  (  47.87%)
sequential         2.1765        1.3294e-02        1.5561e+09        1.5583e+03  (   8.37%)
combinational    109.2445          291.9555        4.3862e+08          839.8240  (   4.51%)
--------------------------------------------------------------------------------------------------
Total          6.1140e+03 uW       812.8589 uW     1.1698e+10 pW     1.8625e+04 uW
```

Figure 24: power report after logic synthesis (Topographical Flow)

Table 19: Power Results

Internal Cell Power	6.11 mW
Leakage Power	11.698 mW
Switching Power	812.86 µW
Total Power	18.625 mW



The above results shows that most of power consumption is due to leakage power which is the static power consumed in the transistor due to constant current from VDD to Ground and it increases with shrinking the transistor size, that is why its value is high in our case. The second most type of consumed power is the internal power which is dissipated within the cell boundary in the form of dissipation during switching due to charging or discharging of any internal capacitors and the short circuit power which is caused by the flow of a current through a direct path between VDD and Ground during switching of both NMOS and PMOS transistors for a short period of time. The third type is Switching power which is dissipated in load capacitance during charging and discharging.

4.1.2.5 Timing Results:

Setup Time

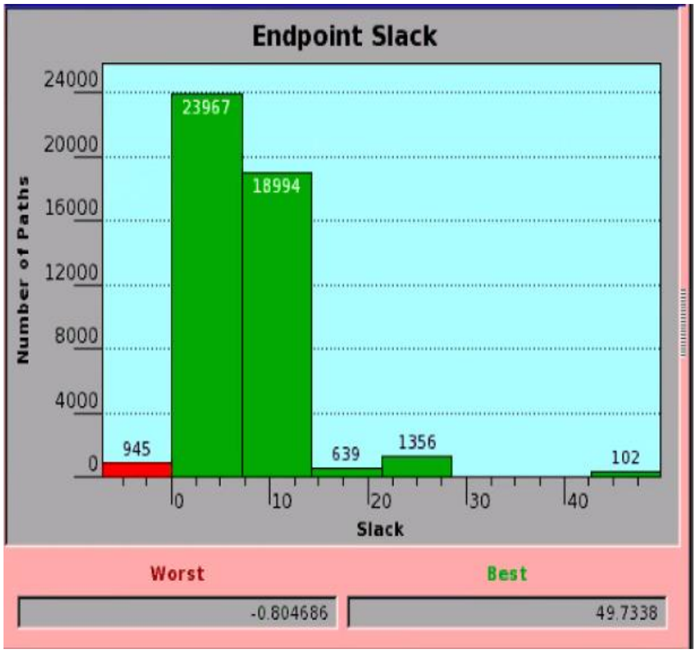
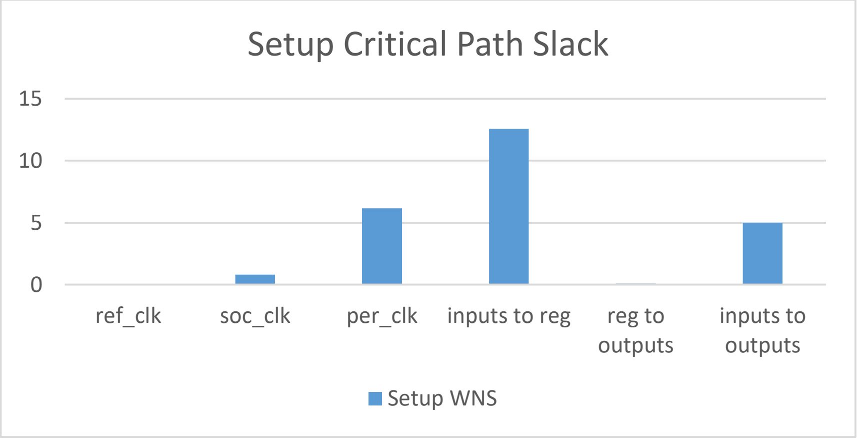
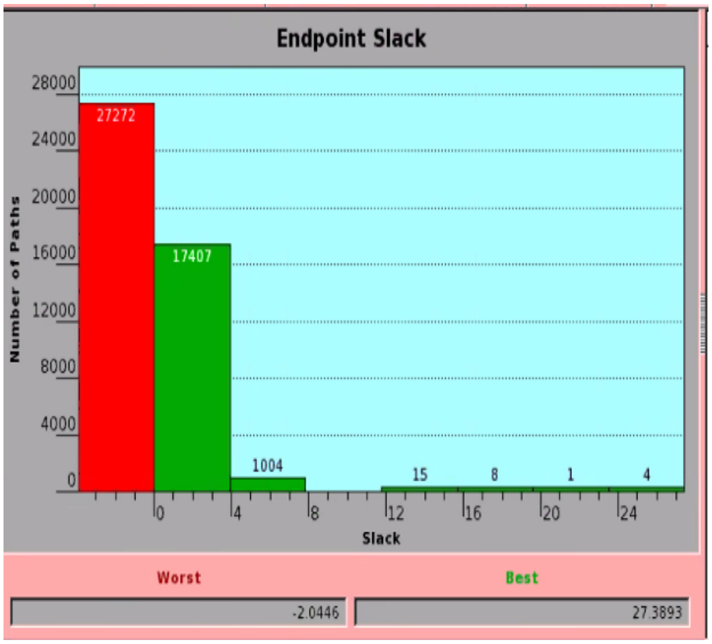


Figure 2: Setup time: Hold Time Histogram after 2nd Pass Logic Synthesis (Topographical Flow)

From the above figure, there is Setup Time violation equals -0.8 ns which is small enough to be solved in the later optimizations after Design Planning stage.



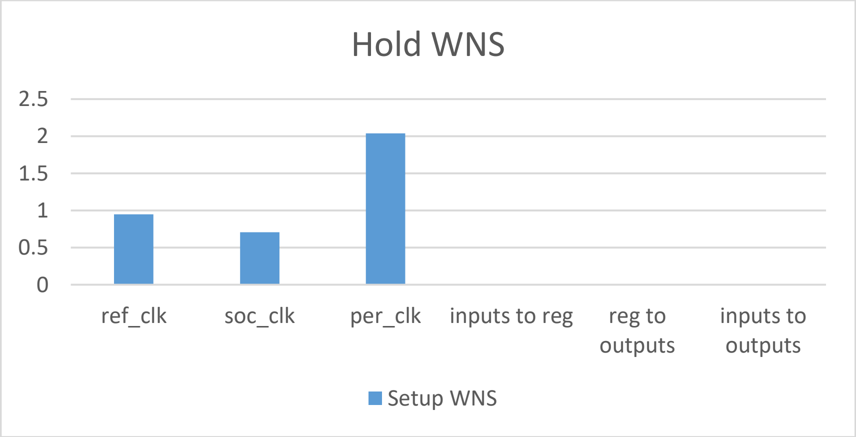
Hold Time



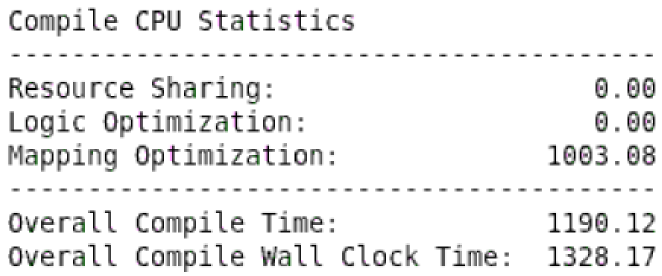
Worst -2.0446

Best 27.3893

Figure 25: Hold Time Histogram after 2nd Pass Logic Synthesis (Topographical Flow)



4.1.2.6 Run Time:



```
Compile CPU Statistics
----------------------------------------
Resource Sharing:                   0.00
Logic Optimization:                 0.00
Mapping Optimization:            1003.08
----------------------------------------
Overall Compile Time:            1190.12
Overall Compile Wall Clock Time: 1328.17
```

Total Compile time equals 1328.17 seconds which equals 22.13 minutes.

4.2. Design planning:

4.2.1 Flat flow:

Table 20: Design Planning (Flat Flow)

Core Utilization	0.401
Number of Rows	3708
Core Width(micron)	6200
Core Height(micron)	6200
Aspect Ratio	1

From the above table core area can be calculated to be 38.44 mm^2 and the die area of the chip is 49 mm^2, where the die height is 7mm and the die width is 7mm.

IR drop map of the power network:

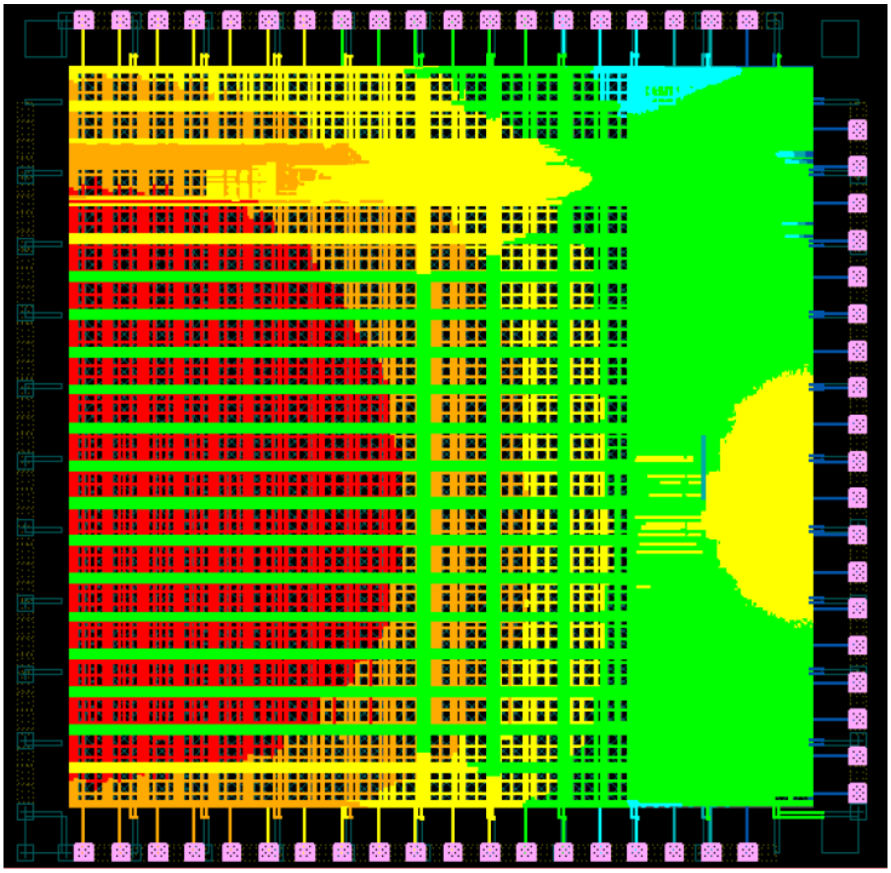
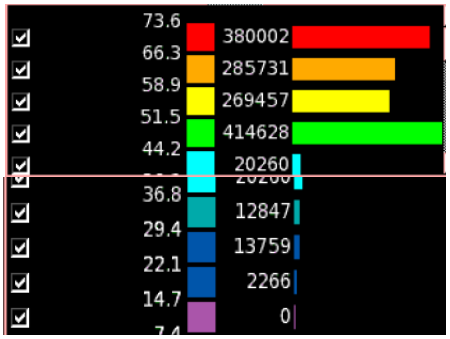


Figure 26: IR Drop map (Flat Flow)



From the above map the maximum IR drop is 73.6 mV which is 7.75% of the used power supply value which is 950 mV and this drop is acceptable since it’s lower than 10% limit recommended by Synopsys.

Congestion map at the end of the design planning stage:

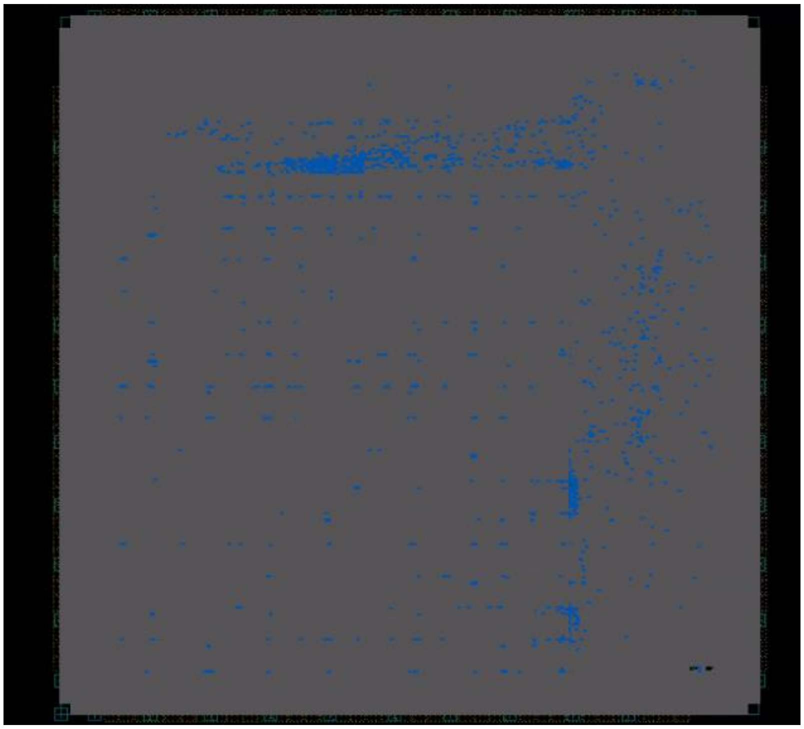
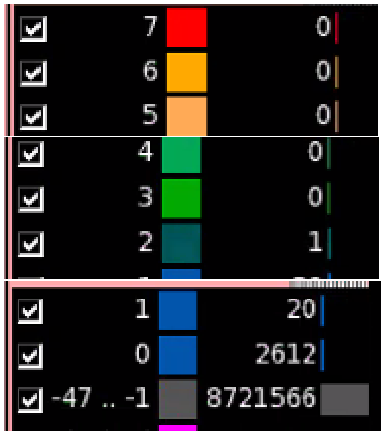


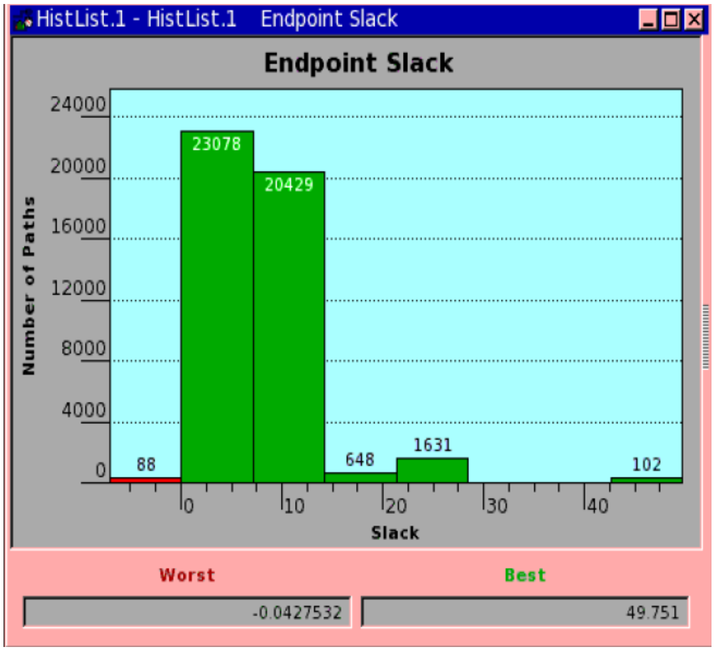
Figure 27: Congestion Map after design planning (Flat Flow)



☑	7	🟥	0
☑	6	🟧	0
☑	5	🟧	0
☑	4	🟩	0
☑	3	🟩	0
☑	2	🟩	1
☑	1	🟦	20
☑	0	🟦	2612
☑	-47 .. -1	▨	8721566

As shown in the above map there is no congestion at all on the edges of all global routing cells (GRCs), this is due to the well-defined offset between hard macros in both X and Y directions.

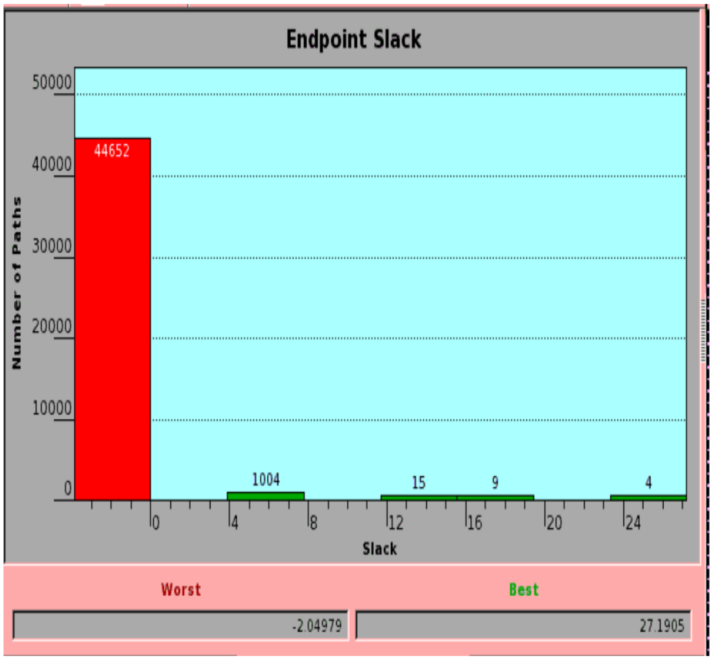
Histogram of end point slack of setup and hold timing checks:



Worst: -0.0427532

Best: 49.751

Figure 28: Setup Histogram after Design Planning (FlatFlow)

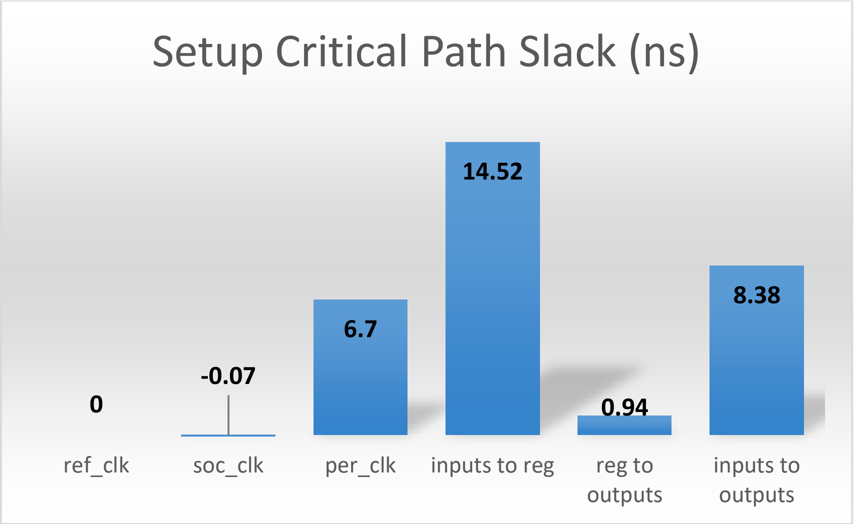


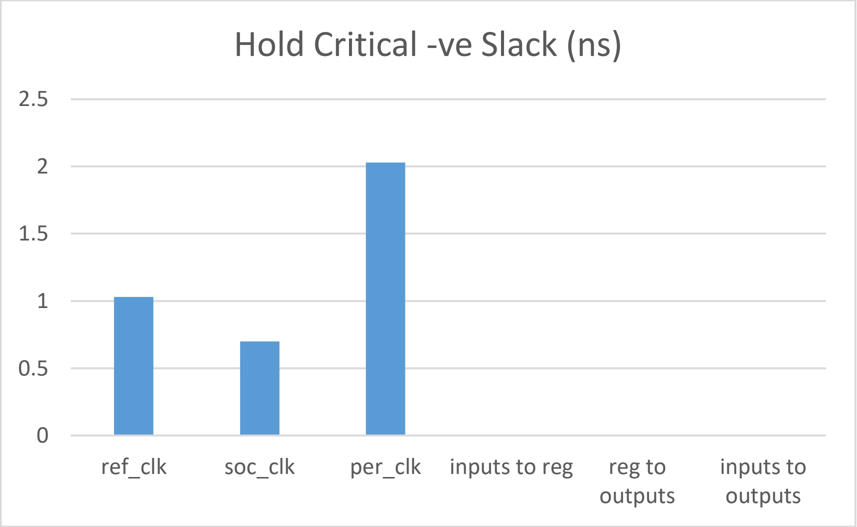
Worst: -2.04979

Best: 27.1905

Figure 29: Hold Histogram after Design Planning (Flat Flow)

Critical slack on of setup and hold timing checks each clock domain:





4.2.2 Hierarchal flow:

Table 21: Design Planning (Hierarchal Flow)

Core Utilization	0.401
Number of Rows	3708
Core Width(micron)	6200
Core Height(micron)	6200
Aspect Ratio	1

From the above table core area can be calculated to be 38.44 mm^2 and the die area of the chip is 49 mm^2, where the die height is 7mm and the die width is 7mm.

IR drop map of the power network:

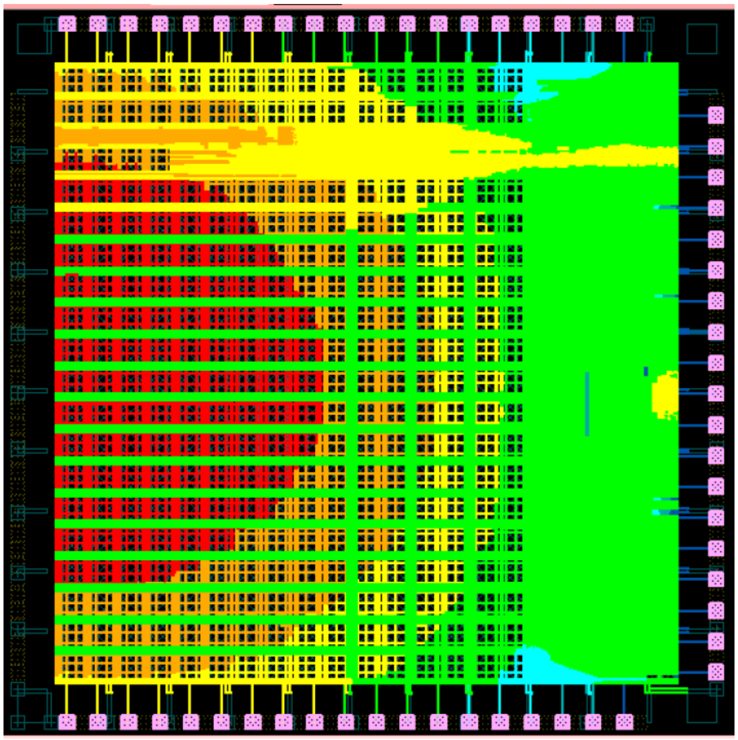
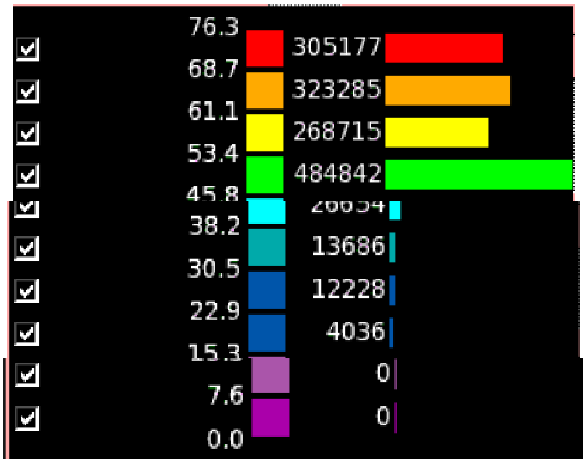


Figure 30: IR Drop map (Hierarchal Flow)



From the above map the maximum IR drop is 76.3 mV which is 8.03% of the used power supply value which is 950 mV and this drop is acceptable since it's lower than 10% limit recommended by Synopsys.

Congestion map at the end of the design planning stage:

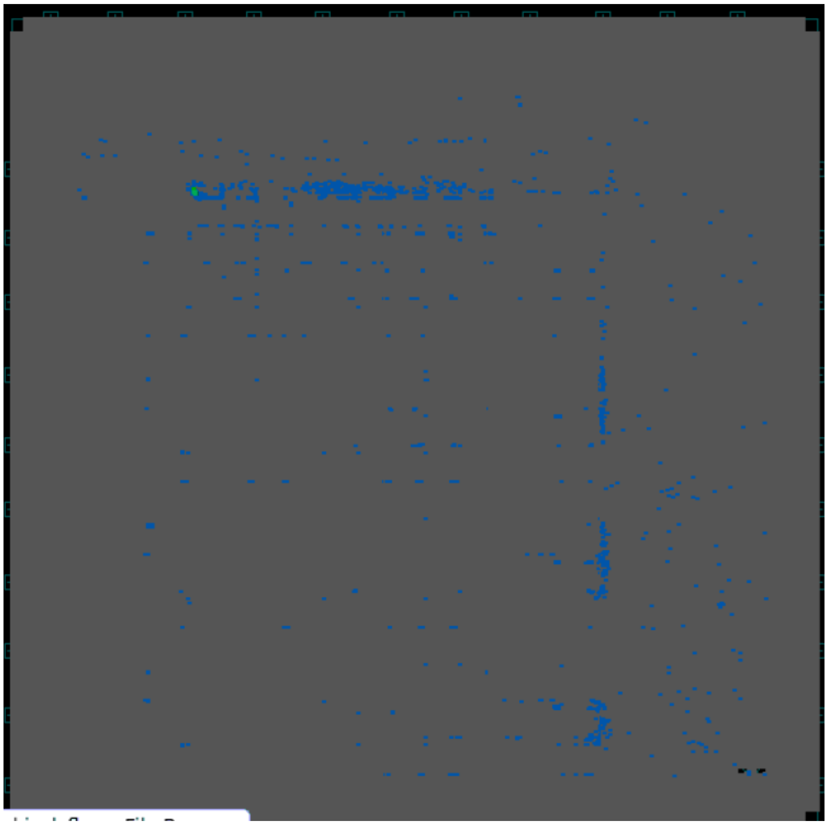
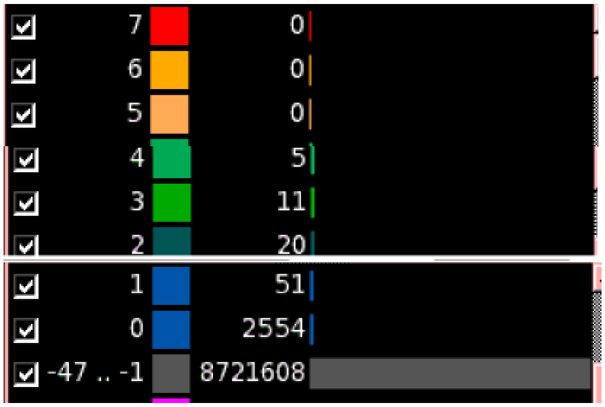


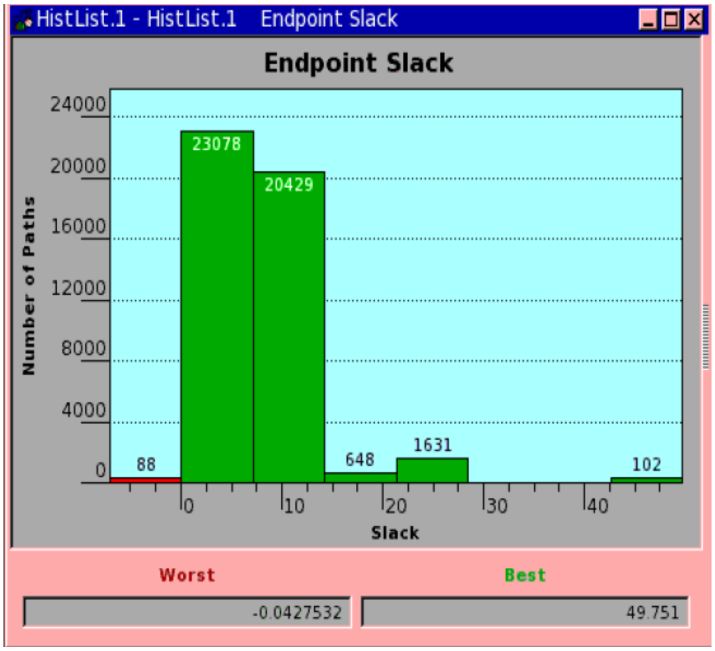
Figure 31: Congestion Map after Design Planning (Hierarchal Flow)



☑	7	🟥	0
☑	6	🟧	0
☑	5	🟧	0
☑	4	🟩	5
☑	3	🟩	11
☑	2	🟩	20
☑	1	🟦	51
☑	0	🟦	2554
☑	-47 .. -1	▨	8721608

As shown in the above map there is no congestion at all on the edges of all global routing cells (GRCs), this is due to the well-defined offset between hard macros in both X and Y directions.

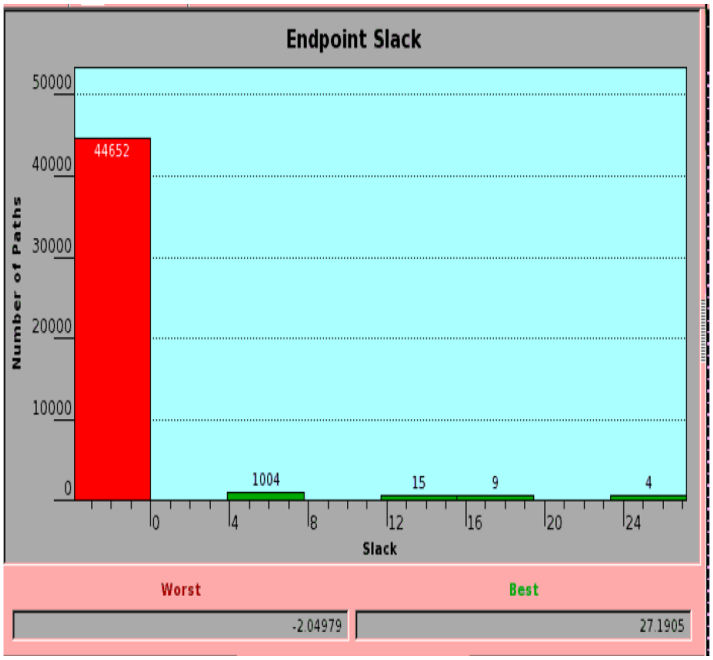
Histogram of end point slack of setup and hold timing checks:



Worst -0.0427532

Best 49.751

Figure 32: Setup Histogram after Design Planning (Hierarchal Flow)

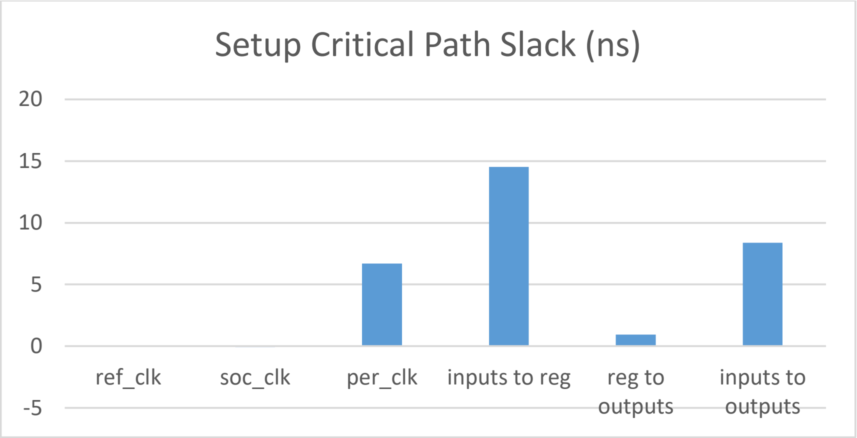


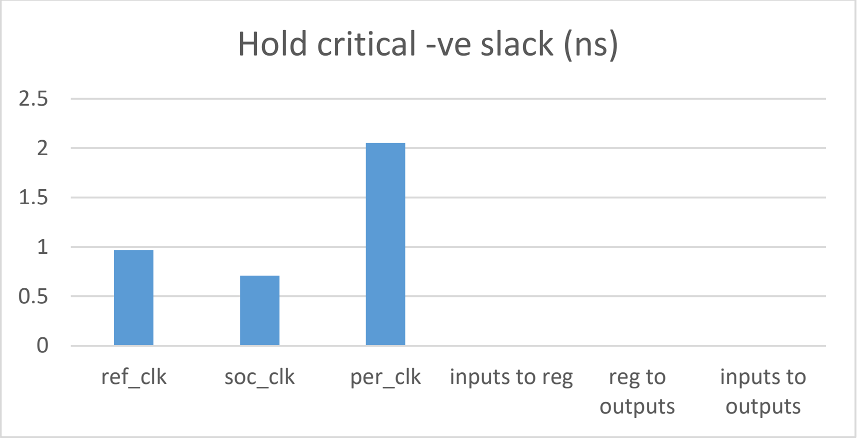
Worst -2.04979

Best 27.1905

Figure 33: Hold Histogram after Design Planning (Hierarchal Flow)

Critical slack on of setup and hold timing checks each clock domain:





4.2.3 Topographical flow:

Table 22: Design planning (topographical Flow)

Core Utilization	0.401
Number of Rows	3708
Core Width(micron)	6200
Core Height(micron)	6200
Aspect Ratio	1

From the above table core area can be calculated to be 38.44 mm^2 and the die area of the chip is 49 mm^2, where the die height is 7mm and the die width is 7mm.

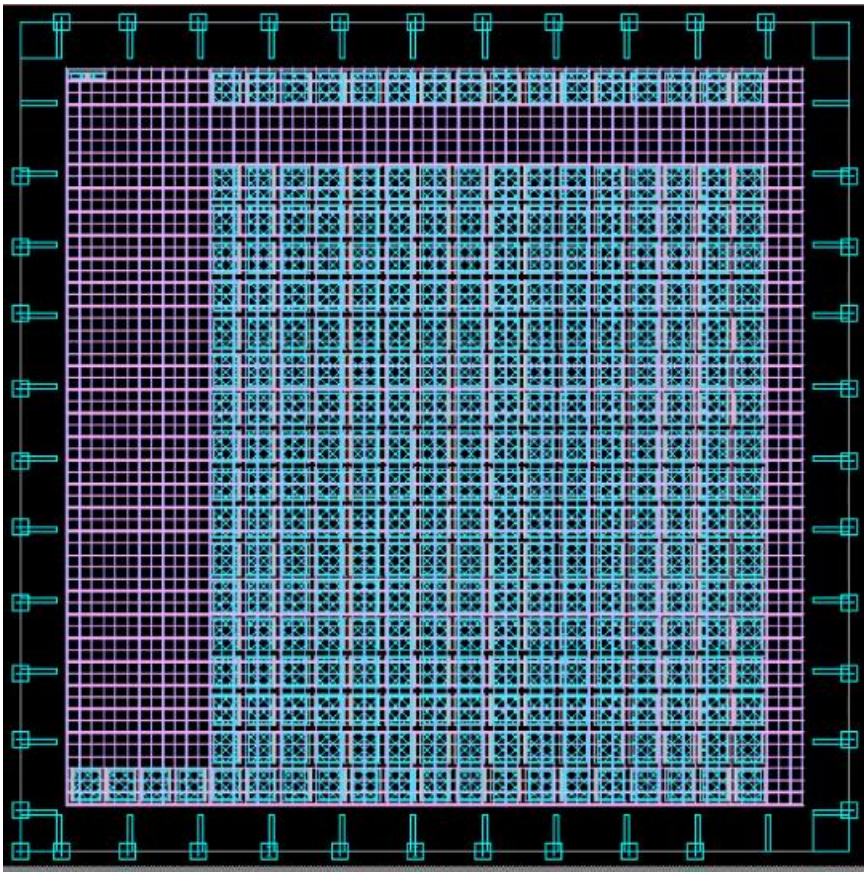


Figure 34: Design Layout (Topographical Flow)

Layout of the Topographical Flow is different from other Flows.

IR drop map of the power network:

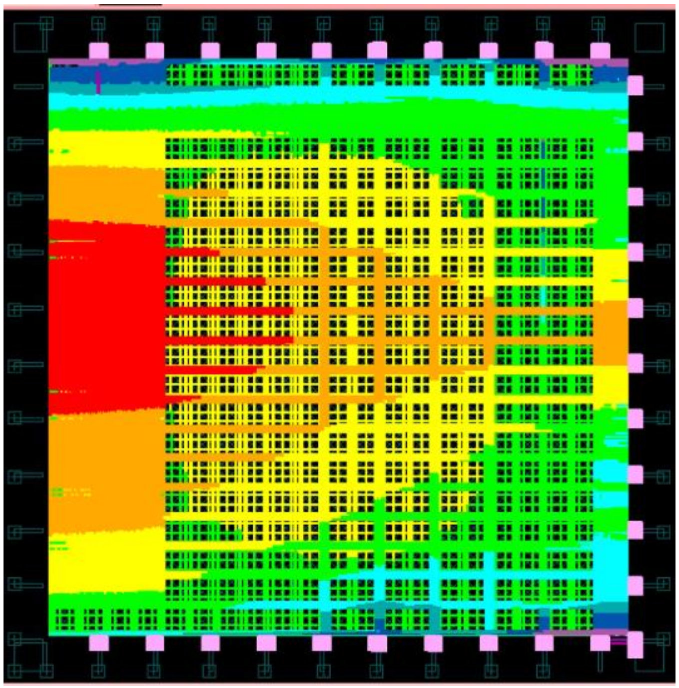
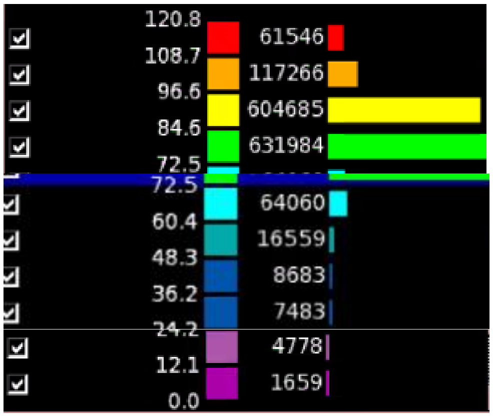


Figure 35: IR Drop map (Topographical Flow)



From the above map the maximum IR drop is 120.8 mV which is 12.7% of the used power supply value which is 950 mV. This value is higher than 10% recommended IR drop by Synopsys, however it could be improved by reducing the offset between the power and ground straps which will result in the reduction of the resistance of the power and ground mesh, therefore reduces the IR drop on them, and this can be done in the future work of this thesis, however 12.7% IR-drop is still acceptable.

Congestion map at the end of the design planning stage:

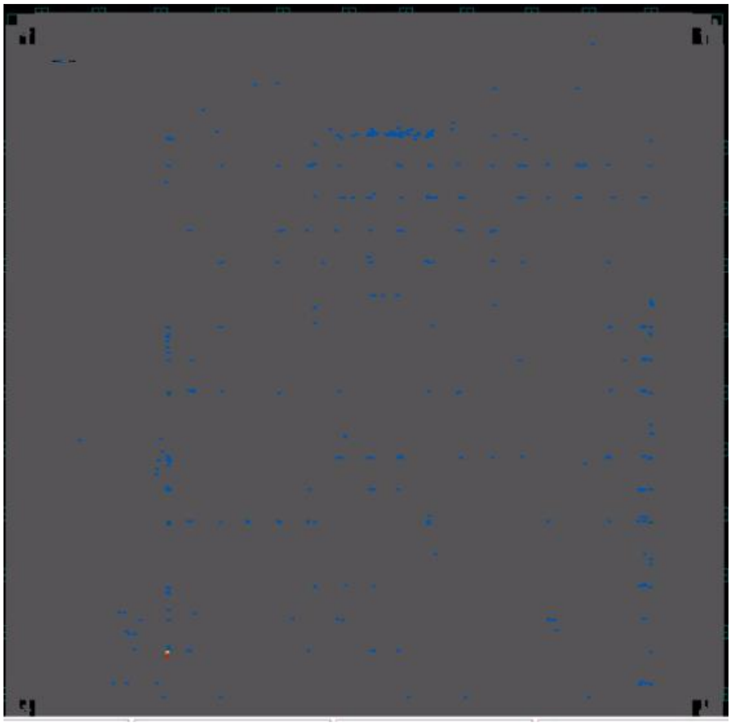
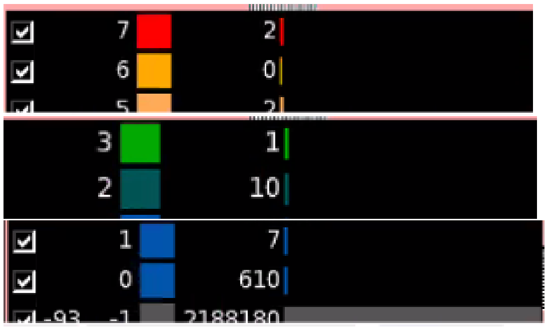


Figure 36: Congestion Map after design planning (Topographical Flow)



☑		7	🟥	2
☑		6	🟧	0
☑		5	🟧	2
		3	🟩	1
		2	🟩	10
☑		1	🟦	7
☑		0	🟦	610
☑	-93	-1	▨	2188180

As shown in the above map there is no congestion at all on the edges of all global routing cells (GRCs), this is due to the well-defined offset between hard macros in both X and Y directions.

Histogram of end point slack of setup and hold timing checks:

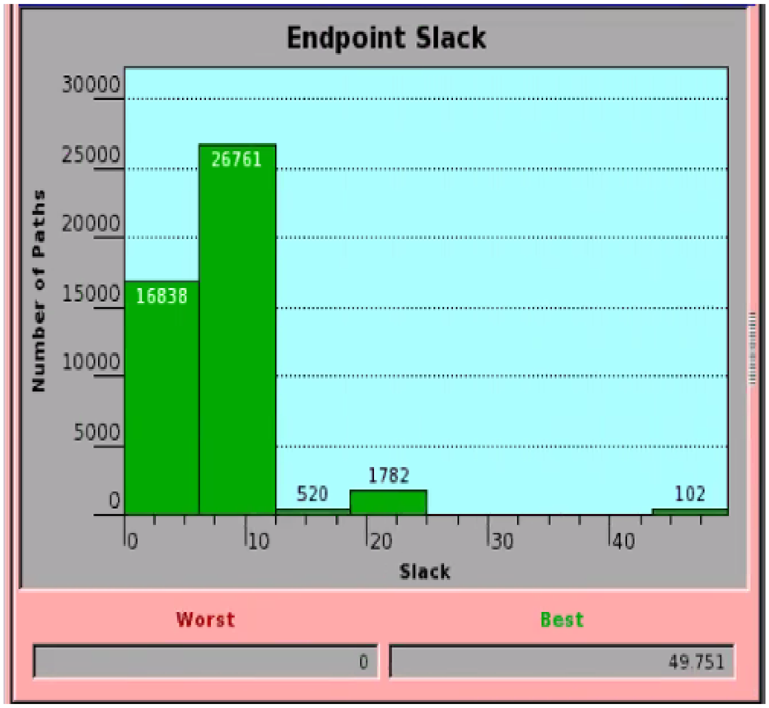


Figure 37: Setup Histogram after Design Planning (Topographical Flow)

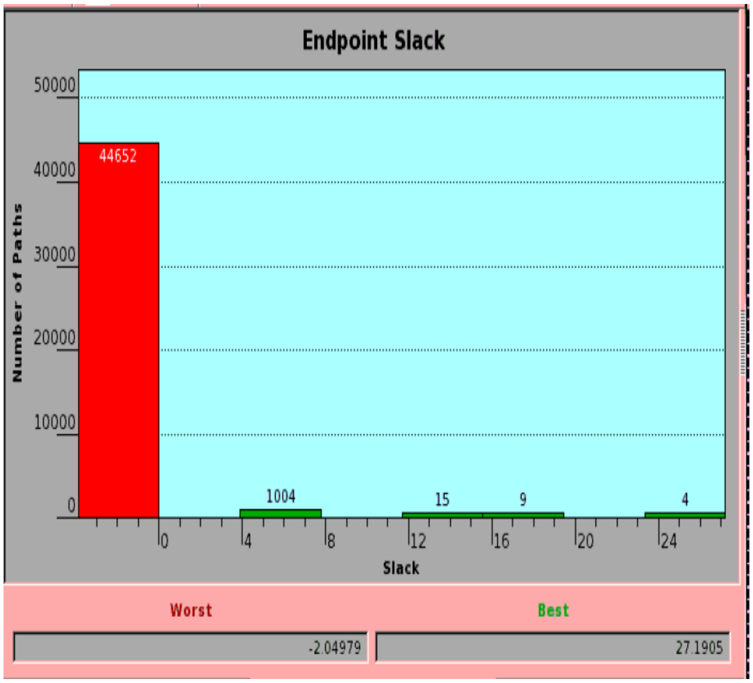
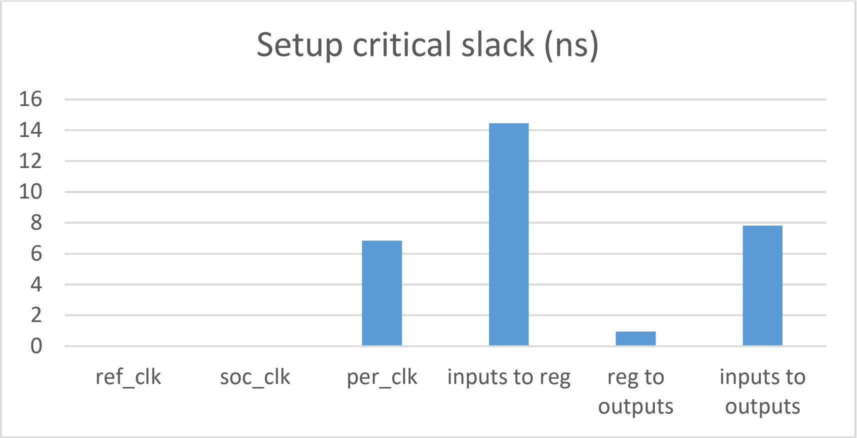
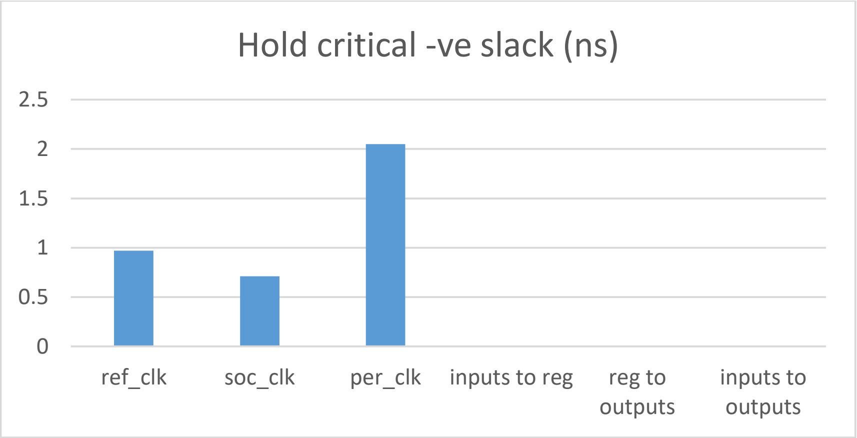


Figure 38: Hold Histogram after Design Planning (Topographical Flow)

Critical slack of setup and hold timing for each clock domain:





4.3 Placement:

4.3.1. Flat Flow

4.3.1.1. Congestion Map:

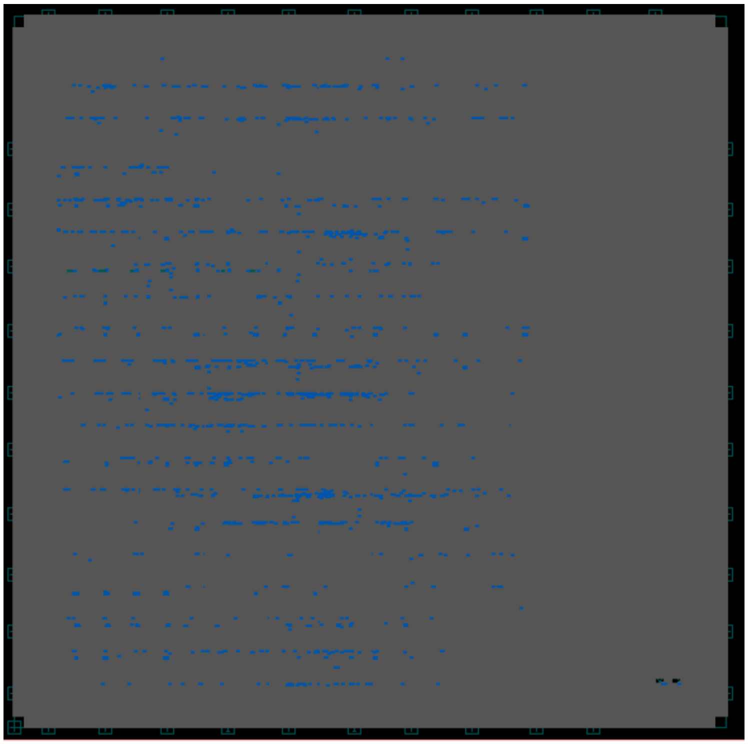
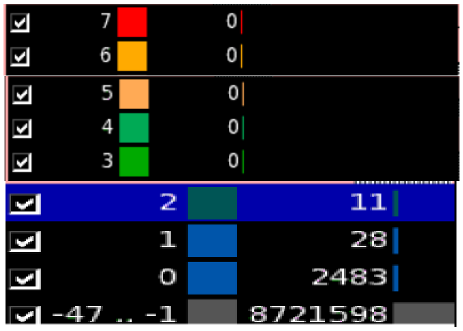


Figure 39: Congestion Map after Placement (Hierarchal Flow)

As shown in the above map there is no congestion at all on the edges of all global routing cells (GRCs), this is due to the well-defined offset between hard macros in both X and Y directions.



☑	7	🟥	0
☑	6	🟧	0
☑	5	🟧	0
☑	4	🟩	0
☑	3	🟩	0
☑	2	■	11
☑	1	🟦	28
☑	0	🟦	2483
☑	-47 .. -1	▨	8721598

4.3.1.2. Timing Results

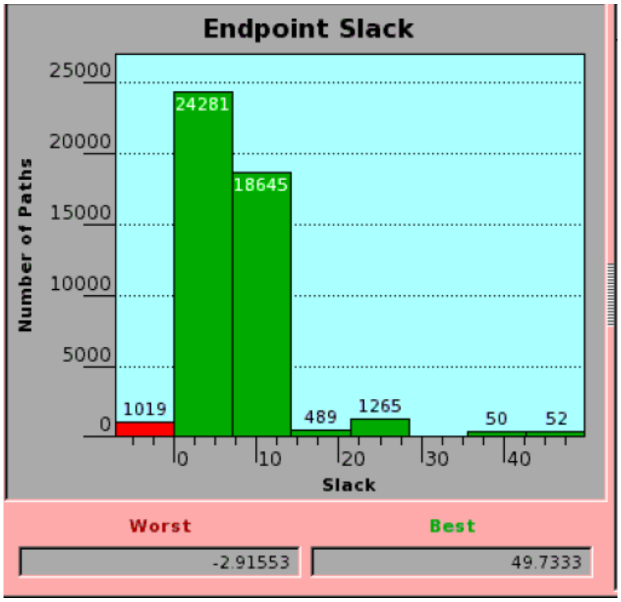


Figure 40: Setup Time Histogram after Placement (Flat Flow)

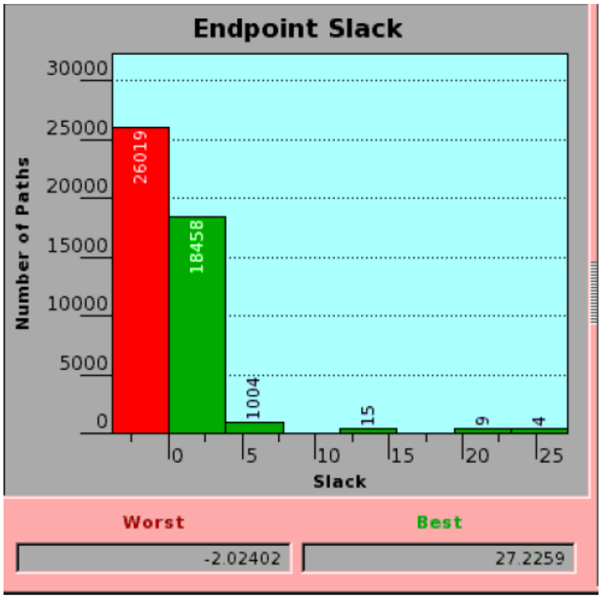
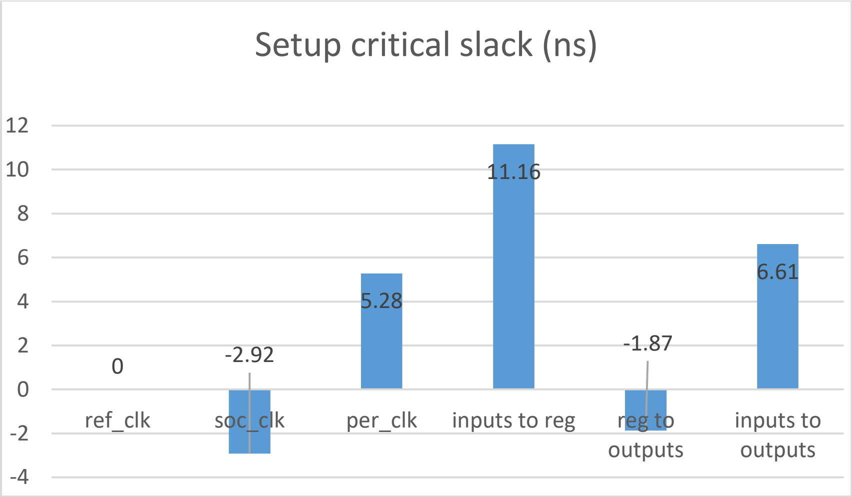
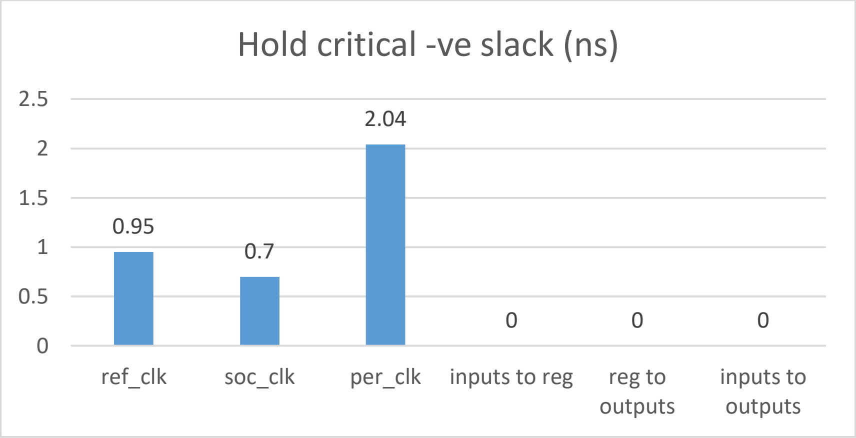


Figure 41: Hold Time Histogram after Placement (Hierarchical Flow)


Setup critical slack (ns)
12
10
8
6
4
2
0
-2
-4
0
-2.92
5.28
11.16
-1.87
6.61
ref_clk
soc_clk
per_clk
inputs to reg
reg to outputs
inputs to outputs


Hold critical -ve slack (ns)
2.5
2
1.5
1
0.5
0
0.95
0.7
2.04
0
0
0
ref_clk
soc_clk
per_clk
inputs to reg
reg to outputs
inputs to outputs

4.3.2. Hierarchical Flow

4.3.1.1. Congestion Map:

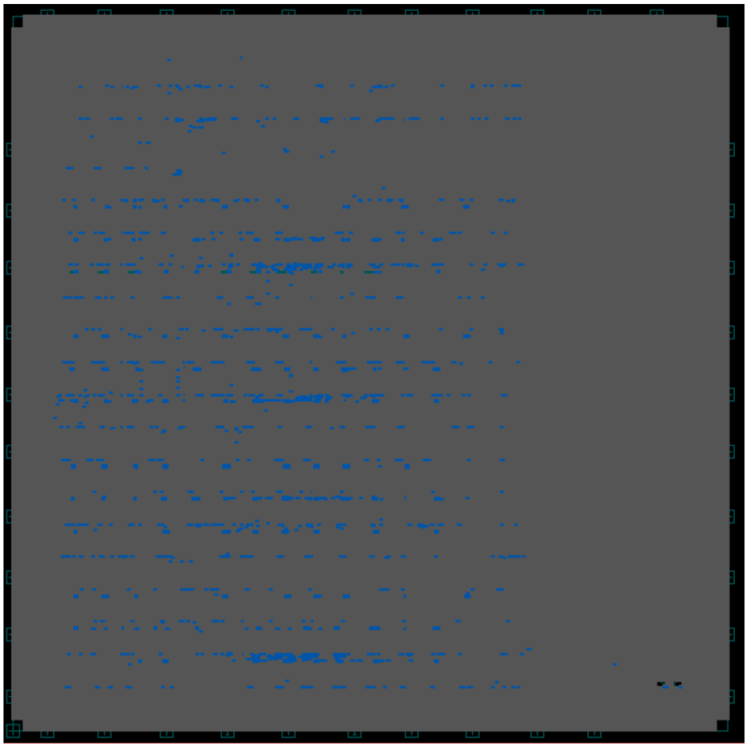
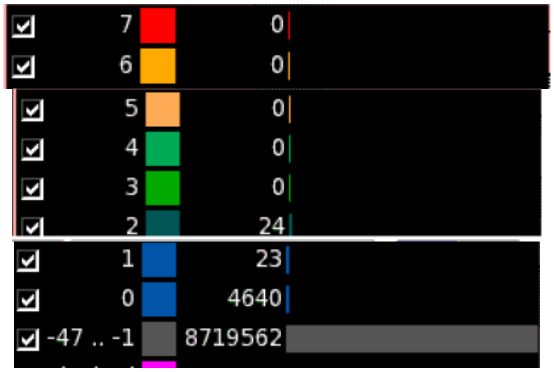


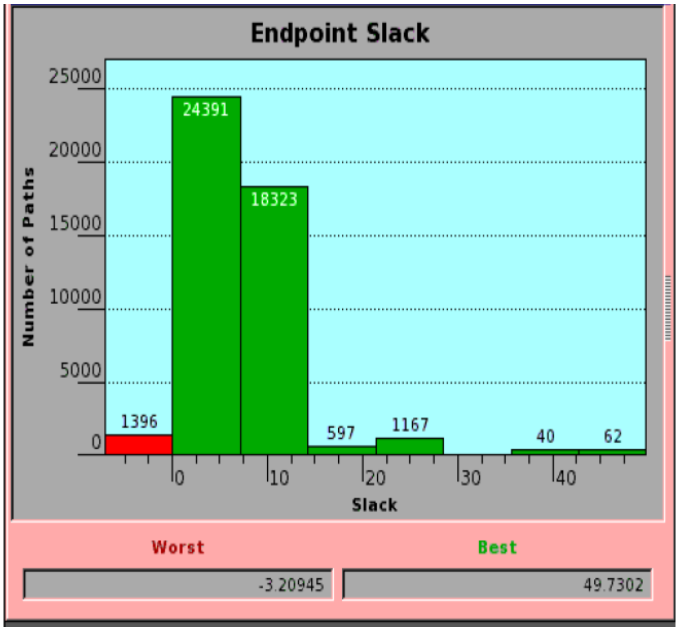
Figure 42: Congestion Map after Placement (Hierarchal Flow)

As shown in the above map there is no congestion at all on the edges of all global routing cells (GRCs), this is due to the well-defined offset between hard macros in both X and Y directions.



☑	7	🟥	0
☑	6	🟧	0
☑	5	🟧	0
☑	4	🟩	0
☑	3	🟩	0
☑	2	🟩	24
☑	1	🟦	23
☑	0	🟦	4640
☑	-47 .. -1	⬛	8719562

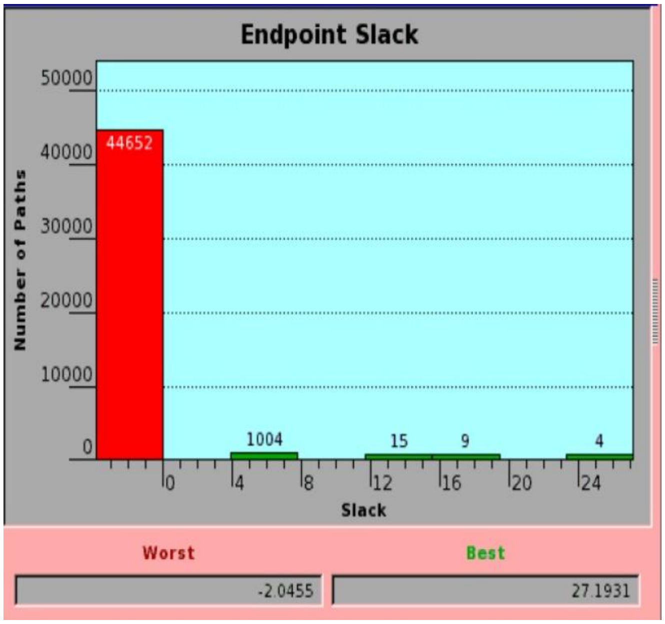
4.3.1.2. Timing Results



Worst -3.20945

Best 49.7302

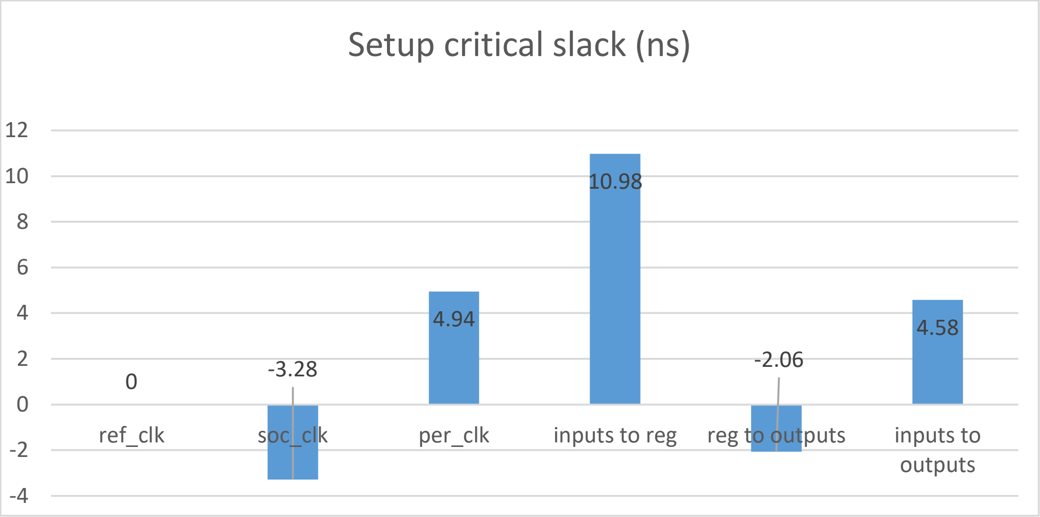
Figure 43: Setup Time Histogram after Placement (Hierarchical Flow)

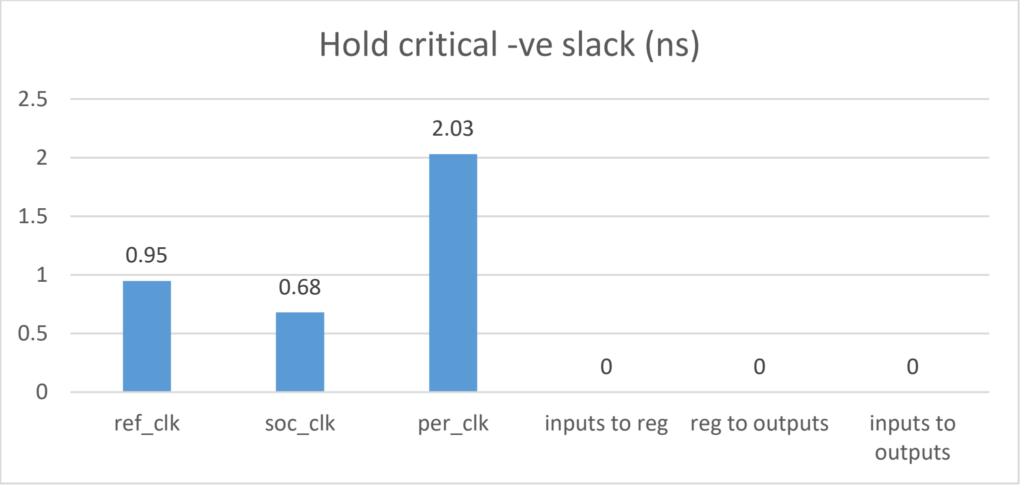


Worst -2.0455

Best 27.1931

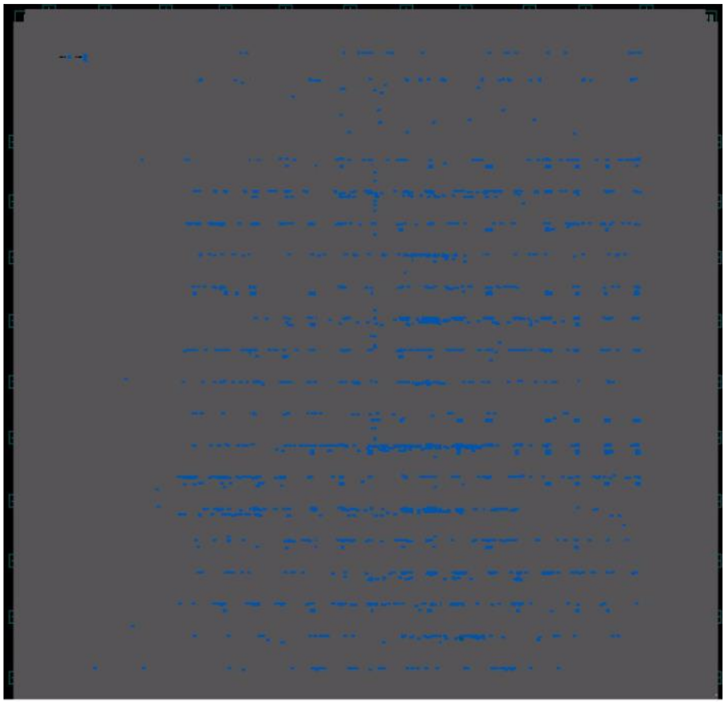
Figure 44: Hold Time Histogram after Placement (Hierarchical Flow)

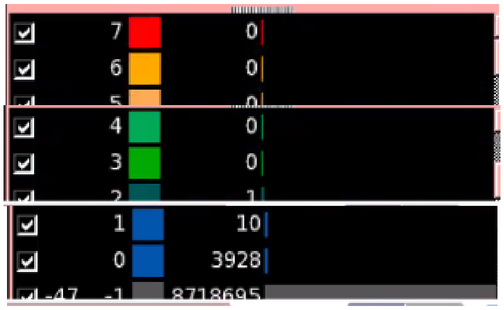

Setup critical slack (ns)
12
10
8
6
4
2
0
-2
-4
0
-3.28
4.94
10.98
-2.06
4.58
ref_clk
soc_clk
per_clk
inputs to reg
reg to outputs
inputs to outputs


Hold critical -ve slack (ns)
2.5
2
1.5
1
0.5
0
0.95
0.68
2.03
0
0
0
ref_clk
soc_clk
per_clk
inputs to reg
reg to outputs
inputs to outputs

4.3.3. Topographical Flow

4.3.1.1. Congestion Map:





☑		7	🟥	0
☑		6	🟧	0
☑		5	🟧	0
☑		4	🟩	0
☑		3	🟩	0
☑		2	🟩	1
☑		1	🟦	10
☑		0	🟦	3928
☑	-47	-1	⬛	8718695

4.3.1.2. Timing Results

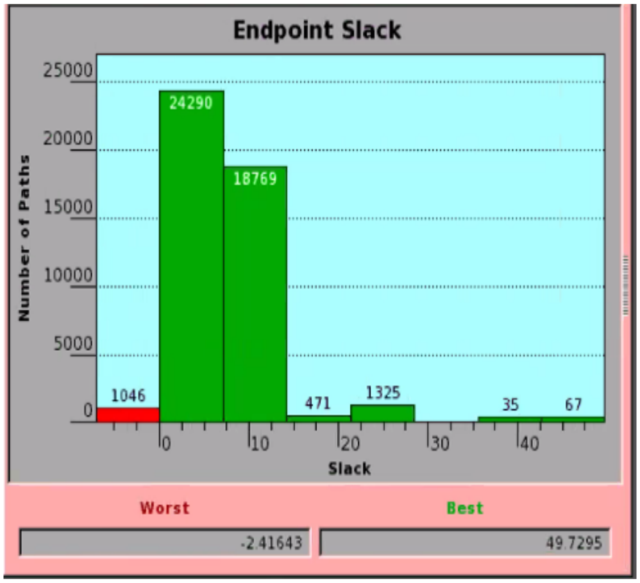


Figure 45: Setup Time Histogram after Placement (Topographical Flow)

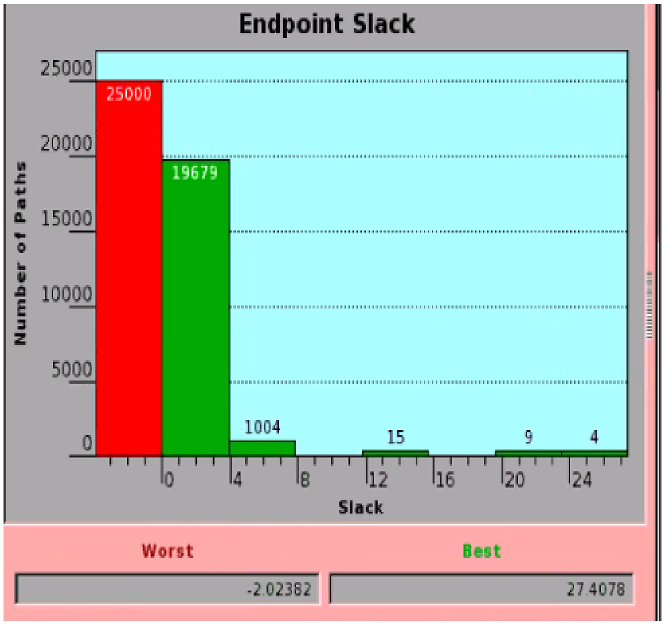
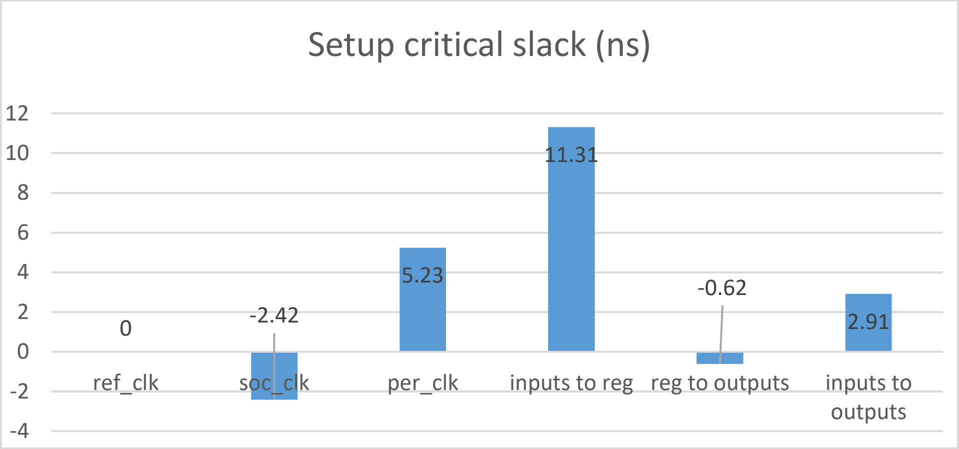
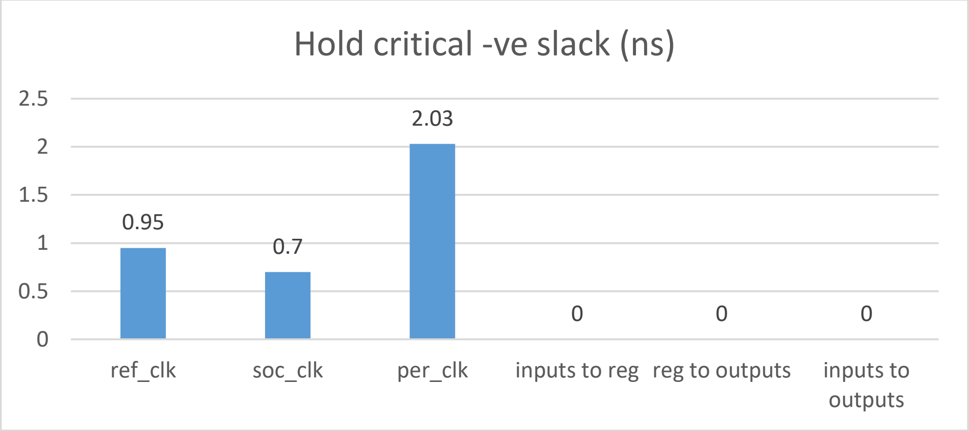


Figure 46: Hold Time Histogram after Placement (Topographical Flow)


Setup critical slack (ns)
12
10
8
6
4
2
0
-2
-4
0
-2.42
5.23
11.31
-0.62
2.91
ref_clk
soc_clk
per_clk
inputs to reg
reg to outputs
inputs to outputs


Hold critical -ve slack (ns)
2.5
2
1.5
1
0.5
0
0.95
0.7
2.03
0
0
0
ref_clk
soc_clk
per_clk
inputs to reg
reg to outputs
inputs to outputs

4.4 Clock Tree Synthesis (CTS)

4.4.1. Flat Flow

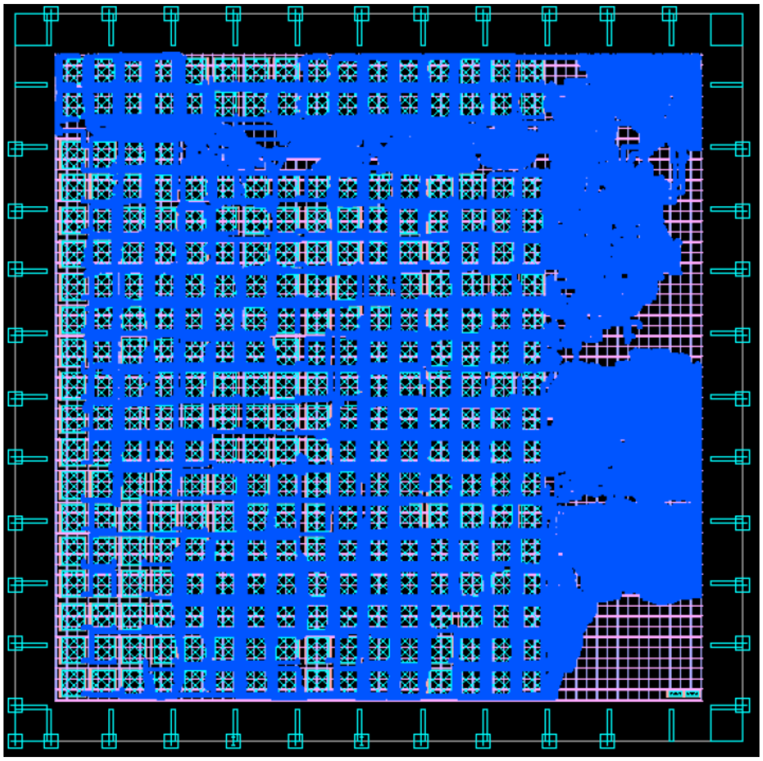


Figure 47: SoC Domain Clock Tree (Flat Flow)

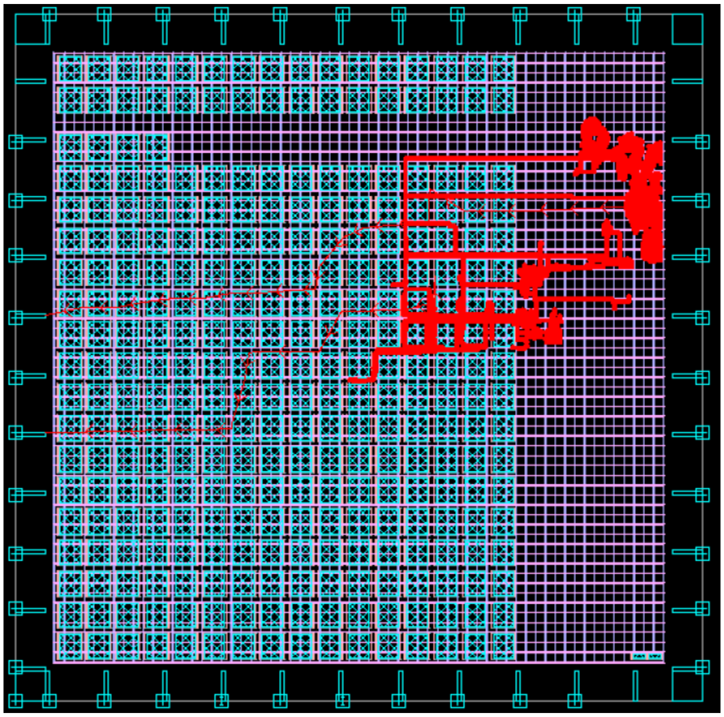


Figure 48: Peripheral Domain Clock Tree (Flat Flow)

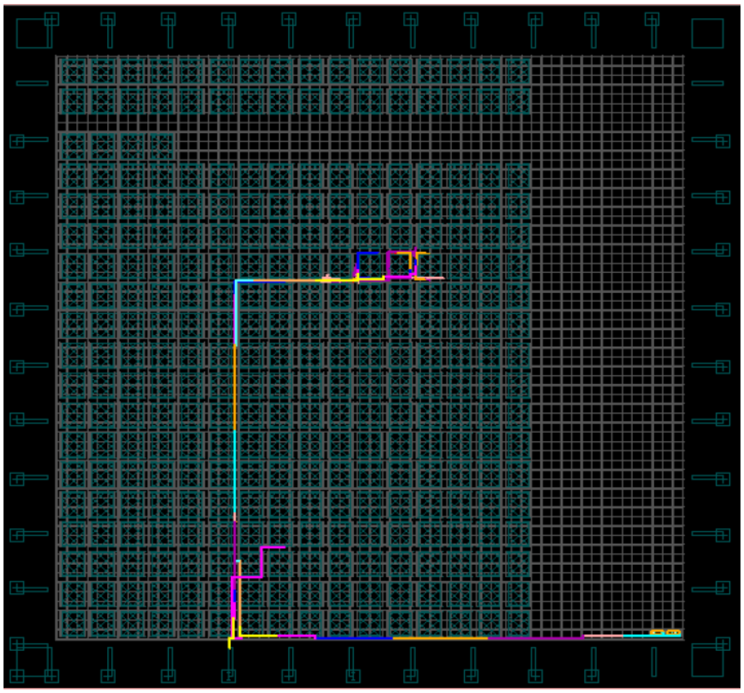


Figure 49: Reference Clock Tree (Flat Flow)

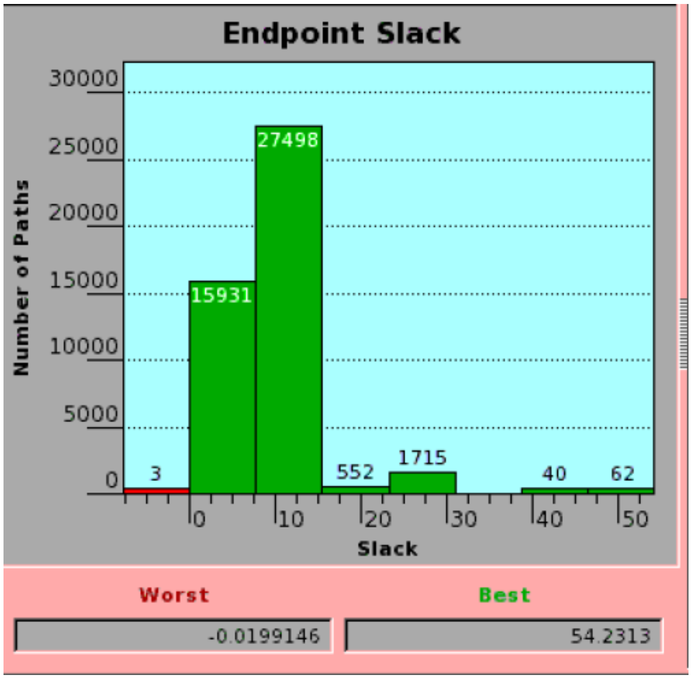


Figure 50: Setup Time Histogram after CTS (Flat Flow)

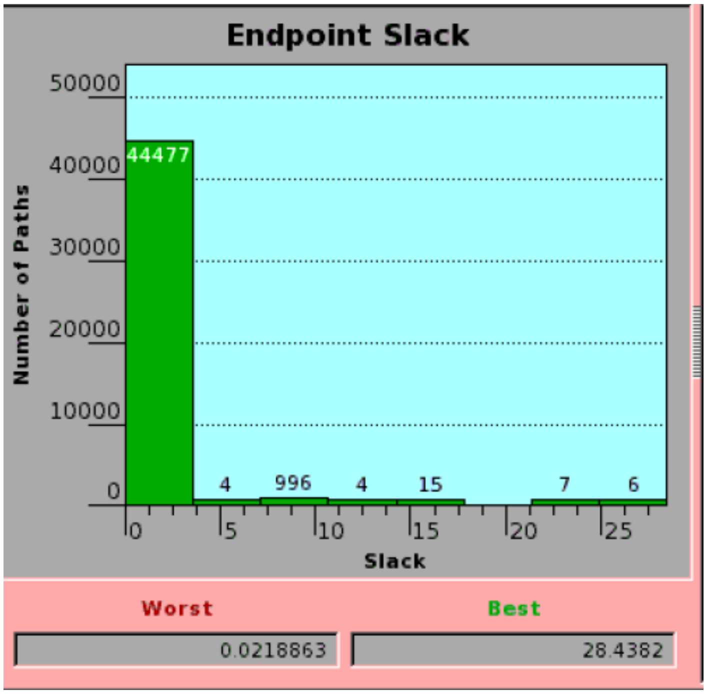


Figure 51: Hold Time Histogram after CTS (Flat Flow)

4.4.2. Hierarchical Flow

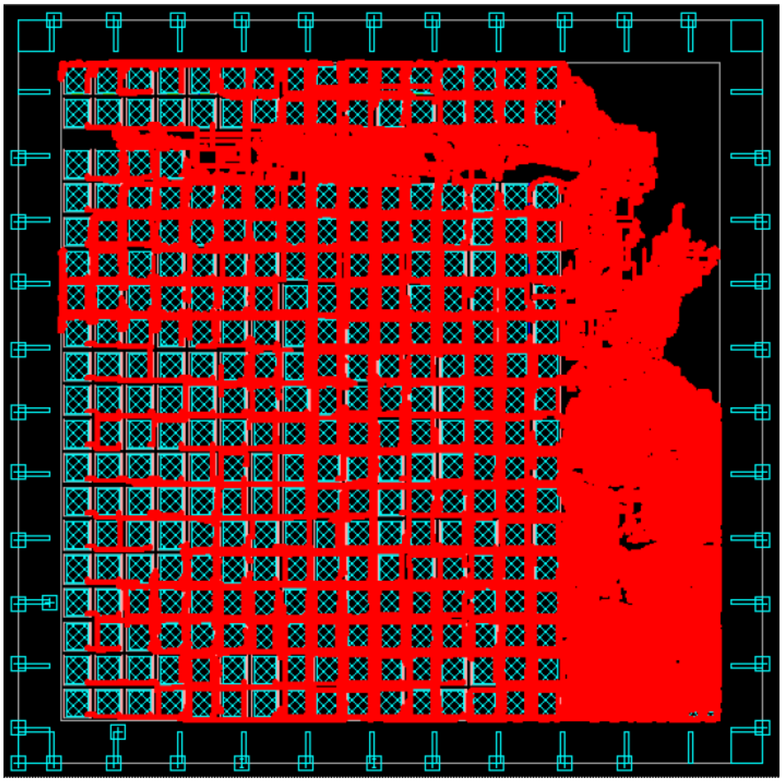


Figure 52: SoC Domain Clock Tree (Hierarchical Flow)

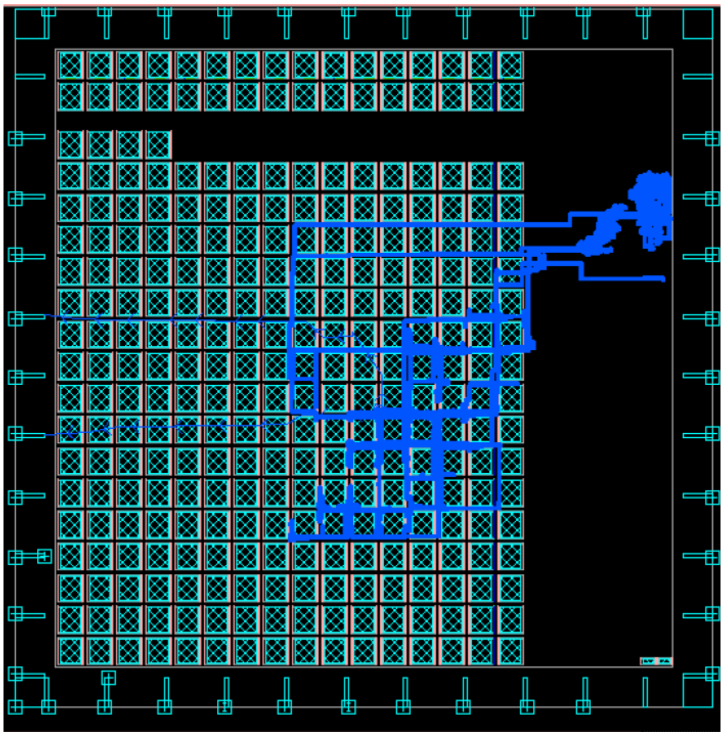


Figure 53: Peripheral Domain Clock Tree (Hierarchical Flow)

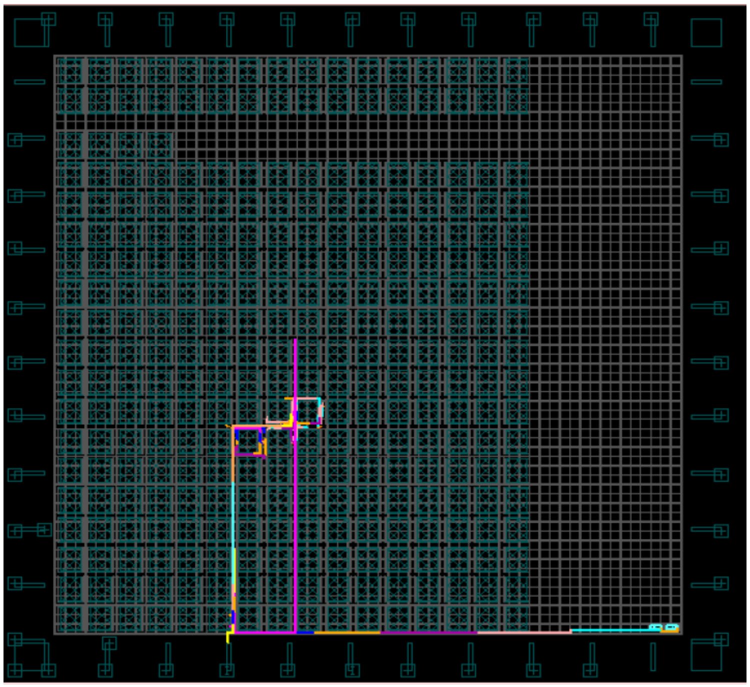


Figure 54: Reference Clock Tree (Hierarchical Flow)

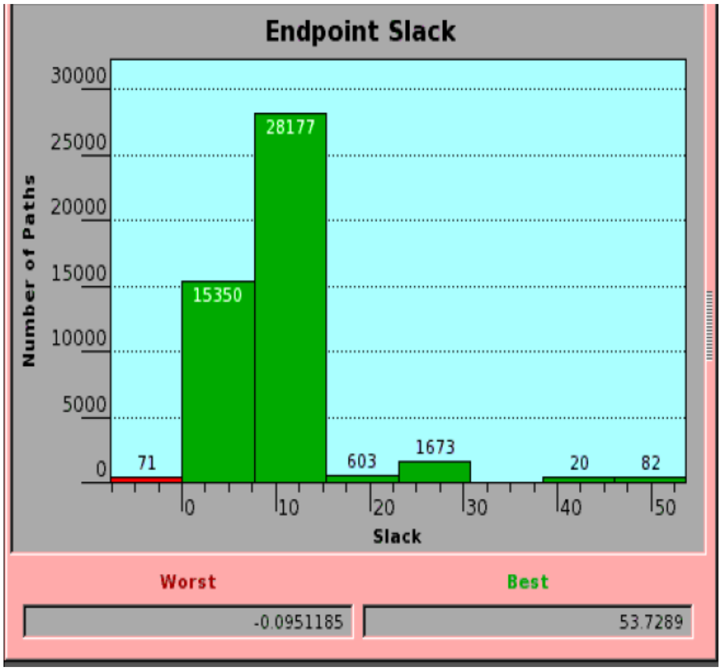


Figure 55: Setup Time Histogram after CTS (Hierarchical Flow)

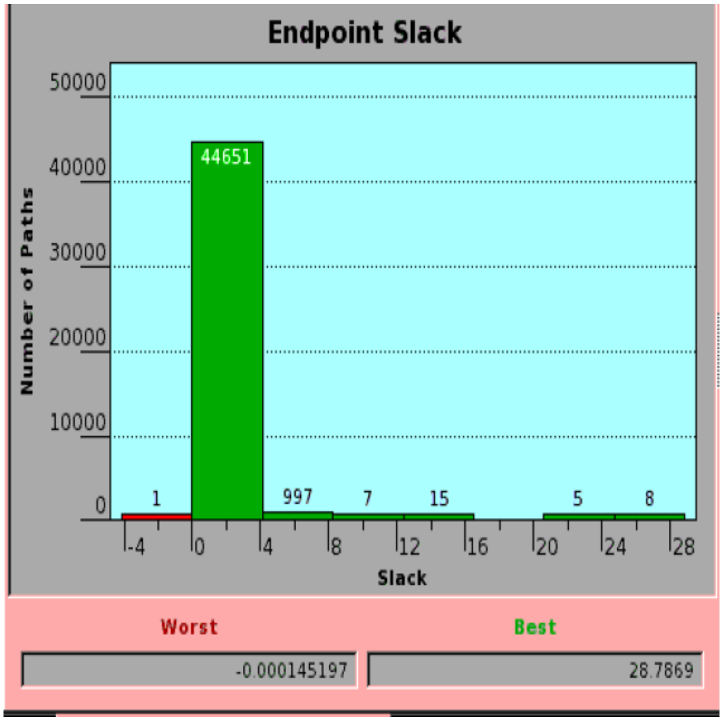


Figure 56: Hold Time Histogram after CTS (Hierarchical Flow)

4.4.3. Topographical Flow

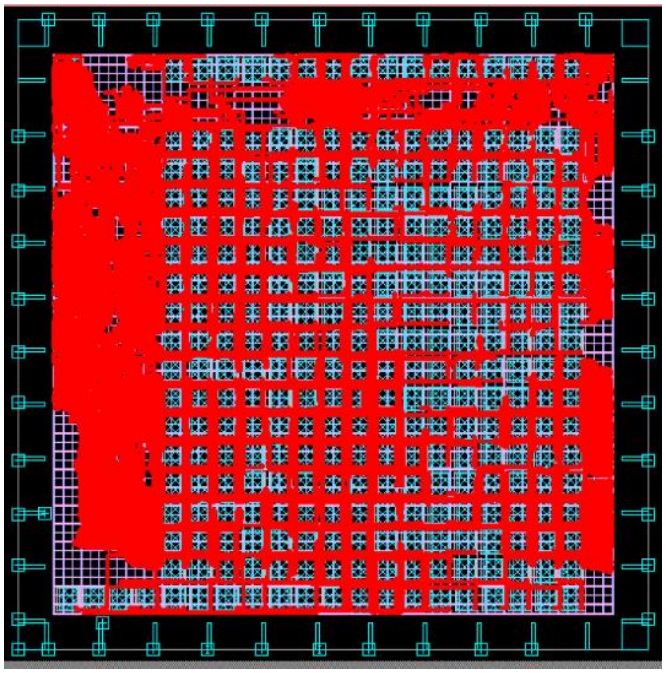


Figure 57: SoC Clock Tree (Topographical Flow)

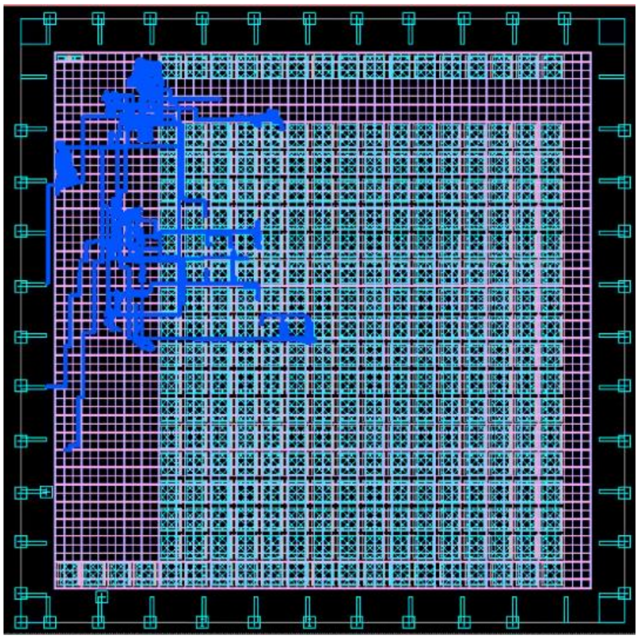


Figure 58: Peripheral Clock Tree (Topographical Flow)

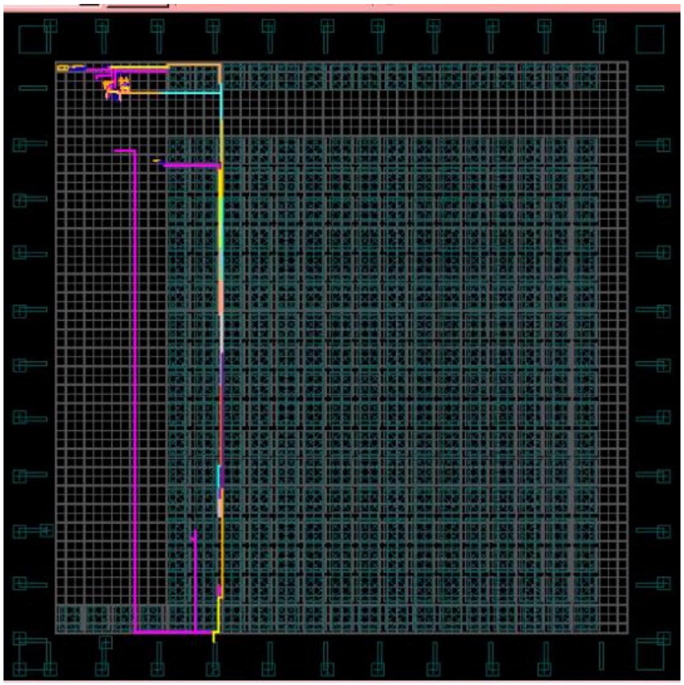


Figure 59: Reference Clock Tree (Topographical Flow)

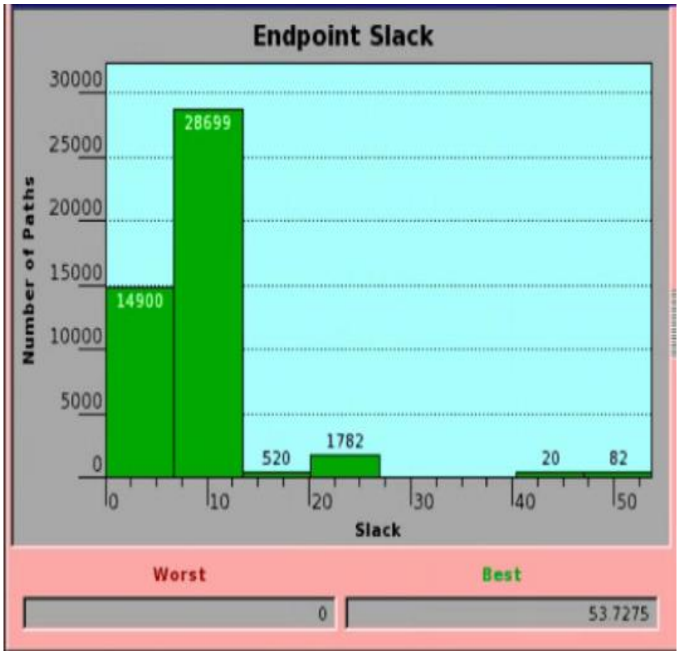


Figure 60: Setup Time Histogram after CTS (Topographical Flow)

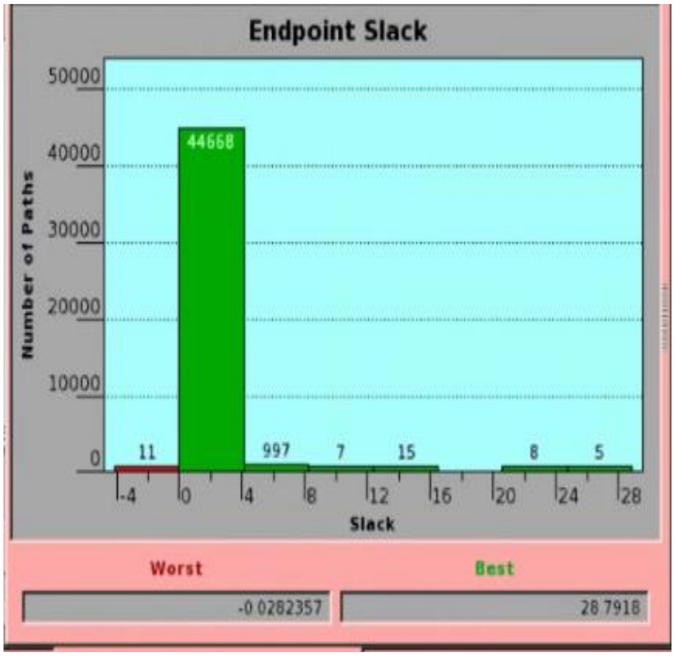


Figure 61: Hold Time Histogram after CTS (Topographical Flow)

4. Routing and PnR Final Results

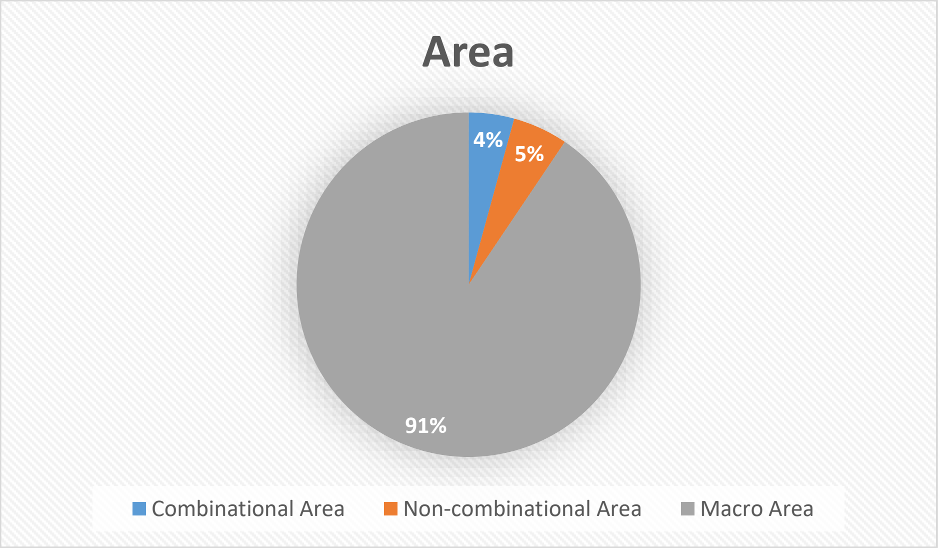
4.4. Area Results

4.4.1. Flat Flow

Table 23 : Flat Flow Area Results

	After Removing Hierarchy
Total Cell Area	16349625.9 µm^2
Net Area	1351556.03 µm^2
Total Design Area	17701181.9 µm^2

Combinational Area	697474.8 µm^2
Non-combinational Area	846293 µm^2
Macro Area	14805857.5 µm^2

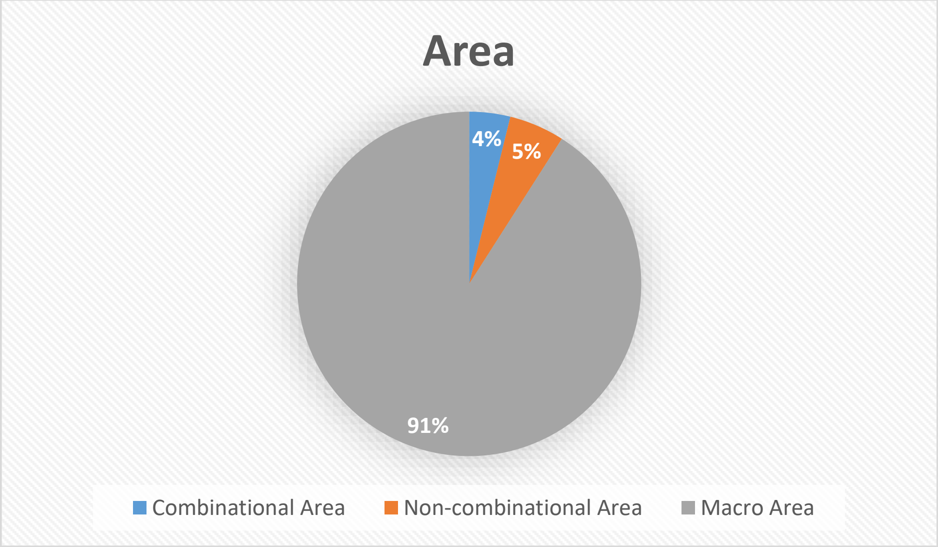


4.5.1.2. Hierarchical Flow

Table 24: Hierarchical Flow Final Area Results

	After Removing Hierarchy
Total Cell Area	16281037.7 µm^2
Net Area	1410248.6 µm^2
Total Design Area	17691286.3 µm^2

Combinational Area	627477.5 µm^2
Non-combinational Area	847702.8 µm^2
Macro Area	14805857 µm^2

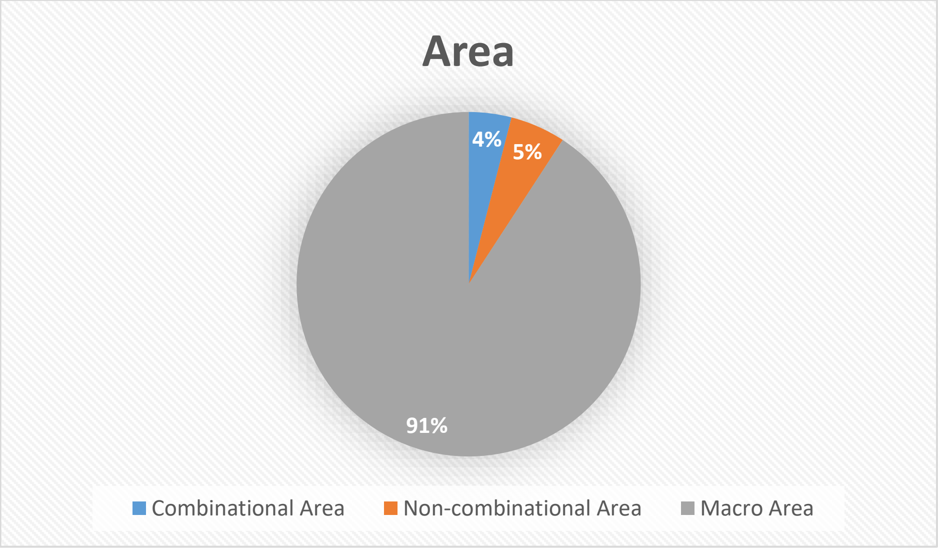


4.5.1.3. Topographical Flow

Table 25 : Topographical Flow Final Area Results

	After Removing Hierarchy
Total Cell Area	16307357.7 µm^2
Net Area	1355910.3 µm^2
Total Design Area	17663268 µm^2

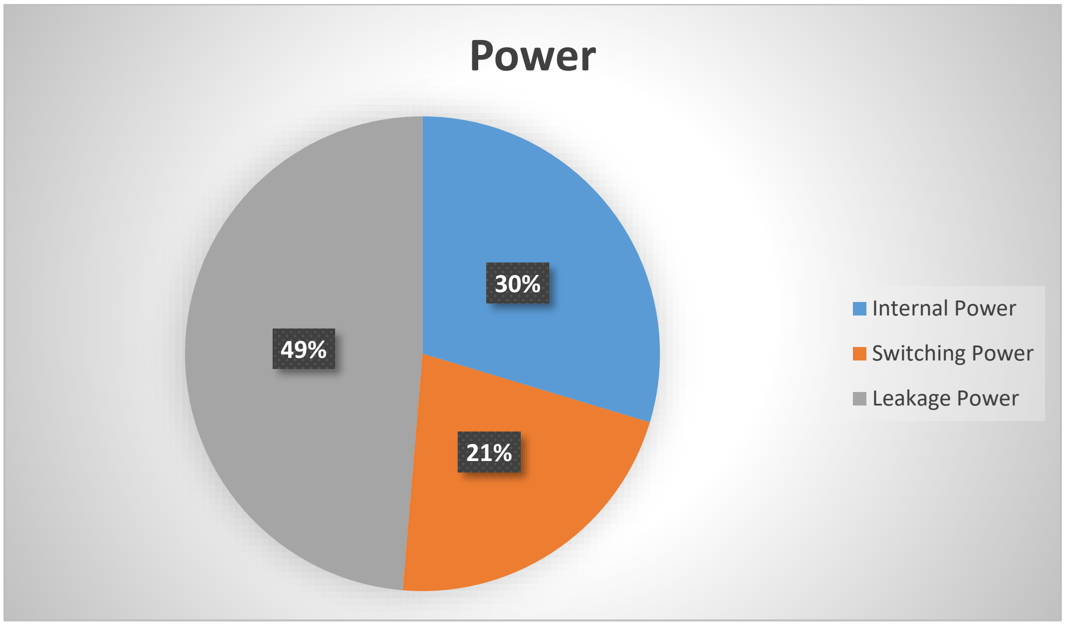
Combinational Area	653124 µm^2
Non-combinational Area	848375 µm^2
Macro Area	14805857.5 µm^2



Power Results

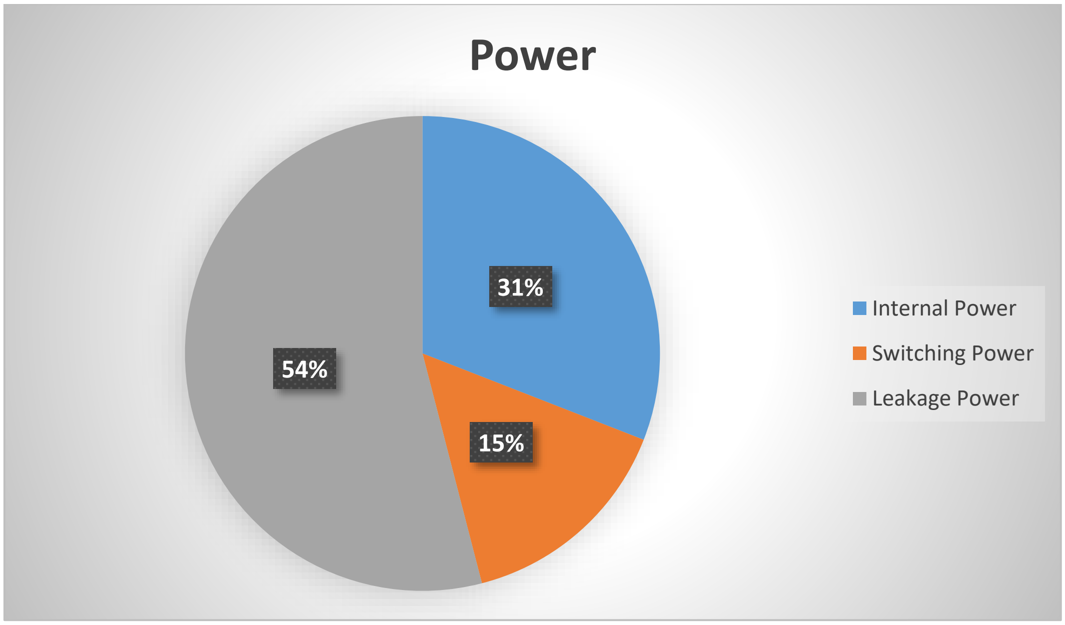
4.4.2. Flat Flow

Internal Power	7.36 mW
Switching Power	5.61 mW
Leakage Power	11.75 mW
Total Power	24.73 mW



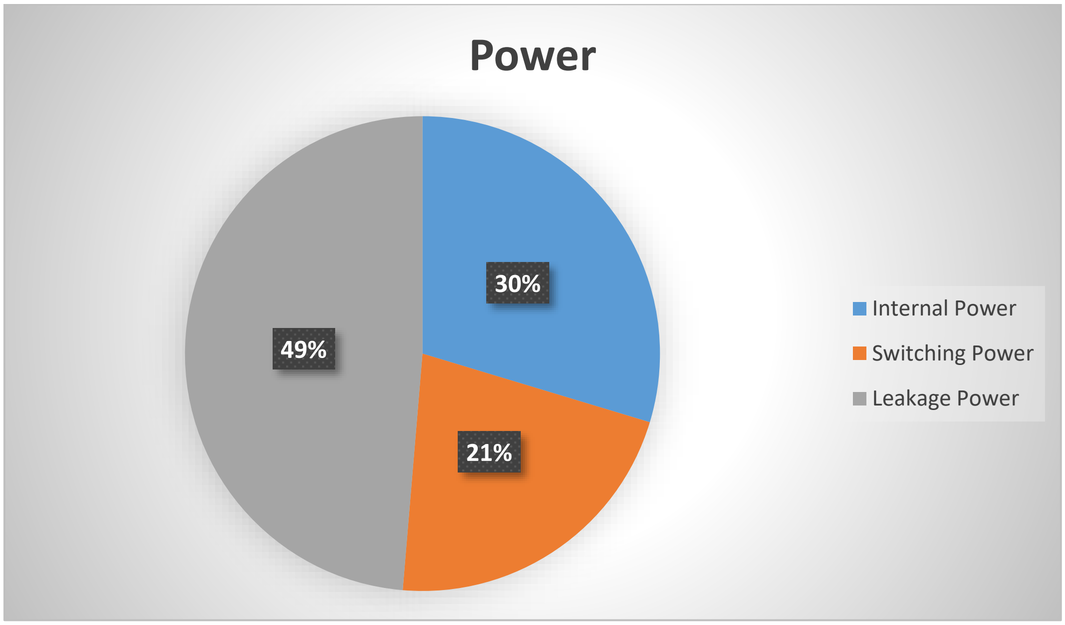
4.5.3.2. Hierarchical Flow

Internal Power	6.68 mW
Switching Power	3.25 mW
Leakage Power	11.67 mW
Total Power	21.6 mW



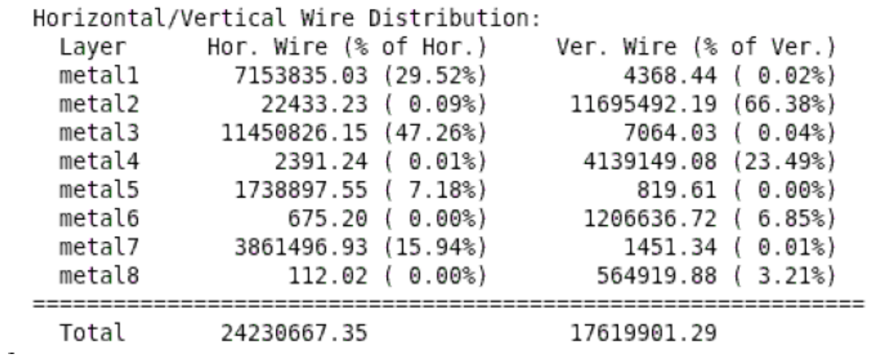
4.5.3.3. Topographical Flow

Internal Power	7.15 mW
Switching Power	5.22 mW
Leakage Power	11.73 mW
Total Power	24.103 mW



Wire Statistics

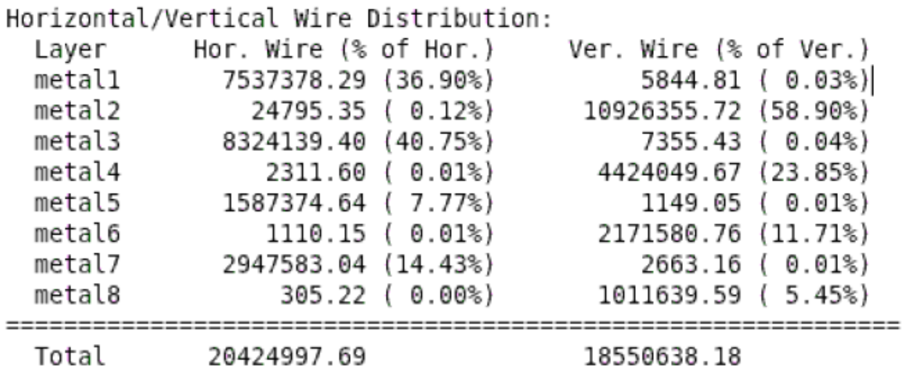
4.4.3. Flat Flow



```
Horizontal/Vertical Wire Distribution:
  Layer      Hor. Wire (% of Hor.)     Ver. Wire (% of Ver.)
  metal1       7153835.03 (29.52%)          4368.44 ( 0.02%)
  metal2         22433.23 ( 0.09%)      11695492.19 (66.38%)
  metal3      11450826.15 (47.26%)          7064.03 ( 0.04%)
  metal4          2391.24 ( 0.01%)       4139149.08 (23.49%)
  metal5       1738897.55 ( 7.18%)           819.61 ( 0.00%)
  metal6           675.20 ( 0.00%)       1206636.72 ( 6.85%)
  metal7       3861496.93 (15.94%)          1451.34 ( 0.01%)
  metal8           112.02 ( 0.00%)        564919.88 ( 3.21%)
=============================================================
  Total       24230667.35               17619901.29
```

Figure 62:Horizontal/Vertical Wire Distribution (Flat Flow)

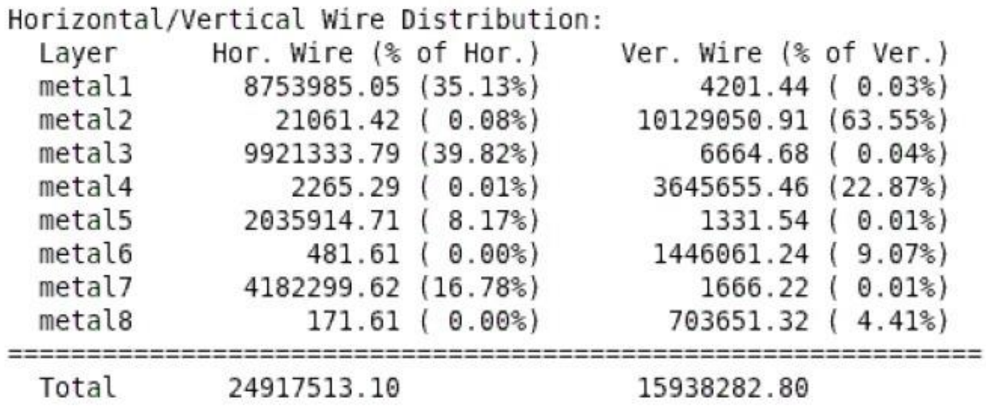
4.5.3.2. Hierarchical Flow



```
Horizontal/Vertical Wire Distribution:
  Layer      Hor. Wire (% of Hor.)     Ver. Wire (% of Ver.)
  metal1       7537378.29 (36.90%)          5844.81 ( 0.03%)
  metal2         24795.35 ( 0.12%)      10926355.72 (58.90%)
  metal3       8324139.40 (40.75%)          7355.43 ( 0.04%)
  metal4          2311.60 ( 0.01%)       4424049.67 (23.85%)
  metal5       1587374.64 ( 7.77%)          1149.05 ( 0.01%)
  metal6          1110.15 ( 0.01%)       2171580.76 (11.71%)
  metal7       2947583.04 (14.43%)          2663.16 ( 0.01%)
  metal8           305.22 ( 0.00%)       1011639.59 ( 5.45%)
=============================================================
  Total       20424997.69               18550638.18
```

Figure 63: Horizontal/Vertical Wire Distribution (Hierarchical Flow)

4.5.3.3. Topographical Flow



```
Horizontal/Vertical Wire Distribution:
  Layer      Hor. Wire (% of Hor.)      Ver. Wire (% of Ver.)
  metal1     8753985.05 (35.13%)          4201.44 ( 0.03%)
  metal2       21061.42 ( 0.08%)      10129050.91 (63.55%)
  metal3     9921333.79 (39.82%)          6664.68 ( 0.04%)
  metal4        2265.29 ( 0.01%)       3645655.46 (22.87%)
  metal5     2035914.71 ( 8.17%)          1331.54 ( 0.01%)
  metal6         481.61 ( 0.00%)       1446061.24 ( 9.07%)
  metal7     4182299.62 (16.78%)          1666.22 ( 0.01%)
  metal8         171.61 ( 0.00%)        703651.32 ( 4.41%)
===============================================================
  Total     24917513.10               15938282.80
```

Figure 64: Horizontal/Vertical Wire Distribution (Topographical Flow)

4.3. TimingResults

Flat Flow

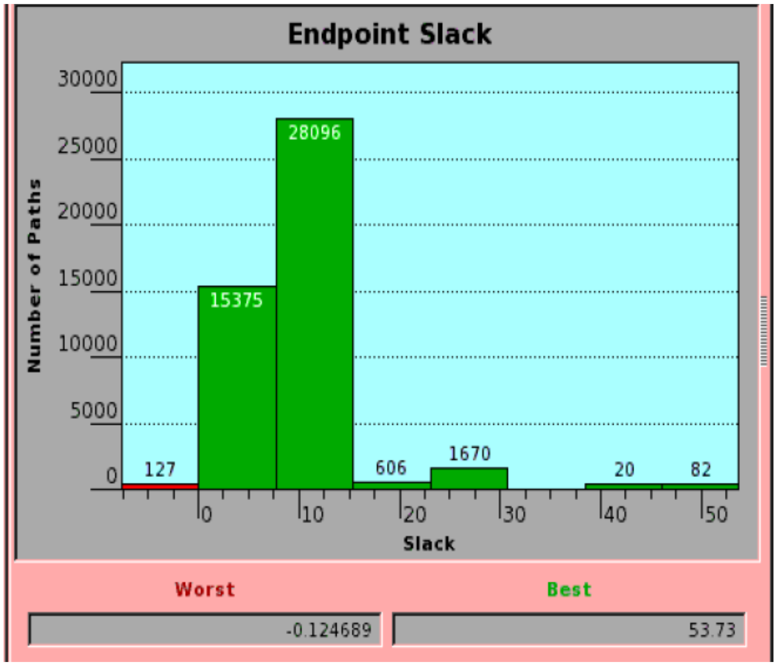


Figure 65: Setup Time Histogram after PnR (Flat Flow)

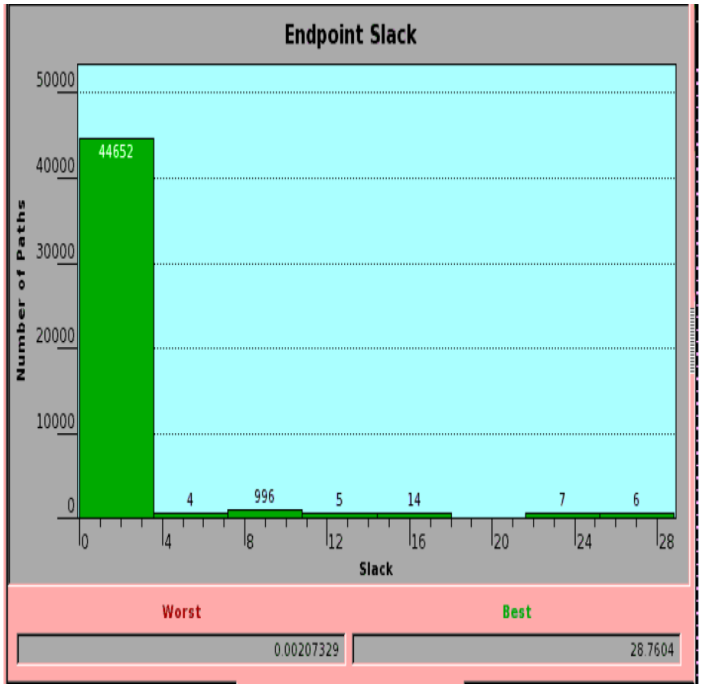


Figure 66: Hold Time Histogram after PnR (Flat Flow)

Hierarchical Flow

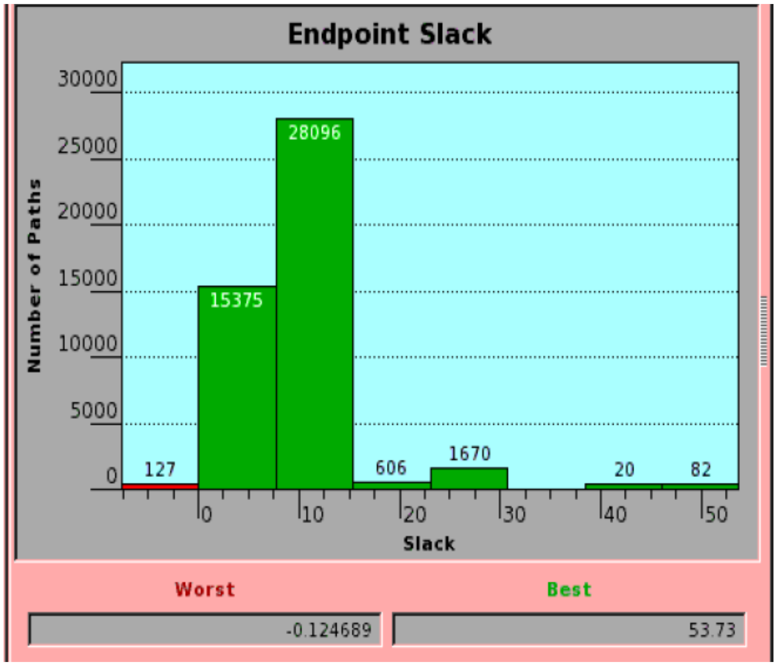


Figure 67: Setup Time Histogram after PnR (Hierarchical Flow)

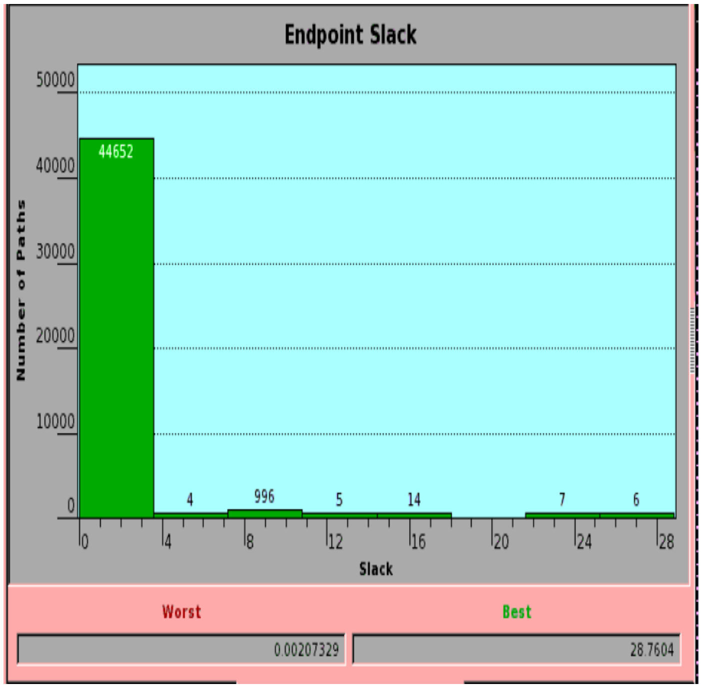


Figure 68: Hold Time Histogram after PnR (Hierarchical Flow)

Topographical Flow

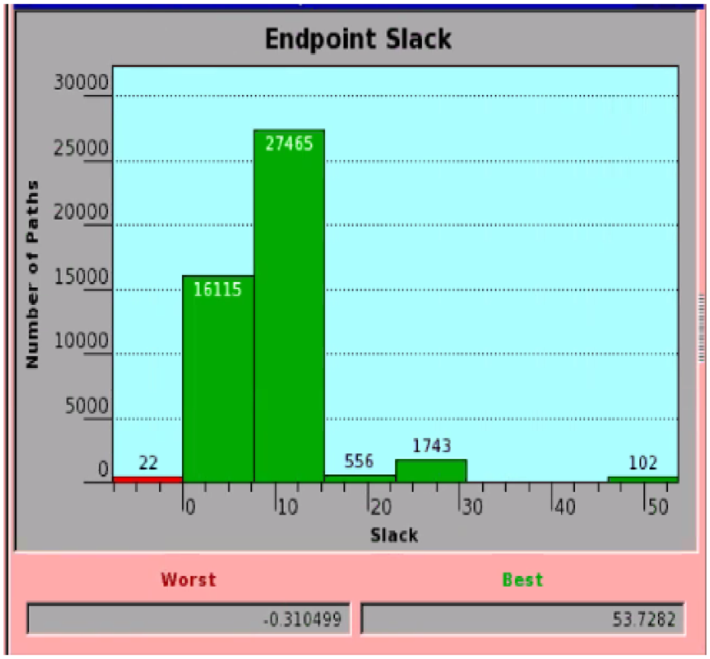


Figure 69: Setup Time Histogram after PnR (Topographical Flow)

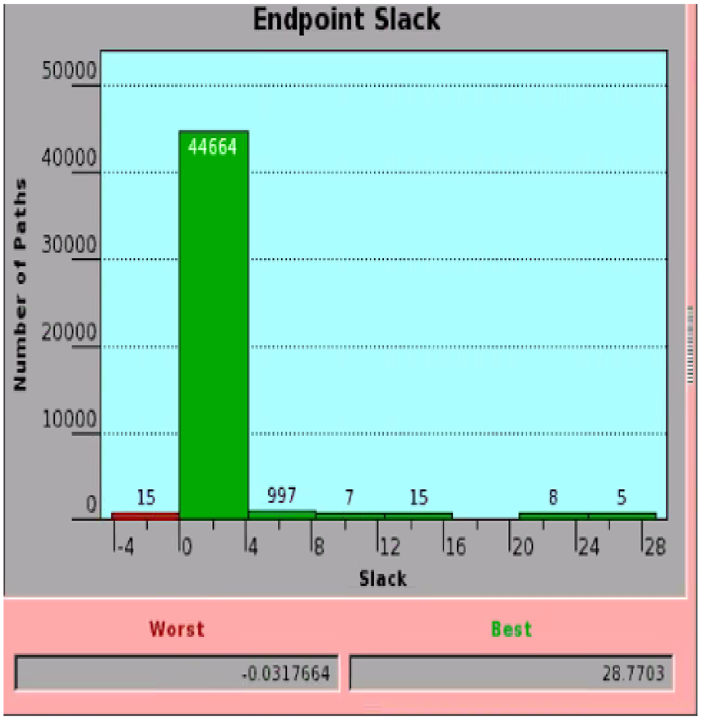


Figure 70: Hold Time Histogram after PnR (Topographical Flow)

5. Conclusion and Future Work

Here, we presented the core implementation results, mainly the logic synthesis and the PnR, the work presented here could be complemented in the future by carrying out standard verification procedures using Formality for formal verification, as explained in Appendix A, and doing Layout versus Schematic (LVS) simulations. Furthermore, the whole PULPissimo core could be used to instantiate multicore IoT processors like OpenPulp. The design could also be integrated with a hardware processing engine in a heterogeneous computing architecture for computationally intensive tasks.

References:

[1] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti and L. Benini, "Quentin: an Ultra-Low-Power PULPissimo SoC in 22nm FDX," 2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), 2018, pp. 1-3, doi: 10.1109/S3S.2018.8640145.

[2] PULP Platform (2021). “PULPissimo Datasheet”.

[3] Pasquale Davide Schiavone (2019), Understanding and working with PULP. Online presentation. https://pulp-platform.org/docs/riscv_workshop_zurich/schiavone_wosh2019_tutorial.pdf

[4] Synopsys, ”Design Compiler Guide”, Synopsys, Inc, September 2019

[5] Synopsys, ”IC Compiler Guide”, Synopsys, Inc, September 2019

[6] Synopsys, Digital Standard Cell Library SAED_EDK32/28_CORE DATA-BOOK.

[7] Mehta, A.B. (2018). Functional Verification: Challenges and Solutions. In: ASIC/SoC Functional Design Verification. Springer, Cham. https://doi.org/10.1007/978-3-319-59418-7_2

[8] Sridhar Gangadharan, Sanjay Churiwala (2013). Constraining Designs for Synthesis and Timing Analysis A Practical Guide to Synopsys Design Constraints. Springer, https://doi.org/10.1007/978-1-4614-3269-2

[9] Khosrow Golshan (2007). Physical Design Essentials, An ASIC Design Implementation Perspective. Springer, https://doi.org/10.1007/978-0-387-46115-1

[10] Abdelrahman Eltokhi and Si-vision team (2019), ASIC Physical Design, PowerPoint slides.

Appendix: ASIC flow

1. Introduction:

ASIC stands for Application Specific Integrated Circuits, ASICs are Integrated circuits customized and manufactured as an implementation of a specific an algorithm rather than implementing it as a function on a general purpose processor or programming it on an FPGA. And the advantages of ASIC over FPGA and general purpose microcontroller are summarized as following:

1) High utilization as we use only wanted cells.
2) High performance but more complex.
3) Low cost of fabrication in the massive production.

All ASIC flows consists of front-end and back-end parts, and in this thesis the flat, hierarchal and topographical flows will be illustrated in details in the following sections.

2. Front-end part:

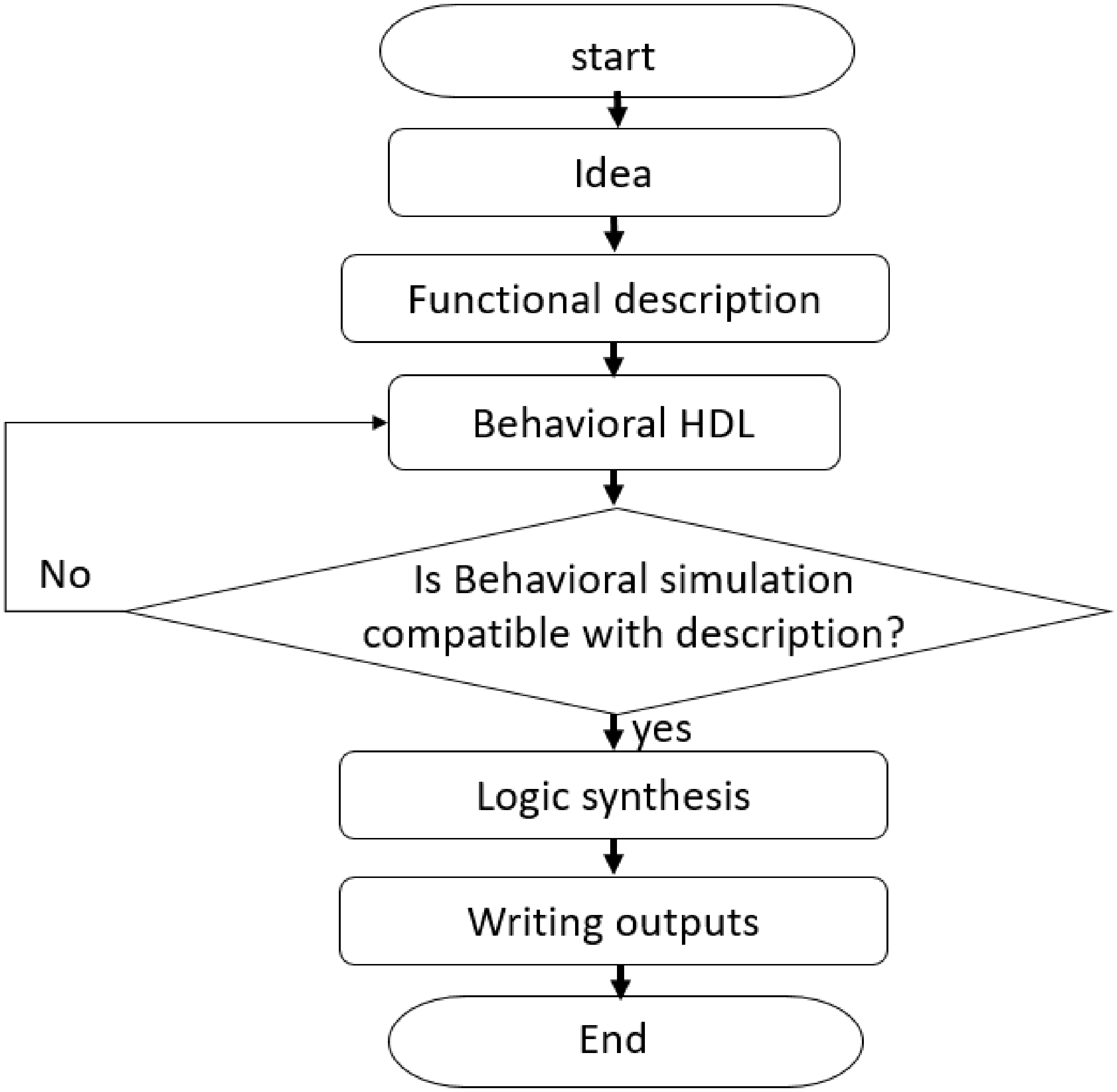


Figure 71: flow chart of the front-end part in ASIC

2.1 Idea or specific algorithm:

The front-end part is begun by an idea, a specific algorithm or a function that we want to implement by ASIC, hence specifying the inputs and outputs properties and all others specifications, like a required operating frequency, area, leakage power, maximum static and dynamic power consumption, etc. And then starting to write a RTL code that describes and performs the required function.

2.2 Behavioral HDL simulation:

After finishes the RTL code, the RTL simulation is performed to ensure that the RTL code work properly and performs the required function and it is cyclic and bit accurate (which means that produces the right bit value in the right cycle). If the RTL code isn’t satisfied the previously mentioned conditions, it must be modified.

2.3 Logic synthesis step:

The logic synthesis step concerns to convert the RTL into netlist. And it can be used with a trial and error method is used when it is wanted to know the best constraints that the RTL code can be work properly with a specific technology library. The logic synthesis step with a trial and error method by the design compiler (DC), as it is used in this thesis, can be illustrated in the following flow chart:

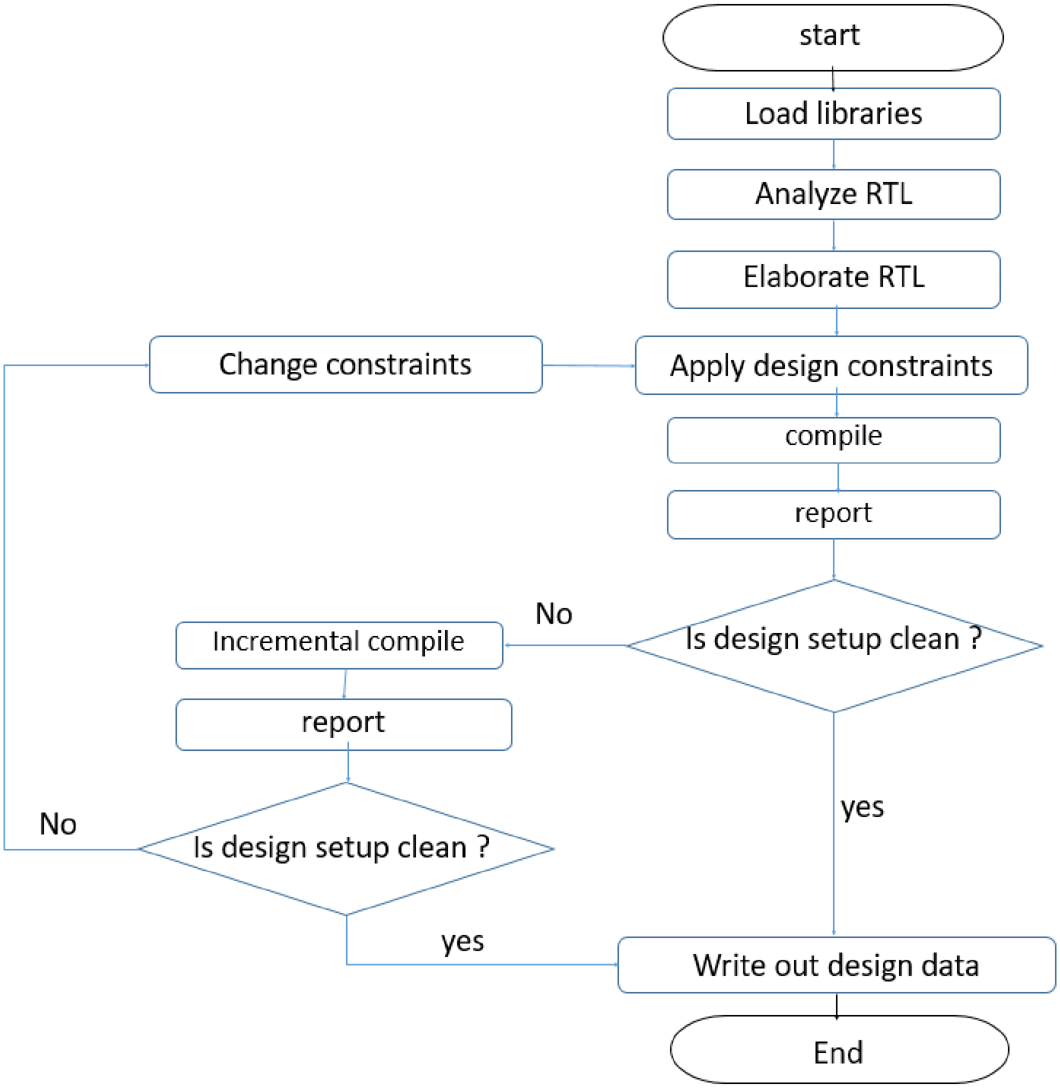


Figure 72: flow chart of the logic synthesis step with a trial and error method

2.3.1 Libraries loading:

The Load libraries step which consists of the following:

1) Technology libraries that mainly contain:

- Standard cells and input-output (IO) pads information. The standard cells and input-output (IO) pads are used to map/compile the GTECH cells in the unmapped gate-level netlist (where GTECHs are technology-independent cells and don't have any timing, area and power information) to standard cells to create a mapped gate-level netlist.
- Operating conditions which is basically the process, voltage and temperature variations of the used technology library.
- Units attributes of each quantity.
- Wire load models (WLM) which is used to estimate the length of each net based on its fan-out number information, and then multiplies it in the per-unit value of the resistance and capacitance.

2) Link libraries that contains timing information of all standard cells, IO pads and used macros to be used in mapping and reporting.

2.3.2 Analyze and elaborate steps:

The analyze and elaborate steps are used to read the RTL coed in synthesis tool (design compiler), where

- The analyze step: reads the RTL codes, informs the designer if there are any syntax errors or non-synthesizable statements and converts the RTL code into intermediate binary formats to be read by the elaborate step.
- The elaborate step: translates the intermediate binary formats into GTECH (technology independent gates) gate-level netlist.

We could use instead of these two commands "read_file" command which makes analyze and elaborate in single step, "link" command must follow "read_file" command.

2.3.3 Constraints applying:

The applying of design constrains in the DC tool is used to define the following matters:

1) timing constraints which can be summarized in the following points:
    - Clock period.
    - Source latency.
    - Input and output delay constraints.
    - Uncertainty of the clock which is the summation of the estimated skew of the clock, jitter and margin.
    - False (timing paths that wouldn’t happen as they logically cannot occur) and multicycle (timing paths that need more than clock cycle to be executed) paths.
    - Path groups, their weight and critical range (to solve the sub-critical timing paths).

2) Area goal.

3) Leakage Power and power intent of the design.

4) Electrical DRC constraints which are maximum input transition, maximum load capacitance and maximum fan-out number of each net in the design.

The timing paths are categorized in 4 types of paths:

1) Register-to- register path: starts from a clock pin of a launch sequential element and ends at a clock pin of a capture sequential element.

2) Input-to- register path: starts from an input port of the design and ends at a clock pin of a capture sequential element.

3) Register-to-output path: starts from clock pin of a launch sequential element and ends at output port of the design.

4) Input-to-output path: starts from input port of a design and ends at output port of the design.

The ‘create_clock’ command is used as a DC command to define a clock domain. We pass the period and the port at which the clock object will be attached to.

The timing paths of input-to-register, register-to-output and input-to-output types are constrained by defining another constraint which are input and output delay constraints and that is done by “set_input_delay” and “set_output_delay” commands.

2.3.4 Compilation step:

The compilation step is mainly convert the GTECH gate-level netlist into mapped gate-level netlist. And it is done in the DC by the “compile_ultra” command for more optimization on timing and area.

2.3.5 Reporting:

The generating reports are for:

1) Area

2) All mapped standard cells

3) Quality of results (QoRs)

4) Clock gating

5) Setup timing reports (maximum delay type).

The design mustn’t have any setup timing violation to writing the outputs of the logic synthesis, otherwise the designer must perform incremental optimization on the mapped gate-level netlist and report to check whether there are any setup timing violations or not. And that what is means by setup-violation clean design.

Whereas the design is allowed to have any number of hold timing violations in this step, as they will be solved by the clock tree synthesis (CTS) step in the place and route (PnR) flow.

The reporting of area is done by “report_area” command, the setup timing reporting is done by the “report_timing –delay_type max”.

If the reports appear negative slacks we should make another incremental compile, and get another reports to see if the slacks are fixed or not, if still negative slacks appear change the constrains and make another compile then report until there is no negative slacks (the design is setup clean).

2.3.6 Writing the outputs:

The outputs are mainly 2 files:

1) the gate-level netlist written in a HDL and the constraints written a specific format to be easily read by the layout tool. In case using design compiler to perform the logic synthesis step, the outputs will be gate-level netlist written in a Synopsys internal database called ddc.
2) The constraints is written in a format called Synopsys design constraints (SDC).

3. Back-end part:

The back-end part in the ASIC flow is called PnR which stands for place and route, but in fact it contains more steps before and after the place and route steps, here is a flow chart that describes that all main steps of the PnR flow in short:

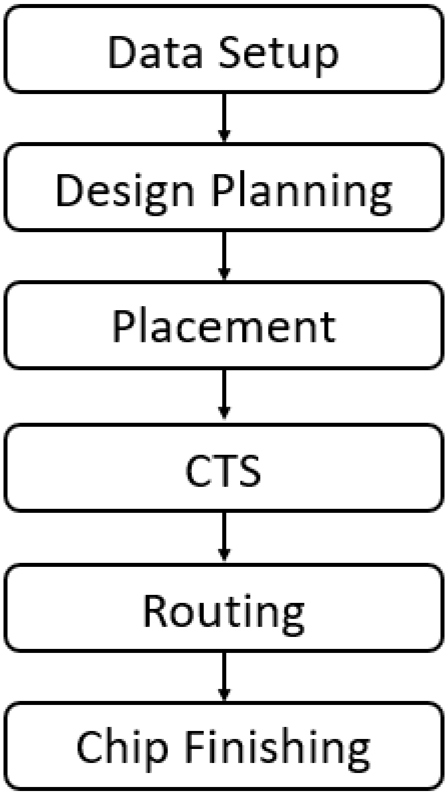


Figure 73: summarized flow chart of the PnR flow

The below section describes each step in more details.

3.1 Data setup:

In the data setup step, the designer gives specific inputs to the PnR tool, which is the IC compiler in this thesis, to prepare it to the nest steps in the flow. These inputs can be summarized as following:

3.1.1 Target library:

1) Standard cells and input-output (IO) pads information, like timing, area, leakage power, Boolean function and information about each pin in each cell.

2) Operating conditions.

3) Units attributes of each quantity.

4) Wire load models (WLM).

And it is has .db and .lib extension, where the .lib file contains the previously mentioned information in ASCII format to be easily read by the designer and the .db file contains these information in binary format to be easily read by the IC compiler. And it the .lib file is the only one existed the IC compiler will convert it into binary format (.db extension). And its directory is saved in reserved variable called “target_library” in the IC compiler tool.

3.1.2 Link library:

Link libraries contain timing information of all standard cells, IO pads and used macros to be used in analyzing and reporting the standard cells, IO pads and used macros that are considered as parts of the different timing paths in the design. And its directory is saved in reserved variable called “link_library” in the IC compiler tool.

3.1.3 Physical library

The physical libraries contains physical information about standard cells, IO pads and used macros, and these information like:

1) Abstraction view of the layout, which is called FRAM view.

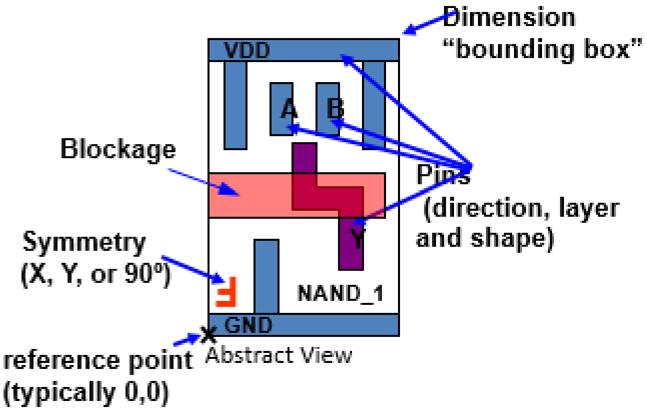


Figure 74: example on FRAM view of NAND logic cell

2) Layers that is used in the layout to be considered as obstacles in the routing step and prevented any short can happened during placement or routing steps.

3) Pins layer, its size and its locations.

4) Units attributes.

5) Defines placement site.

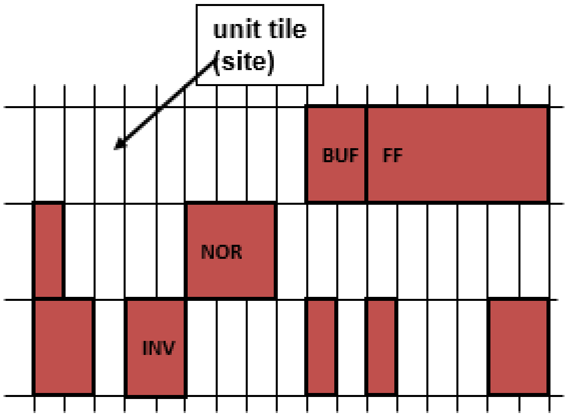


Figure 75: example on the site (unit tile).

6) Technology file which contains the geometrical design rule checks (DRCs) of each layer in the used technology library. And the extension of that file is “.tf”.

These information is placed in internal database of the Synopsys company called Milkyway database and it is created by command called “create_mw_lib” which mainly takes the FRAM view of each standard cells, IO pads and used macros, and the technology file. And all the previously mentioned files are provided by the vendor of the used technology library.

The following steps show how to create a Milkyway database in the Synopsys tools:

1) Define the power and ground nets.

   For example, set the following variables:

```
set mv_power_net VDD

set mw_ground_net VSS

set mw_logic1_net VDD

set mw_logic0_net VSS

set mw_power_port VDD

set mw_ground_port VSS
```

   If you do not set these variables, power and ground connections are not made during execution of write_milkyway. Instead power and ground nets can get translated to signal nets.

2) Use the create_mw_lib command to create the Milkyway design library.
   For example,

```
 create_mw_lib -technology $mw_tech_file \ -
mw_reference_library $mw_reference_library \
$mw_design_library_name.
```

3) Use the open_mw_lib command to open the Milkyway library that you created.
   For example,

```
open_mw_lib $mw_design_library_name.
```

4) (Optional) Use the set_tlu_plus command to attach TLUPlus files. For example,
set_tlu_plus_files -max_tluplus $max_tlu_file \ -min_tluplus $min_tlu_file\ -tech2itf_map $prs_map_file.

5) In subsequent topographical mode sessions, you use the open_mw_lib command to open the Milkyway library. If you are using TLUPlus files for RC estimation, use the set_tlu_plus_files command to attach these files.
For example,

```
open_mw_lib $mw_design_library_name
set_tlu_plus_files -max_tluplus $max_tlu_file \
-min_tluplus $min_tlu_file \
-tech2itf_map $prs_map_file
```

The following Milkyway library commands are also supported:

- copy_mw_lib
- close_mw_lib
- report_mw_lib
- current_mw_lib
- check_tlu_plus_files
- write_mw_lib_files
- set_mw_lib_references

3.1.4 Table look up plus files:

The Table look up plus (tlup) files contain the resistance and capacitance per-unit values of each of metal layer to be used in extraction of the RC model of the estimated length of the interconnects or the actual interconnect, after the track assignment stage in the routing step.

They consist of 3 different files which are called tluMax, tluMin and mapping files where the mapping file maps the names of layers that are in the interconnect technology format (ITF) file to the ones in the technology file. Where the interconnect technology format (ITF) file contains an ordered list of all conductive and dielectric layers in the technology library.

3.1.5 Gate-level netlist:

The gate-level netlist contains the mapped all standard cells and IO pads and used macros in the RTL.

3.1.6 Constraints:

The timing, area, electrical DRCs and power constraints written in Synopsys design constraints (SDC) format. And that is used and written by the design compiler tool.

3.2Design planning step:

The Design planning step is very important step, as all the following steps depend on it, so that it should be performed carefully. It is usually done by many iterations after beginning by an initial planning as all design stages in all fields, not in the ASIC designing only. The following is a flow chart describing the Design planning step:

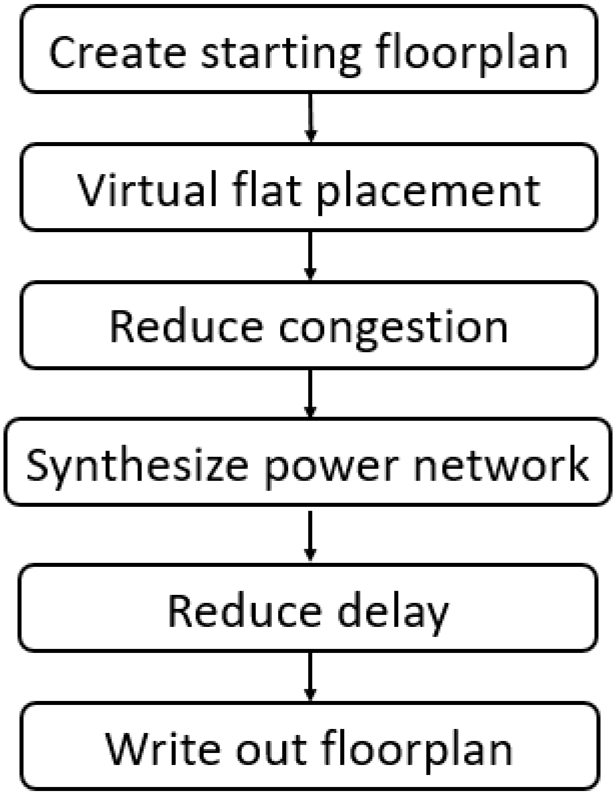


Figure 76: flow chart of design planning step

The following sections describe each step in more details.

3.2.1 Creating floor-planning:

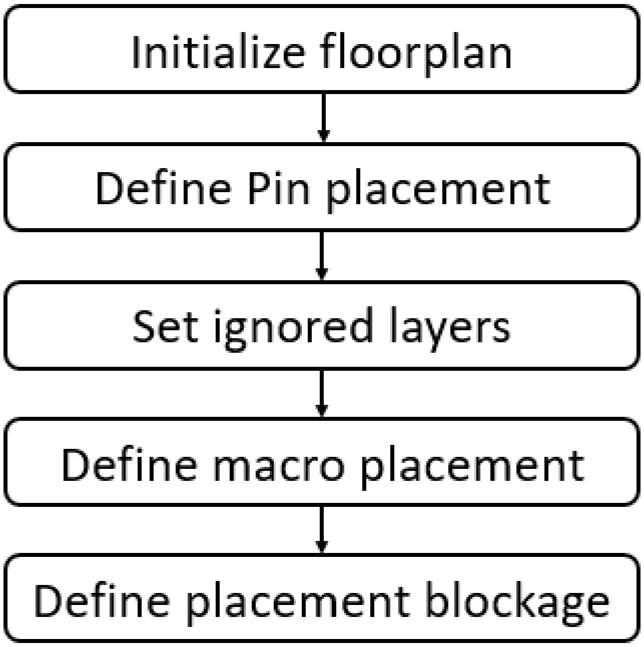


Figure 77: flow chart of creating floor-planning step

The Creation of the floor-planning begins with the “create_floorplan” command which takes the following arguments:

1) Control type by which the IC compiler tool set the height and width of the core of the chip. And it can be height and width or respect ratio.

2) Height and width dimensions of the chip core, if the control type is set to height and width value.

3) Utilization ratio which is ratio between the combined area of all used standard cells, IO pads, macros and estimated net area and the core area.

4) Respect ratio: which is the ratio between the the height of the core and the width of the core.

5) Determining whether the placement of the standard cells is done on horizontal or vertical rows.

6) The distance between the core boundaries and the IO pads edges from each side in the chip.

The pin placement is done by the “set_pin_physical_contraints” command and the pad placement is done by the “set_pad_physical_contraints”. The both previously commands are set for each pin and pad in the design to have well-defined placement of them, otherwise the IC compiler tool will place them automatically and they may be dis-ordered.

Defining of the macro placement is done by the collecting the macro in arrays and these array can be one or two dimensional and can have any number of macro cells in any order, the designer wants and the designer can specify the offset between the macro cells in an one array in both direction, the X and Y directions. The macro array is handled by the “set_fp_macro_array”.

After setting the macro array, the designer should specify the locations of these macro array by the “set_fp_macro_options” command, otherwise the IC compiler will place them automatically and the overlapping the different macro arrays can happen. It worst to mention that all the macro placement constraints are soft constraints, which means that the IC compiler tool will try to honor them, but if it finds better options than the specified ones, it will choose them over the specified macro placement constraints.

During determining the macro placemenet constraints, the designer should observe some basic rules when measuring QoR, such as pushing hard macros to the core boundary and aligning similar hard macros. However, the critical measurements that are used to evaluate the placement results are timing and routability.

Defining the macro keep-out margin is essential step as it prevent the placer engine in the IC compiler tool from placing the standard cells in specified area around the macro cells, therefore it will reduce the congestion and keep more space for the interconnects that connect the macro pins to the standard cells and IO pads.

3.2.2 Virtual flat placement:

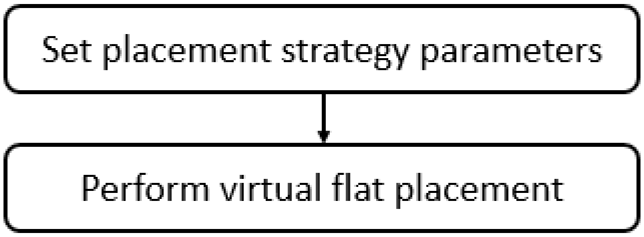

Set placement strategy parameters
Perform virtual flat placement

Figure 78: flow chart of virtual flat placement step

As the above flow chart shows, the strategy of the virtual placement engine should be set before beginning of that placement. The strategy is set by the “set_fp_placement_strategy” command. And the usual setting are:

1) The virtual in-place optimization (VIPO) is set to on.
2) Sliver size which determines the upper bound of the size of channels that are between the macro cells at which there is no standard cells placement.
3) Congestion effort.

After the setting of the virtual placement strategy, the virtual placement itself is performed by the “create_fp_placement” command and that command requires the driven by which the placement is guided, and the available option is timing one or/and congestion.

Virtual flat placement is the simultaneous placement of standard cells and macros for the whole chip. The initial virtual flat placement is very fast and is optimized for wire length, congestion, and timing. For designs with macros, plan groups, or voltage areas that have not already been placed, virtual flat placement can help you decide on the locations, sizes, and shapes of the top-level physical blocks.

3.2.3 Reducing congestion:

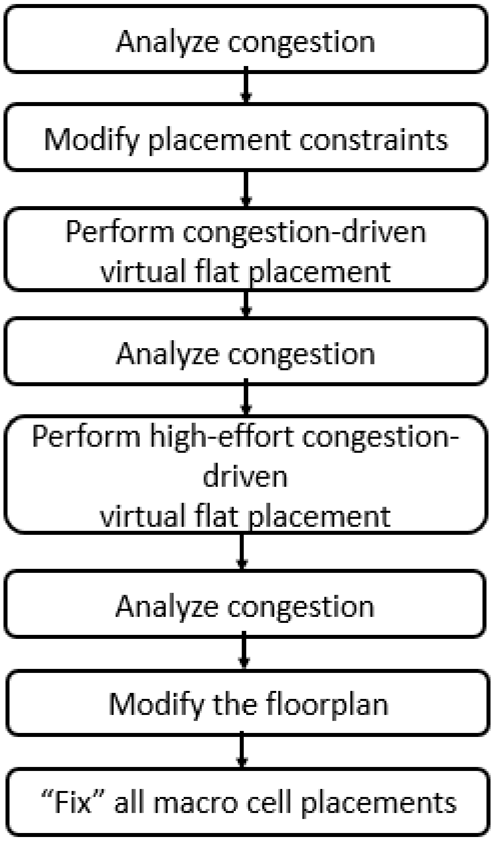

Analyze congestion
Modify placement constraints
Perform congestion-driven virtual flat placement
Analyze congestion
Perform high-effort congestion-driven virtual flat placement
Analyze congestion
Modify the floorplan
“Fix” all macro cell placements

Figure 79: flow chart of reducing the congestion step

After the virtual placement, the congestion should be analyzed and that is done by the “route_global –congestion_map_only” or “route_fp_proto -congestion_map_only”. The first command is more accurate than the other, but the second one is faster.

If the design is congested, the following is available to solve this problem:

1) Changing macro placement or their orientation, then run another floor-planning placement with a timing and congestion drivens.

2) Increasing core area by decreasing the core utilization.

3) Increasing the keep-out margins.

Congestion reporting is a map at which the design is divided into cells call global routing cells and Number of nets crossing the global routing cell (GRC) edge, versus Number of available routing tracks are computed and then compared.

3.2.4 Power network creating:

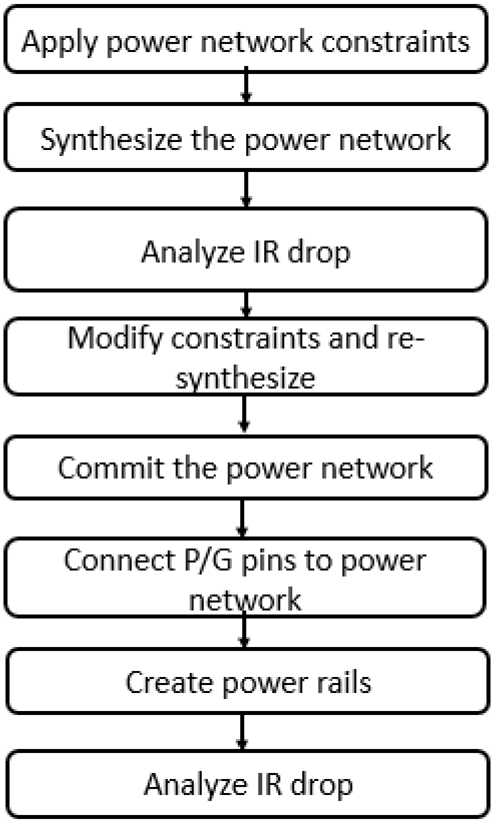

Apply power network constraints
Synthesize the power network
Analyze IR drop
Modify constraints and re-synthesize
Commit the power network
Connect P/G pins to power network
Create power rails
Analyze IR drop

Figure 80: flow chart of power network syntheisizing step

Power network consists of rings, straps, stacked vias and supply rails, where

1) **Rings:** they are thick wires routed in the space that between IO pads boundaries and core boundaries, and they are usually routed in the highest metal layers of the technology library (if the ASIC consists of 10 metal

layers, then the rings will be in the 10th, 9th, 8th and 7th metal layers), they are usually 2 rings, one for power and the other for ground, and they connected to the power and ground pads in order to move the power and ground from the outside the ASIC to the space between IO box and core area. They are created by the “create_rectangle_rings” command in the IC compiler tool.

2) **Straps:** they are thick straight wires routed across the core area itself in the same metal layers that contains the rings, they connect to rings using array of vias in order to move the power and ground (p/g) from the rings (space that between IO box and core area) to inside core area itself and they takes grid shape. They are created by the “create_power_straps” command in the IC compiler tool.

3) **Supply rails:** they are thin wires routed in the 1st layer, they are connected to the rings and straps using special vias called stacked vias and they supply the standard cells which also are in the 1st metal layer with power and ground. They are created by the “preroute_standard_cells” command in the IC compiler tool.

4) **Stacked vias:** array of vias that connects the supply rail in the 1st layer to the straps. They are created by the “create_preroute_vias” command in the IC compiler tool.

To pass this step, the IR drop has to be equal or less than 3% of the supply voltage. That to keep the noise margin of standard cells large and to prevent the delay of them from degrading.

3.2.5 Reducing delay:

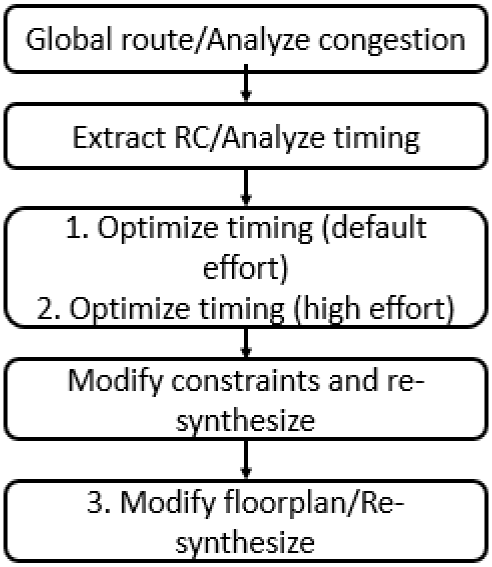

Global route/Analyze congestion
Extract RC/Analyze timing
1. Optimize timing (default effort)
2. Optimize timing (high effort)
Modify constraints and re-synthesize
3. Modify floorplan/Re-synthesize

Figure 81: flow chart of the reducing delay step

As the previously mentioned, the design is mainly performed by iterations, so that after synthesizing and committing of the power network, the delay must be analyzed and reported. The considered delay type checks are the setup timing one only.

3.2.6 Extracting:

The extracting of the RC models of estimated interconnects and actual wired ones, like the power network, should be performed and saved to be used in the dc compiler, like in the topographical flow.

3.3 Placement:

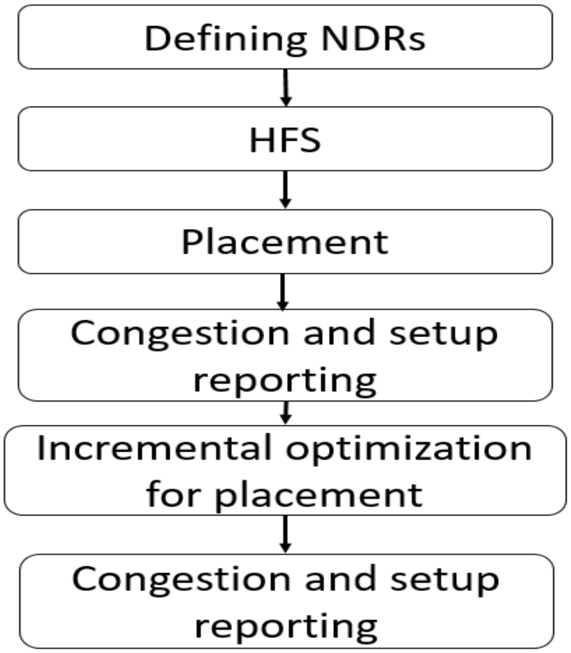

Defining NDRs
HFS
Placement
Congestion and setup reporting
Incremental optimization for placement
Congestion and setup reporting

Figure 82: flow chart for placement stage

There are many configuration settings that affect the behavior of placement in IC Compiler, including the following controls:

- Setting the Congestion Options
- Setting the Move Bounds
- Defining Inter-cell and Boundary Spacing Rules
- Defining the Buffer Strategy for Optimization
- Setting the Preferred Buffers for Hold Fixing
- Enabling Tie Cell Insertion
- Setting Placement and Optimization Attributes

To pass the placement step, the congestion and the setup timing slack should be acceptable.

The following sections go through each step in more details.

3.3.1 Setting of the non-default routing rules:

The non-default routing (NDR) rules are set for routing the clock trees, but it is set before the placement to be taken in the consideration during the placement stage. They are set using the "define_routing_rules" command followed by "set_net_routing_rule" command.

3.3.2 The high fan-out synthesis:

The high fan-out synthesis (HFS) targets the static signal nets with high fan-out number and builds a tree of buffers and inverters to divide these high fan-out nets into smaller nets, therefore reduces the output load capacitance and enhances the delay or place the standard cells away from each other to improve the congestion. The HFS is performed by a "create_buffer_tree" command.

3.3.3 Placement:

It is done by a "place_opt" command which performs the following:

1) global placement (coarse placement):
    - Choose location for each cell.
    - Cells can overlap.
    - Orientation is ignored.
    - Aims to meet the area and power target and avoid routing congestion.

2) Electrical DRC violation fixing (HFS)

3) Legalized placement:
    - Fixing the cell overlap.
    - Assigning the cell to site rows.
    - Aims to enhance the rout-ability and avoids DRC violation.

4) Optimization:

During this stage, the tool performs timing (solve setup violations only), area, congestion, and leakage-power optimization.

3.3.4 Congestion and setup reporting:

The congestion is reported by the “route_global” command. And the setup timing checks are reported by a “report_timing –delay_type max” command

3.3.5 Incremental optimization for placement:

The incremental optimization of placement is mainly performed by a “psynopt” command, but its options are set based on the target of that optimization:

If the congestion is wanted to be optimized, the options should be “-congestion”. And if the setup timing is wanted to be optimized, the options should be “-only_size”, “-no_design_rule” or “-only_design_rule”.

3.4 Clock tree synthesis:

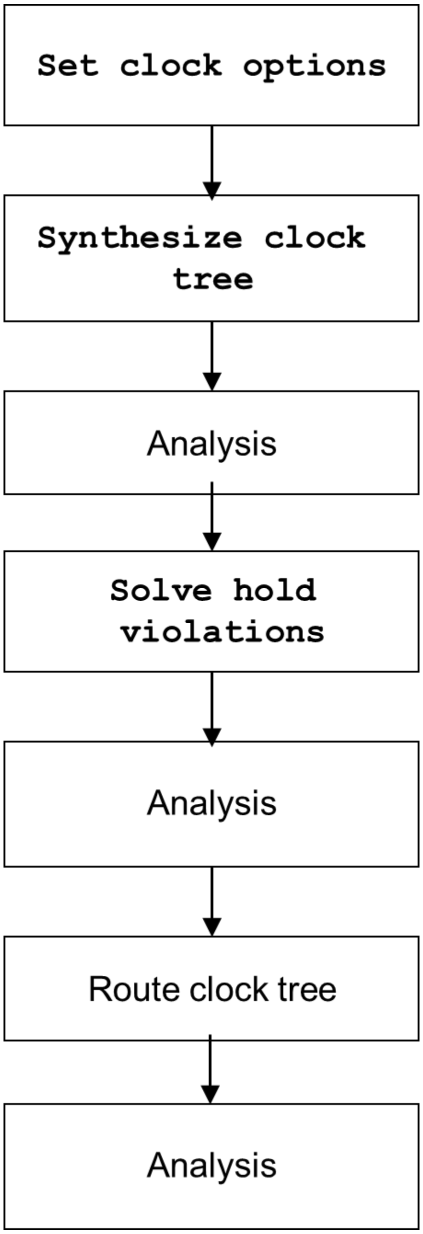


Figure 83: flow chart of clock tree synthesis

The following sections go through each step in more details

3.4.1 Setting of the clock options:

The setting of the clock options is done by a "set_clock_tree_options" command. And the common options are

1) The references by which the clock tree is built, and the target libraries must meet the following requirements:
   - Any cell in the logic library that you want to use as a clock tree reference (a buffer or inverter cell that can be used to build a clock tree) or for sizing of gates on the clock network must be usable by clocktree synthesis and optimization.

   - By default, clock tree synthesis and optimization cannot use buffers and inverters that have the dont_use attribute to build the clock tree. To use these cells during clock tree synthesis and optimization, you can either remove the dont_use attribute by using the remove_attribute command or you can override the dont_use attribute by specifying the cell as a clock tree reference by using the set_clock_tree_references command.

   - The physical library should include All clock tree references (the buffer and inverter cells that can be used to build the clock trees).

   - Routing information, which includes layer information and non-default routing rules.

   - TLUPlus models must exist. Where the Extraction requires these models to estimate the net resistance and capacitance.

2) Target skew: which is the maximum global skew, where the global skew is the difference in shortest insertion time and largest insertion time of 2 sequential elements.

3) Target early delay: which is the minimum insertion delay, where the insertion delay is the delay taken by the clock signal to pass from the clock port of the design to the clock pin of the cell.

4) Maximum fan-out.

5) Maximum capacitance.

6) Maximum transition.

7) Using default routing rules at sinks to prevent the DRC violations from happening during the clock tree routing.

8) Clock tree optimizations, like:
   - Buffer resizing
   - Buffer relocation
   - Gate resizing
   - Gate relocation

3.4.2 Clock tree synthesizing:

The synthesizing of a clock tree is done by the "clock_opt –only_cts – no_clock_route" command. And it is better to first synthesize the clock tree, hence analyze the results, and then fix the hold violations and finally route the clock tree.

3.4.3 Analysis:

1) CTS browser:
   - Properties and attributes on clock tree objects
   - Traversing clock tree levels
   - Symbols for CTS objects like buffers, gates and sinks

2) CTS schematic:
   - Trace forward/backward in schematic view
   - Collapses all sinks in the fanout of a CTS buffer for clearer CTS schematic
   - Highlight CTS objects in the layout view

3) Clock arrival histogram.

4) Report by the “report_clock_tree –type skew”.

3.4.4 Solve hold violations:

The Solving of hold violations is done by the “ clock_opt –only_hold_time” command after setting “set_fix_hold” command and “set_fix_hold_options”.

3.4.5 Routing of clock tree:

The routing of the clock tree is done by the “route_group –all_clock_nets” command. And that after checking for all clock tree options.

3.5Routing:

Routing creates physical connections to all clock and signal pins through metal interconnects and after the routing stage, the following requirements must be satisfied:

1) Routed paths must meet setup and hold timing, max cap/trans, and clock skew requirements.

2) Metal traces must meet physical DRC requirements.

Here a flow chart describes the main steps in the routing stage:

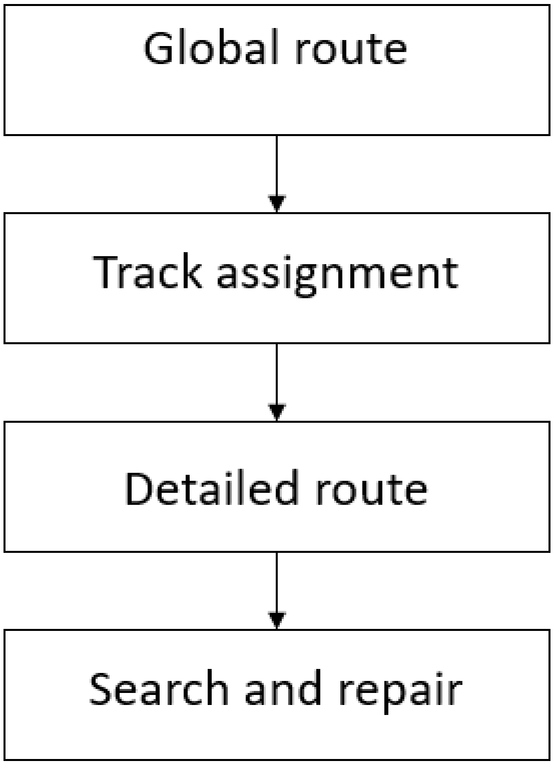


Figure 84: flow chart describes main steps in the routing step

The below sections go through each step in more details.

3.5.1 Global routing:

At the global route, the design is divided into small, square cells called global routing cells (GRCs) and the global routing engine tries to assign each net to specific global routing cell and then to specific metal layer in order to reduce the congestion, which is existed when more tracks are needed than available. And the global routing engine doesn't lay down any metal trace.

To illustrate the difference between the terms of metal track, metal trace and global routing cell, the figure below (figure 22) shows two metal layers, its metal tracks, and its metal track, where the metal trace is the real metal at which the signal travels and the metal track is an imagine centered line at which the metal trace can be made and finally the edges of the global routing cells represent a grid.

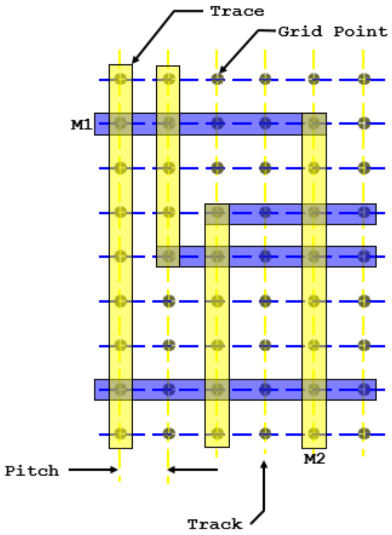


Figure 85: an example of a top view of the first and the second metal layers in a chip

3.5.2 Track assignment:

The track assignment (TA) engine does the following:

- Assigns each net to a specific track.
- Makes the actual metal traces.
- Makes long, straight traces.
- Reduces the number of vias.
- TA does not check or follow physical DRC rules.

3.5.3 Detailed routing:

The detailed routing (DR) does the following:

- Divides the chip into fixed sixes boxes called Sbox and fixes all DRC violations inside each Sbox only.
- DR may not be able to clear all DRC violations.

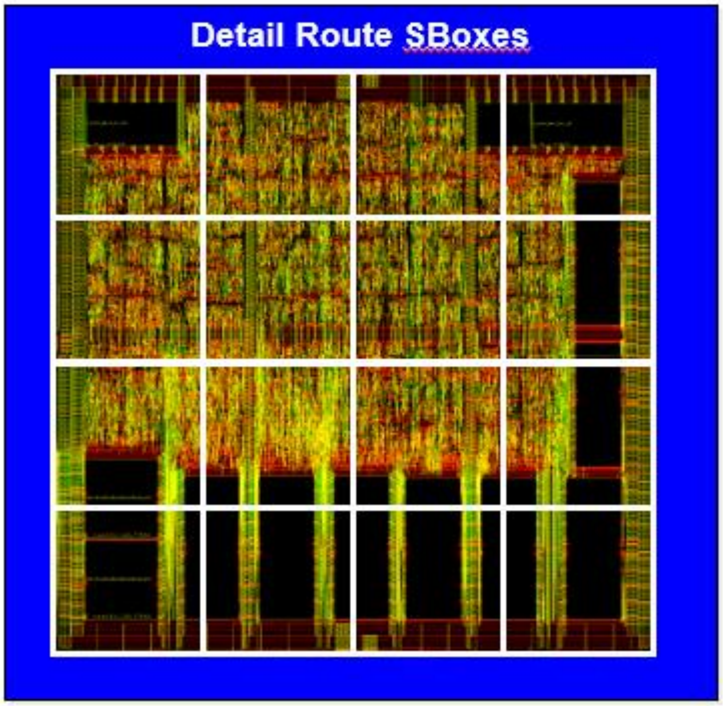


Figure 86: An example on the Sboxes

3.5.4 Search and repair:

The search and repair engine fixes remaining DRC violations through multiple loops and in each loop the tool increases the Sbox size and fixes the DRC inside it, the figure below (figure 24) shows an example of increasing the Sbox sizing.

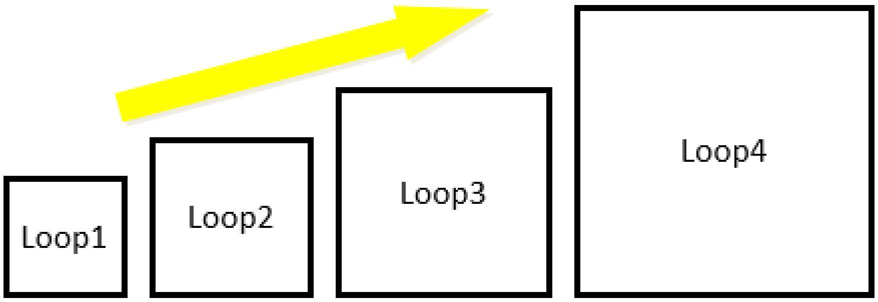


Figure 87: an example of increasing the Sbox sizing.

3.6Chip finishing:

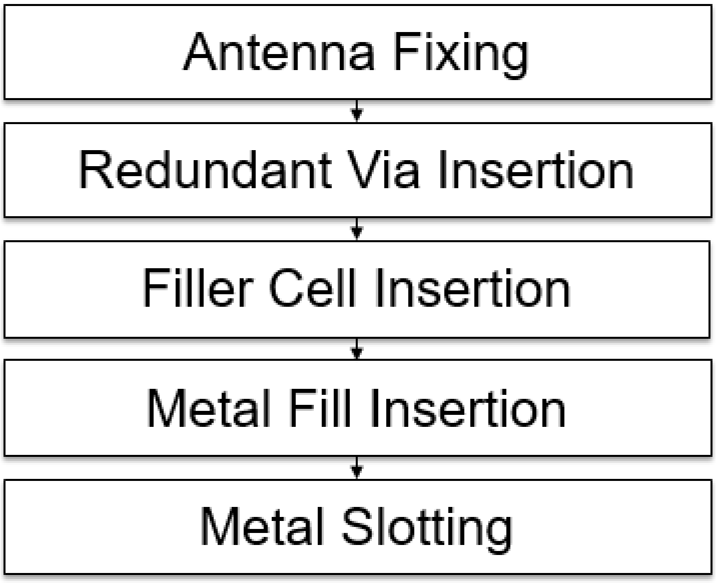


Figure 88: flow chart shows the main steps in the chip finishing stage

The below sections go through each step in more details.

3.6.1 Antenna fixing:

During etching of fabrication of metal layers of the IC exposed to a strong EM, which generates a high instant voltage on the MOSFET gate of the transistors, which may cause a damage to the thin SIO_2, therefore a failure of the transistor, so that the antenna DRC is set to make sure the metal layer that is connected to the gate is less that a specified value. That is done by determining an antenna ratio by the vendor, where Antenna Ratios = Metal Area Connected to Gate / gate area.

The solution of antenna DRC violations are

1) Layer jumping

2) Antenna diode

3.6.2 Redundant via insertion:

At the redundant via insertion step, the tool replace every stand-alone via with an array of vias. And that for the following reasons:

1) Reducing the resistance of vias.

2) And if a one via is damaged the other vias will work, therefore there is no single-point of failure.

3.6.3 Filler cell insertion:

At the filler cell insertion step, density of the chip needs to be uniform to increase the fabrication yield, that is done by filling the remaining empty sites in all placement rows with a special standrad cell called filler cell. And the same is performed with IO pads where the empty spaces between the wire bounding IO pads are filled with special pad called filler pads.

AT the IC compiler tool, the designer uses the “insert_pad_filler” command to insert IO cell fillers and the “insert_stdcell_filler” command to insert standard cell fillers.

3.6.4 Metal fill insertion:

A metal wire in low metal density region receives a higher ratio of etchant can get over-etched, therefore there is a DRCs to control this issue and it is called minimum metal density rules. And this step the tool add redundant wires to each metal layer to meet its minimum metal density rule.

3.6.5 Metal slotting:

The wafer is made flat (planarized) by a process called Chemical Mechanical Polishing (CMP), and at which Metals are mechanically softer than dielectrics, which can cause the following:

1) CMP leaves metal tops with a concave shape which is called dishing, and the wider the metal the more pronounced the dishing.
2) Very wide traces can become quite thin, therefore the dishing becomes severe and is called erosion.

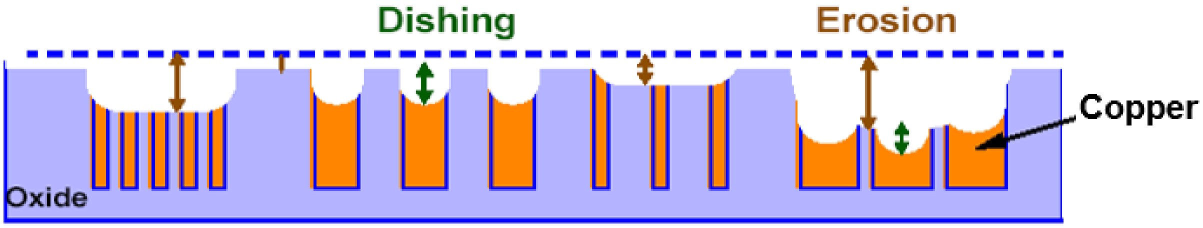


Figure 89: an example on the dishing and the erosion of copper

Maximum metal density rules are used to control erosion.

4. Main verification steps:

The main verification steps in the ASIC flow are the post-layout static timing analysis (STA) and formal verification. And the following sections go through each verification step in more details.

4.1Post-layout static timing analysis:

The post-layout static timing analysis can be done by the Prime-Time Synopsys tool. Where all critical paths in the design are analyzed to determine all timing violations that can be setup, hold, recovery and removal and to solve these violations.

The Prime-Time Synopsys tool takes the following as inputs:

1) Gate-level netlist, which is an output from the PnR tool.

2) Constraints, which mainly are physical, timing, area and leakage power constraints.

3) Standard Parasitic Extraction Format (SPEF) file which is generated by parasitic extractors like CALIBRE and contains the RC parasitic model of interconnects.

4) Logic libraries.

The Prime-Time Synopsys tool produces the following as outputs:

1) Timing reports.

2) Standard Delay Format (SDF) file, which contains delays of cells and interconnects that are in the design.

4.2 Formal verification:

The Formal verification is done by the formality Synopsys tools and it can be three types:

1) A RTL code versus Modified RTL code, which is used to check either the modified RTL has the old function of, the old RTL, or not.

2) A RTL code versus the gate-level netlist, which is used to check either the RTL code are synthesized correctly by the synthesizing tool or not. And this kind of the formal verification is used after the logic synthesis step.

3) The Gate-level netlist versus Modified gate-level netlist, which is used to check either the modified gate-level netlist still has the old function of the old gate-level netlist or not. And this kind of the formal verification is used after the clock tree synthesizing (or after any modification happens in the gate-level netlist).

There are very important definitions in the formal verification which are:

1) Reference Design: the basic/main design (design we sure about its functionality that is correct).

2) Implementation Design: the design that we need to check either its functionality is logically equivalent to the Reference design or not.

3) Logic cone: block consists of combinational logic which drives a compare point.
   - Compare point: it can be one of the following
   - Output port (primary output of the design).
   - Capture DFF
   - Input of a black box, that is within the design.

And the main Steps in the formality flow:

1) Read: Reference Design and Implementation Design are segmented into logic cones.

2) Match: tool matches or maps each compare point in the reference design to its corresponding compare point in the implementation design.

3) Verifying.

4) Debugging.

Code Availability: Code available upon request from any of the authors. Contact us at: hossam.gomaa99@eng-st.cu.edu.eg