Zewail City for Science and Technology
University of Science and Technology

Nanotechnology and Nano Electronics Engineering Program

ASIC Implementation of OpenPULP RISC-V Core

A Graduation Project
Submitted in Partial Fulfillment of
B.Sc. Degree Requirements in

Nanoelectronics

Prepared By

Ahmed Khaled	201601309
Amr Nasser	201601186
Islam Adam	201601026
Mennatullah Khaled	201601533
Moaaz Khaled	201600058

Supervised By

Dr. Hassan Mostafa	Signature
Dr. Amr Helmy	Signature
----------	----------

2020/2021

Table of contents:

1- Introduction and Literature review

1.1 General Introduction about RISC-V

RISC-V core processor is considered to be a free and open-source Instruction Set Architecture (ISA) that has paved the way for a new era of processor development and innovation through open standard collaborations. RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the next 50 years of computing design and innovation. The goal of RISC-V is to be a universal ISA in the field of microprocessors and to fit different and widespread applications. It could target all sizes of processors from the tiny ones in the embedded microcontrollers to the most suitable ones for high-computing applications. Moreover, it aims to be compatible with different implementation platforms either FPGA or ASIC. Also, it should be suitable with popular programming languages and software stacks. Although it has just appeared for a few years, it has proven its stability and continuity in its base ISA unlike the very early generations of AMD (Am29000), Intel (i960), Zilog (Z8000), and Motorola (8800). RISC-V ISA is defined as a base integer ISA, which must be present in any implementation, plus optional extensions to the base ISA. The base integer ISAs are very similar to that of the early RISC processors except with no branch delay slots and with support for optional variable-length instruction encodings.

There are two main types of ISA which are Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). The differences between these types can be summarized in the following table:

CISC	RISC
Original microprocessor ISA	Redesigned ISA that emerged in the early 1980s
One instruction can take several clock cycles	Single-cycle instruction
Hardware-centric design: ISA does a lot of computation using hardware circuitry in many cycles	Software-centric design using high level compilers
More efficient use of RAM than RISC	Multiple memory access
Complex and variable length instruction	Simple standardized instructions
Can support microcodes (instructions are treated like small programs)	Only one layer of instruction
A large number of instructions	A small number of fixed-length instructions
Compound addressing mode	Limited addressing mode

The ISA can decide the application of its processor. Globally, we could find that there are only two ISA which are X86 and ARM and each ISA application is totally different from the other. For instance, almost more than 99% of personal computers (PCs), laptops, and servers are based on X86 or AMD64 ISA and the intellectual property (IP) belongs to Intel and AMD respectively. In addition, more than 99% of mobile phones and tablets are based on ARM ISA and their IPs are divided into A series, R series, and M series. These implementations of single area usages of ISA were the reason why RISC-V has appeared to the market and could grow up very fast. RISC-V is a reduced ISA that has been designed to suit many fields of application, especially the applications of storage, edge computing, and AI applications. These applications are very critical applications in today's market. These applications that are uniquely addressed by RISC-V make it possible for RISC-V to compete with any ISA in the market such as X86 and ARM.

1.2 Problem Definition

Over the past few years, many problems arose regarding the speed of different applications, especially AI applications which need a lot of memory and high speed to be accomplished in their required time. AI acceleration is a set of approaches and algorithms to speed up high intensive tasks required. The acceleration can be divided into both software and hardware acceleration. Both kinds of acceleration should be incorporated to fulfill required tasks properly with respect to their constraints.

To achieve targeted processing constraints, the parallelism concept is introduced. Parallelism is meant to split the required computation into small tasks so that they can be spread among small computational blocks. This approach can be done on-chip level with schedulers and multi-core processors. There are many products from different companies that are introduced to fulfill the tasks of hardware acceleration. These products can be categorized into four different computing devices: FPGA, GPUs, ASIC, and Neuromorphic chips. FPGA (Field Programmable Gate Array), is a class of computing elements that can be reprogrammed through its gates. GPUs (Graphical Processing Units) were deployed for only graphical processing at first. However, it has been found that they can be adopted for AI and achieved great results. ASIC (Application Specific Integrated Circuit) can be described as a family of processors to carry out certain specific tasks.

In this project, we will integrate different platforms of hardware implementation. We will implement an optimized RISC-V core optimized for high-speed computation and integrate many of this core to achieve the concept of parallelism into a GPU and all of this work will be

integrated later with a hardware FPGA implementation for one of the machine learning algorithms (Convolutional Neural Network (CNN)) achieving the optimum approach for integrating software and hardware acceleration.

1.3 Objectives

The recent silicon industry suffers from a shortage in manufacturing process components which leads to a massive storm in related industries like automotive and cell phones. This led big companies like Apple to postpone their new release of the iPhone. Also, there are some expectations for price increase through the rest of 2021 in the automotive and electronics markets. It is a supply chain so any change in a step will lead to a constructive effect on the following steps whether it was positive or negative. Modern technology companies must have a time-to-market specification in order to keep competence. Time-to-market means the time taken from the first ideation to selling the product to the customers. Through this time design, manufacturing, testing, and marketing are made. Our main scope is the backend design of RISC-V from RTL to GDS||. RISC-V is already used in many applications and there is a need to accelerate its design process. Our main objective is to provide a clear GDS|| file of the open-source RTL Design of OpenPULP Core (CV32E40P) using three different flows ; the flat, hierarchal, and topographical flows. Targeting high-performance matrix with higher speed and low area. Then, this design could be used as a part of an AI acceleration system that has a hardware part and a software part, the software would be used on our RISC-V core.

1.4 Functional Requirements/product specification.

As mentioned before, we hope to use this RISC-V core inside an acceleration system for AI applications specifically for automotive vehicles. According to the recent research in the area of automotive researchers are looking for high-speed systems with a low area as in runtime a car

would face a high-speed environment with varying inputs for the system which needs an ultra-high processing capability. Working on the Frequency we target to work at the max available speed in the scope of our used library and computational capabilities. For power, we target to have low IR drop through the design to be able to be suitable to our target application. As initial specifications hope to achieve these values for area power and speed.

1. Area: 0.1 mm2.
2. Clock frequency: 500 MHZ.
3. Power:

1.5 Report Organization

This thesis document consists of six chapters, chapter 1 gives a brief background about RISC-V architecture. Then it goes to our problem definition and how it could be solved followed by the objectives and product specification and report organization. In chapter 2 a market research with literature review is made to track the revolution of RISC-V in today's industry. Chapter 3 describes the core work made through the project including project purposes and constraints in a section, followed by the project technical specifications section. Moreover, design alternatives and how to differentiate between them is discussed in design alternatives and justifications. At the end of chapter 3, we discuss the chosen design and its components in two sections. Proceeding to chapter 4 project execution is clarified by showing the task taken through the journey and Gantt chart followed by a detailed description of the different subsystems inside the design, at the end of the chapter we discuss testing and evaluation of the final product. Following, Chapter 5 shows the cost analysis and evaluation of the project in terms of environmental impact, ethics and sustainability …etc. Last but not least, a conclusion is driven in chapter 6 followed by used references.

2- Market and Literature Review

2.1 A survey of the state of the art concerning the subject under consideration.

As mentioned above in the introduction section, PULP is a collaborative project between the Integrated Systems Laboratory (IIS) of ETH Zürich and Energy-efficient Embedded Systems (EEES) group of the University of Bologna. The IIS are having a long tradition in tapping out ASICs. They have tapped out nearly 500 chips in different applications. Since the IIS is greatly involved in the PULP project beside tapping out ASICs, they have tapped out more than 30 PULP based ASICs ranging in complexity and targeting different applications. In this section, we shall review the most important of these different implementations arranged from the oldest to the most recent in some detail. That Chronological order is preserved unless we had to mention related chips together. A summary including the year, application, technology, manufacturer, chip dimensions, number of gates, voltage and frequency is presented in table 1.

In 2013, the chip Or10n was implemented using the UMC180 technology. This chip does not use the Open RISK code. But rather the code is optimized for ASIC implementation. Also improvements have been made to increase the IPC -instructions per cycle to above 0.9. The same improved core was also used in the Sir10us chip with minor changes in the memory implemented.

Parallel Ultra Low Power Processor (PULP) is a shared data memory, parallel processor architecture. It has 3 different versions PULPv1, PULPv2, and PULPv3. PULPv1 is implemented using ST28 FDSOI technology using lower leakage RVT transistors. It contains one cluster with 4 OpenRISC cores modified to have a much higher IPC and a dedicated

DMA controller. The aim of the chip design is to test aggressive body biasing techniques especially to reduce the transistors off current. The system has 6 power islands that can be switched between two external body bias voltages in a short period of time[1].PULPV2 differs from PULP v1 by 5 main points. First, it uses the faster low-leakage (LVT) transistors. Second, It has an FFL integrated within the system for internal clock generation. It has standard cell-based memories for low-voltage operation. AXI buses and Master and slave SPI interfaces are also included. It has 64 kBytes of L2 memory, 40 kBytes of TCDM (out of which 8 Kbytes is SCM), and in total 4 kBytes of private SCM[2]. Finally, the PULPV3 version of the chip was implemented using the same technology as PULP v1. In total 128 kBytes of L2, 48 kBytes of TCDM (32 kByte SRAM + 16 kBytes of SCM) are included. There is a shared 4 kBytes 4 way set associative instruction cache for the processor cores using SCMs. Moreover, the chip has two body bias regions. The body bias voltages for each region vary from -1.8V to Vdd/2 + 0.3V. The chip also includes two on-chip FLLs and the cluster can be completely shut down, which is another useful low power feature. Besides, there is an integrated dedicated hardware accelerator for 2D convolutions and a hardware convolution engine with energy efficiency and bandwidth consumption optimizations[3].

The four chips, Artemis[4], Hecate, Selene[5], and Diana[6] represent a series that aims to add floating-point operations to a 4 core cluster. The cores used are the Or10n. The main difference between the 4 chips is in the number and type of FPUs used. In Artemis, a dedicated FPU is added to each core. In Hecate, it was noticed that there is a small probability that all cores utilise the FPU at the same time. Thus, the 4 cores here share 2 FPUs. Selena chip is a little bit different in that it uses a different representation for floating-point where it uses a Logarithmic Number System(LNS). The LSA makes the

difficult calculations as the multiplication and division much easier while the normal addition and subtraction processes are much more difficult in LSA. Since the LSA FPU is much larger than the ordinary FPU used in the other chips, only one LSA FPU is shared among the cores in the Selene chip. Finally, the Diana chip is very similar to Artemis with a dedicated FPU in each core. However, to tradeoff area and power vs precision, one of the FPUs is exact - similar to the one used in Artemis- while the remaining 3 FPUs are inexact.

Mia-Wallace chip (2015) is designed to have enough memory and interfaces allowing it to be used in a variety of applications. It contains 4 Or10n cores that support both vector instructions and a functional debug interface. It has two power domains, one at 1.2V and the other at 0.3V. At the lower power, the standard SRAM is not functioning. Besides, the chip has 2 FLL to generate an operating frequency ranging from 0 to 500MHz from the standard 32kHz oscillator for both the cores cluster and other components on the SoC. It also includes a large L2 memory as well as different interfaces like SPI, UART and a JTAG port. A convolutional accelerator is also included where it can compute 2x 16-bit pixels per cycle[7].

VivoSoC is a combination of flexible analog acquisition circuitry and a SAR-ADC with a PULP processor. Its main application is to be used in biomedical signal acquisition as in wearable EEG, ECG and EMG. It uses a dual-core PULP processor along with other peripherals. It was implemented in 2015. Later on in 2016, VivoSoc was updated and extended to make the second version VivoSoC2[8] and VivoSoC2.001[9] for a wider range of applications including mass-market consumer fitness tracker, medical-grade telehealth point-of-care, implantable devices for chronic disease monitoring and management. VivoSoC2 is empowered by a Quad-core RISK processor equipped with a larger variety of

peripherals. The third generation of the VivoSoC collection includes the VivoSoC3 and VivoSoC 3.142 which were developed in 2018 and 2019 respectively. They both target the same application and also support simultaneous acquisition and processing. For all the VivoSoC collection, the tight energy budgets of portable devices needed to be considered. Thus, fine-tuning of the operating point of each SoC block was used to trade-off unnecessary precision with power. The technology used with VivoSoC is SMIC 130/110.

In 2015, the three chips Manny, Sid, and Diego were tapped out for research purposes. They were designed with a target of the lowest possible power consumption taking the advantage of near or subthreshold operation, beside approximate computing principles. The chips system is based on a 4 core PULP system 64 kByte of L2 memory, 16 kBytes of TCDM and 4 kBytes of shared instruction cache. This system is then augmented by a shared approximate single precision floating point unit. In addition there is a generic hardware accelerator interface which allows the in-exact accelerators to be connected to the main system. Two generic FIR filters are also present on the chip. The main difference between the three chips is the used library where the Manny chip uses the subthreshold library, and thus should be the largest and slowest. The Diego library is the low vt library and thus should be the fastest. And finally the Sid chip uses the standard library. The three chips are considered to be of high area of more than 50 square millimeters. The implementation technology is the Alp180.

Honey Bunny chip (2015) is the first PULP chip utilizing RI5CY. It has 4 RI5CY cores, 64 kBytes of TCDM,256 kBytes of L2 memory and 4 kBytes as standard cell based memories. This large memory is designed to support various applications. An FFL is included in the design to support the desired frequency within the range from 0 to 666MHz.

Imperio (2015) is the first ASIC implementation of PULPino in the UMC 65nm technology. PULPino is a microcontroller-like system based on a small 32-bit RISC-V core with an IPC close to 1[10].

Patronus is a chip implemented in 2016 that uses the RISKY core which has been optimised for energy efficiency. However, it is considered a large chip with around 40kGE. The Patronus system has a special design where it has 3 cores: Eeny, ZeroRISK and Remus. Eeny is a single cycle RISK-V core. ZeroRisk is a RISK-V core with 3 stage pipeline. Remus is a RISC-V core with CFI (Control FLow Integrity) to withstand fault attacks. At a time, only one of the cores is active and that is what makes this system different. All the cores have access to the different memories in the system as well as the different interfaces.

Mr. Wolf chip (2017) is an IoT processor utilizing optimized RISK-V cores. It includes one cluster of eight 32 bit RISC-V cores, two FPUs each shared by 4 cores, 64 kBytes TDMC beside 512 kByte of L2 memory. A zero-risky is used as the controller of this design. Also an LDO is integrated to generate the internal voltages**[11]**.

The Automario chip developed in 2018 using the UMC65 technology is the first ASIC implementation of a multi-cluster PULP. It contains 2 clusters each with 4 RISKY cores beside an instruction cache of 8 kBytes and TCDM memory of 64 kBytes.

Poseidon chip (2018) is the first PULP based design in the GF 22nm technology. Poseidon is built from 3 independent modules. The first is called Kerbin including the 64 bit RISC-V core, Ariane, along with a 32 kB instruction cache and a 32kB data cache. The second module is basically a PULPissimo design updated with an autonomous uDRAM I/O

subsystem. It also includes a small hardware accelerator for binary neural networks in addition to 512kB of memory. The third and final module in Poseidon is called Hyperdrive which is a binary neural network accelerator**[12**].

Kosmodrom chip (2018) is designed to help in the evaluation of different library options of the GF 22FDX process using a realistic benchmark. It includes two 64 bit Ariane cores. One of which is optimized for high performance(1.3GHz typical) and the other is optimised for extremely low power consumption(300MHz typical case operation) using different optimization corners and different standard cell libraries. Moreover, the design included 1.25 MByte of SRAM memory that is used by the Ariane cores, an Adaptive Body Biasing block, a neurostream accelerator, and test structures for electro-optical interfaces[13]. Later on in 2019, an improved version of Kosmodrom called Baikonur was issued fixing issues related to third party IP[14]. The main difference between the two is that the two Ariane cores in Baikonur had different operating conditions than Kosmodrom. That is, one core is optimized for high performance(1GHz, worst casel) and the other is optimised for extremely low power consumption(300MHz worst case operation)

Arnold (2018) is a design that combines the PULPissimo design with an eFPGA where the eFPGA is programmed through the PULPissom's RI5CY core via memory mapped r/w operations. Since the eFPGA has an extra APB bus interface, it can be accessed as a standard peripheral. The eFPGA can also access the same memory that the processor can access as it has 4 TCDM access ports. The chips also include a set of peripherals and a uDMA to copy data from and to the peripherals[15].

Scarabaeus -standing for Specifically Crafted Acronym Referencing an Ariane in a Brilliantly and Artistically Engineered Unprecedented SoC- is a PULP based SoC developed using the UMC65 technology in 2018 as well. It represents an Ariane based RISK-V SoC that includes a Data and Instruction cache of 4 kBytes and 64 kBytes of L2 memory. The Design is improved by adding a Platform Level Interrupt Controller (PLIC) and a DMA controller that facilitates data transfer to upto 4 dimensions.

Xavier Chip(2019) is an enhanced PULPissimo system including a RISKY core(RV32ICMF) as the main processing engine. It also has a uDMA system that can handle 8 SPI ports. A hardware accelerator is also included for quantised neural networks. The system has 512kByte memory. The Hardware Processing Engine (HWPE) uses the small zero Risky core (RV32-ICM)as a controller rather than using a control FSM.

Urania Chip (2019) is the first ASIC implementation for the bigPULP design used in the HERO project. Urania is a heterogeneous system based completely on RISK-V. It contains an Ariane core as the main processor. It also has two clusters of 4 RISKY cores each. Each core has an individual FPUs. Each cluster has a specialised Hardware Accelerator called PULPo. It is responsible for the first order optimizations to solve a range of problems.

Rosetta Chip (2019) is based on PULPissimo architecture utilizing the RISKY core with a new vector processing ISA extension and multiple accelerators to support different kinds of signal processing applications. The chip has a wide variety of different memory technologies. It has 4 conventional SRAM banks with a total size of 512KiB for normal computations demanding high memory capacities. Other two xSRAM banks of 64KiB total capacity are specifically for the core to execute programs simultaneously without bank conflicts.

Furthermore, a 4 KiB two port latch based standard cell memory is included, which executes small programs at low supply voltages. Other memories such as 64 KiB of eDRAM and 32 KiB of in-SRAM are also available. Beside the different memory technologies, the chip includes a programmable autonomous accelerator for signal processing. As with most of the PULP chips, a wide variety of peripherals are present as well.

Table 1.1

	Or10n	Sir10us	PULP v1	PULP v2	PULP v3	Artemis, Hecate, Selene & Diana	Mia-wallace	VivoSoC	VivoSoC2 & VivoSoC2 .001
Year	2013	2013	2013	2014	2015	2014	2015	2015	2016
Application	Processor	Processor	Pulp	Pulp	Pulp	Pulp	Pulp	Biomedica l	Biomedic al
Technology	180	180	28	28	28	65	65	130	130
Manufactur er	UMC	UMC	STM	STM	STM	UMC	UMC	SMIC	SMIC
Type	Semester Thesis	Semester Thesis	Research	Research	Research	Semester thesis	Research	Research	Research
Dimension s	1525µm x 1525µm	1525µm x 1525µm	1650µm x 1650µm	1650µm x 1650µm	1650µm x 1650µm	1252µm x 1252µm	3950µm x 1875µm	4000µm x 3200µm	4368µm x 4768µm
Gates	80 kGE	100kGE	700 kGE	1800 kGE	2500 kGE	600 kGE	2 MGE	600 kGE	800 kGE
Voltage	1.8 V	1.8 V	0.4-1.2 V	1.0 V	1.0 V	1.2 V	1.2 V	1.2 V	1.2 V-0.6V
Power	1 mW, 1MHz 1.8V	93 mW, 166MHz 1.8V	8 mW @0.7V, 10MHz	100 mW	1.2 mW @ 0.6V, 50MHz	1 mW @1.2V 1MHz	1 mW @1.2V 1MHz	45 mW (@40MHz , 1.2V)	20 mW (@50MH z, 1.2V)
Clock	360 MHz	166MHz	475 MHz	1000 MHz	66 MHz @ 0.6V supplyMHz	500 MHz	400 MHz	140 MHz	64 MHz

Table 1.2

	VivoSoC3 & VivoSoC3 .142	Manny	Diego	Sid	Honey-Bunny	Imperio	Patronus	Mr. Wolf	Automario
Year	2018/19	2015	2015	2015	2015	2015	2016	2017	2018
Application	Biomedical	Pulp	Pulp	Pulp	Pulp	Pulp	Pulp	IoT	Pulp
Technology	110	180	180	180	28	65	65	40	65
Manufacturer	SMIC	ALP	ALP	ALP	GF	UMC	UMC	TSMC	UMC
Type	Research	Research	Research	Research	Research	Semester thesis	Research	Research	Semester thesis
Dimensions	4368µm x 4768µm	7201µm x 8160µm	7201µm x 8160µm	7201µm x 8160µm	1500µm x 2000µm	1252µm x 1252µm	2626µm x 2626µm	3200µm x 3200µm	2626µm x 2626µm
Gates	2 MGE	2 MGE	2 MGE	2 MGE	2 MGE	500 kGE	5 MGE	1800 kGE	3 MGE
Voltage	1.2 V-0.6V	0.6 V	0.8V	1V	1.2 V	1.2 V	1.2 V	0.8-1.1 V	1.2 V
Power	10 mW (@50MHz, 0.8V)	3 W @0.6V 1.5MHz	3 W @0.8V 15 MHz	3 W @1.0V 15 MHz	1 mW @1.2V 1MHz	32.8 mW @1.2V 400MHz	999 mW @ 1.2 V, 400 MHz	153 mW @ 1.1 V, 450 MHz	54 mW @ 1.2 V, 200 MHz
Clock	100 MHz	1.25 MHz	15 MHz	15 MHz	60 MHz	650 MHz (typ) MHz	100 MHz	450 MHz	200 MHz

Table 1.3

	Poseidon	Kosmodrom	Baikonur	Arnold	Scarabaeus	Xavier	Urania	Rosetta	
Year	2018	2018	2019	2018	2018	2019	2019	2019	
Application	Pulp	Pulp	Pulp	Pulp	Pulp	Pulp	Pulp	Pulp	
Technology	22	22	22	22	65	65	65	65	
Manufacturer	GF	GF	GF	GF	UMC	UMC	UMC	TSMC	
Type	Research	Research	Research	Research	Semester thesis	Semester thesis	Research	Research	
Dimensions	3000µm x 3000µm	3000µm x 3000µm	3000µm x 3000µm	3000µm x 3000µm	2626µm x 1252µm	2626µm x 2626µm	4000µm x 4000µm	4100µm x 3000µm	
Gates	-	25MGE	80 MGE	-	1200 kGE	3 MGE	6 MGE	6 MGE	
Voltage	0.8 V	0.8 V	0.8 V	0.8 V	1.2 V	1.2 V	1.2 V	1.2 V	
Power	30 mW @ 0.8 V, 700 MHz	1 pW @ 1.8 V, 1 MHz mW	-	-	45.97 mW @ 1.2 V, 200 MHz	20 mW @ 1.2 V, 100 MHz	1 pW @1.2 V, 1 GHz mW	-	
Clock	700 MHz	1300MHz (typical)	1000MHz(typical)	330 MHz	200 MHz	250 MHz	100 MHz	190 MHz	

2.2 Literature that relates to the subject

A single-core RISC-V 64-bit instruction set architecture (ISA) was implemented in 28 nm technology by a team in Berkeley in 2018. The design was described using Chisel hardware construction language. This language, through features of object oriented programming language, allows fast exploration of design-space while guaranteeing a synthesizable design. The main obstacle in design synthesis was synthesizing multiport register files because they were not offered within the foundry standard cells. So, the multiport registers were custom designed as macros of standard cells to improve performance and help with design convergence. Tri-state buffers were used in the design of multiport registers because they allow multiple entries to be read on the same wire which relieves routing congestion. Further, the read wire was composed of shorter wires arranged hierarchically and connected to the global read wire using multiplexers to reduce parasitic capacitances. Overall, the register files were described at the gate level and went through manually guided place & route. The chip area was 2.7mm x 1.8mm and the core operated at 1 GHz frequency when Vdd was between 0.6 V and 0.9 V. It achieved 3.77 CoreMark/MHz in the famous benchmark CoreMark [16].

Another team from Berkeley and MIT managed to actually fabricate a 64 bit dual core RISC-V processor in 45 nm technology. They used the exact same synopsis tools we are using to perform both synthesis and back end place and route steps. However, their original RTL was written in Chisel but translated into verilog before synthesis. The total chip area of both cores was 2.8 mm x 1.1 mm and it achieved a peak frequency of 1.3 GHz with a Vdd of 1.2 V. Further, it was shown to achieve a high energy efficiency of

16.7 double-precision GFLOPS/W.The advantage of this implementation which makes it special is the integration of a custom vector accelerator with each core. This is a good example of the extensible open source nature of the RISC-V ISA [17].

2.3 Overview of the tools and techniques needed to build state of the art system

Building a state of art ASIC system requires going through the ASIC flow. The three main steps in the flow are the RTL code writing and testing, the synthesis, and the PnR steps, beside the physical verification and timing checking steps. The most popular tools used in logic synthesis are Leonardo Spectrum (Mentor Graphics) Design Compiler (Synopsys), RTL Compiler, Genus™ Synthesis Solution (Cadence). The most popular tools used in the PnR step are Innovus® Implementation System(Cadence), ICC(Synopsys). Finally, the famous tools for physical verification are Calibre DRC, LVS, PEX (Mentor Graphics) Diva, Assura (Cadence), Hercules, ICV (Synopsys).

There are 3 flows that can be followed for ASIC chip design. These are flat, hierarchical and topographical flow. The flat and hierarchical flows are very similar except for the synthesis step. In the flat flow synthesis, the synthesis tool performs auto ungrouping to all the hierarchies in the design. In other words, the design hierarchy is not preserved. The privilege of that is that it allows the tool to perform further optimisations. On the other hand, the hierarchical flow disables the auto ungroup option and thus, preserving the design hierarchy. In both of the previous flows, synthesis is only performed once, then we move to the PnR step. The topographical flow is a little bit different.Synthesis is performed once then the resulting netlist is sourced in ICC and only the power grid is

implemented. Then, the resulting power grid is taken to DC in the topographical mode where a coarse placement and second pass synthesis take place. Then, the resulting floor plan with the coarsely placed cells are taken once again to ICC to perform placement optimization, clock tree synthesis and routing.

In this project, we are following the same ASIC Flow using the Synopsys tools for both logic synthesis and PnR. For Synthesis we are using Design Compiler tool. For PnR, we are using IC Compiler, and for Physical Verification, we are using Herculis. We preferred the Synopsis tools since we are doing this project in collaboration with Si-Vision acquired by Synopsys. After finishing routing successfully, we will use Primetime tool (PT) as a signoff timing check that verifies that the setup and hold timing constraints are met. If not, PT suggests modifications to the timing path to solve the violations. The suggested modifications are then passed back to ICC to perform suitable routing for the added or modified cells. Multiple iterations of timing modifications and routing should then be done until all the timing constraints are met and no physical violations exist.

Regarding the flow, this project intends to perform the 3 flows: Flat, hierarchical and topographical flow. A final comparison between the outcome of the 3 flows shall be presented as well.

3- Project Design

3.1 Project purpose and constraints

The main purpose of the project is to implement a layout of the RISC-V processor that satisfies a very stringent timing constraint. Additionally, the project is intended to help the team members acquire hands-on and detailed experience of the Synopsis ASIC tools. Finally, it is required to obtain a DRC and LVS clean layout. Several constraints restrict the available options in the different steps of the layout implementation process. First, the timing constraint is a clock period of 2 ns which is considered a very high frequency for similar architectures. Second, the design is implemented with an educational process design kit (PDK) that uses a primitive and simple fabrication technology. For instance, the used PDK performs all signal and power routing including the power grid in only 10 metal layers which causes a lot of routing errors. Further, there are only a small number of standard cells available which increases the area and delay due to using multiple cells. Finally, the technology node is a relatively old one which results in worse timing and area. The third constraint is the core utilization which is naturally bound between 0.25 and 0.4. This increases the chance of the occurrence of DRC violations. Fourth, the voltage drop limit that ensures that all cells are working properly and fast enough sets many restrictions on the implementation of the power grid and power routing in general. We choose the maximum allowed voltage drop to be 22 mV. Congestion is also another source of DRC and LVS errors and its value also can affect the timing of the design. So, the fifth constraint is congestion strictly below 1%. Congestion above this value has been experimentally correlated to significantly more DRC and LVS errors. The final constraint is that a large portion of the design is synchronous which requires an extensive and complicated clock tree routing. Not

only does this set more difficulty on achieving the required timing constraint, but also is a source of DRC and LVS errors in signal routes due to the limited number of metal layers. Please refer to the below table for a complete list of the project constraints.

Clock period	2 ns
Technology Library and Design kit	1- Few standard cells 2- Old Technology 3- Limited metal layers
Core Utilization	0.25 - 0.4
Voltage Drop	Below 22 mV
Congestion	Below 1%
Clock Tree Routing	Many synchronous parts

3.2 Project technical specifications

As mentioned previously, RISC-V has many applications. Yet, our target for this project is to optimize the RISCY core to best fit hardware acceleration applications such as CNN hardware acceleration. So, the core will be used as part of the processing system for CNN computation. The most important criteria for these targeted applications is the speed, so our main work is performing timing optimization for the design in all the stages starting from compilation, floorplanning, placement, clock tree synthesis, till routing.

Our main factor is the frequency value which depends on the design architecture, our flow and optimizations, and the library used. The design and library are already determined, so we go through the three flows, flat, hierarchical and topographical, and try to do optimizations on all of them. As a result, what we do now is we start from low frequency and track the slack, then repeat

the process again using higher frequency until we reach the maximum one. For now, we are working on a 3 ns period for the flat flow, and similar periods for the other flows.

3.3 Design Alternatives and justification

3.3.1 Alternative libraries

In Semi-custom ASIC approach designers are using standard cells to implement their design in a fast way. These cells are provided by the foundry as a cell library, which contains all the information needed about each logical cell like AND, OR, FF …etc. Each foundry has a different library as the manufacturing process differs for each foundry [18].

Available cell libraries:

a. NangateOpenCellLibrary_PDKv1_3_v2010_12

OpenCell Library was introduced for the first time by Nangate and then was donated to Si2.org for open use. Then it was introduced to educational use in universities to help develop new EDA tools and techniques. The first development of this library was made by Nangate's Library Creator™ and the North Carolina State University 45nm FreePDK Kit. This library was preferable as it has 3 corners of characterization (slow, typical and fast), multiple flavors of threshold voltages for the available cells and multiple strengths for the basic gates -which allows high fanout-. In addition, it has a wide range of cells like half adders, full adders, Tri-state buffers, latches, inverters, filler cells and clock gated cells [19].

b. SAED_EDK90_CORE

SAED PDK is introduced by Synopsys for educational purposes, it is free for use. It was anticipated to design different combinational and sequential cells. It also provides different driving strengths for each cell and different threshold voltages for the transistors used. It also has some peripherals like Retention Flip-Flops, Isolation Cells, Always-on Buffers, Level Shifters, and Power Gating Cells. These cells strengthen the ability of the library to make higher performance designs [20].

c. ASAP 7nm Predictive PDK

This PDK was introduced for academic use and It does not follow any foundry, it uses the most recent models of 7nm transistors, it supports four voltage threshold voltages (Vth) for both PMOS and PMOS. It supports Cadence EDA tools for schematic and layout. But LVS, parasitic extraction and DRC are made through Mentor Caliber [21].

3.3.2 Alternative technology nodes

Each foundry provides multiple nodes of standard cells. A node is a number referencing the length of the transistor used in building the cells. When comparing between different nodes we found that smaller nodes will be faster charging and discharging so the frequency of their cell will be higher. On the other hand leakage power would be higher for smaller Vts. Also smaller nodes dissipate higher temperatures. A 45nm node was chosen as it introduces acceptable frequency with lower power dissipation which helps us in our goal.

3.3.3 Alternative EDA tools

There are different vendors in this part, there are many areas in IC design flow, not each vendor has a tool in each area as shown in the following table

ASIC flow	***Synopsys***	***Cadence***	***Mentor***	***Others***
Synthesis	Design compiler	RTL compiler	--	Talus Design
PnR	IC compiler	SoC Encounter	Olympus	Aprisa
STA	PrimeTime	ETS	--	TeKton
Power/IR	PrimePower	Voltage Storm	--	Talus Power
DFT/Scan	DFT Compiler	Encounter Test	--	Talus Design

As we see from the above table, Synopsys and Cadence have almost similar products in each area, but each one of them has an edge in specific areas, so their products are as much familiar for designers as the other. Cadence has advantage in the area of analog design so their products are well known for designers in this area. On the other hand, Synopsys dominates the digital flow by its DC, ICC, Primetime and VCS tools which are most popular for digital design in undergrad level. So we choose Synopsys tools in our project as they cover all the needed steps from RTL to GDS|| track.

3.3.4 Alternative flow options

ASIC design depends on understanding requirement specifications and EDA tools behavior. The same RTL design could be synthesized for low power or high speed or both of them with high Area. It is popular that you should choose between speed, power and area. Making optimization to get high speed would force you to use more area, also targeting low power design would force you to work with less frequency and vice versa.

These alternatives are decided by the designer and implemented in the back-end stage of ASIC design. Through our journey in developing our .tcl code, we face many alternatives to apply, each of them has some advantages and disadvantages. Our main goal is getting high speed and acceptable power and no constraint on footprint.

Examples of these alternatives:

A- Power grid layers numbers:

Routing is a main issue regarding the speed of the design, the usual choice is to use 5 metal layers for routing and 5 layers for power grid. This choice is made in our case of 10 layers only, but this still makes some DRCs and problems in speed increase.

So a proposal was made to solve this, we can take more layers for routing, for example from layer 1 to 6 for routing and from layer 7 to 10 for power grid. This change did not make further improvements. Further we increase the routing to be from layer 1 to layer 7 in one time and to layer 8 in another time. These changes have improved the routing but unfortunately increased IR drop as the allowed area for the power grid was lowered. The final version in flat design is making routing from layer 1 to layer 6, CTS from layer 7 to

layer 8 and power grid from layer 9 to layer 10. For hierarchical design from layer 1 to layer 6 used in routing and power grid from layer 7 to layer 10. Considering that the first power grid layer should have vertical straps to make it easy for the tools to deliver power to the cells in the first metal layer.

3.4 Description of the process

The main target of the project is to go through the ASIC flow from RTL to GDS going through synthesis, floorplanning, power grid synthesis, placement, Clock tree synthesis , signal routing, DRC and LVS fixing, then finally, signoff timing checking and fixing using Primetime. In this section, we shall present a brief description of each step of the process of the RTL to GDS flow. A more detailed description of the steps and options shall be provided in section 4.2.

3.4.1 Synthesis

Logic synthesis is the process in which a hardware description Language written design is converted into an optimized gate level netlist that is mapped to a certain technology library. The synthesis process in this project is done using the Design Compiler EDA tool. The synthesis process using Design Compiler can be explained in the following five steps.

1. HDL Inputting: Design files -written in using an HDL such as verilog or VHDL - are inputted to the Design Compiler tool.
2. Translation: DC uses three types of libraries: technology library, synthetic - or DesignWare - libraries and symbol libraries. The generic technology library (GTECH) consists of Flip Flops and the basic logic gates. The DesignWare library includes more complex cells like adders and comparators. Both the generic technology and the synthetic

libraries are technology independent. That means that they are not mapped into a specific technology library. The third type of libraries is the symbol library which is used by Design Compiler to generate schematics for the designs. During synthesis, DC translates HDL to components extracted from the generic technology library and DesignWare libraries.

3. Optimization and mapping: After the design translation into gates, design optimization is done. Then, DC maps the design into a target library. This process is driven by the design constraints, which are the required timing and restrictions that should be met by the synthesis process.
4. Test synthesis: Test synthesis is the process in which test logic is integrated into the design. It enables the testing of the design for early test errors resolving.
5. Finally, the design is ready for the Place and Route step. In this step, the cells are placed and cell interconnections are made. It is then possible to back-annotate the design with the actual interconnect delays. DC can then redo the synthesis for more accurate timing analysis. Redoing the synthesis while having the PnR information is basically what we call the topographical flow.

Figure 1 depicts the basic synthesis flow using the Design Compiler retrieved from the DC user guide.

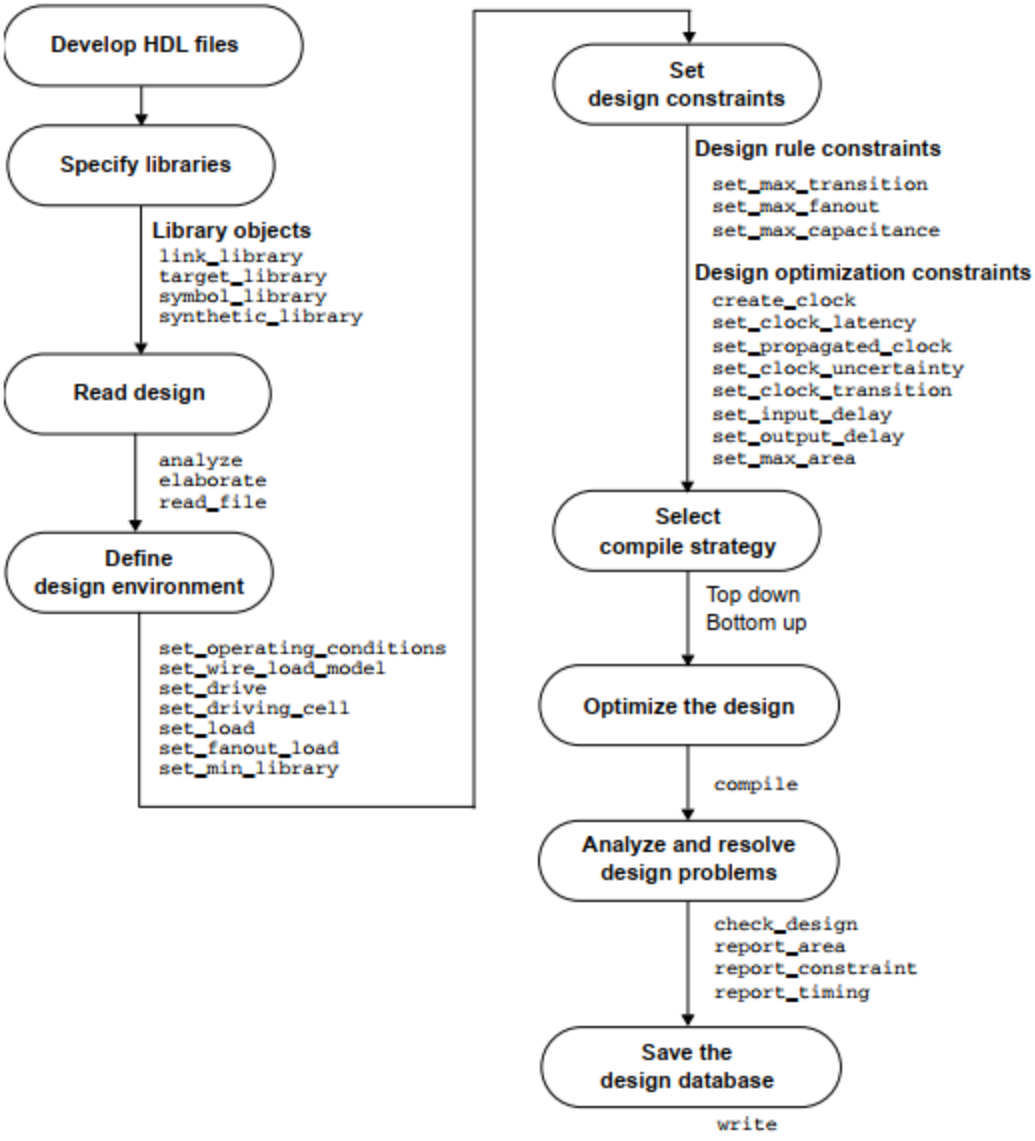

Figure 1. Synthesis flow steps[22]

The synthesis in the topographical flow is a little different from the normal flat and hierarchical flows. In those two flows, synthesis is performed only once without any prior information about the floorplanning or placement of the cells. At that point, the timing estimation is solely based on the Wire Load Model (WLM) which is a model that estimates the delay of a certain signal based on the fanout of the gate where the signal originates. Obviously, this estimation is not very accurate because this is not necessarily the only factor determining the signal delay. Other

important factors include the distance covered by the signal and the parasitics along the way. Here comes the importance of the topographical flow. In the topographical flow, a first synthesis round is conducted based on WLM as the other flow. Then, the floorplanning and power grid synthesis are performed in ICC based on the resulting netlist from the first synthesis. The floorplanning and powergrid information are taken back to the DC tool in topo mode. Using that information, the topographical model of the tool can perform placement of the netlist then estimate timing more accurately. The accurate timing estimation allows the tool to enhance its netlish and even placement in a second pass synthesis process. The process is iterative such that the cells are placed and their placement is used to estimate the timing. Then, the estimated timing is used to enhance the netlist and the placement and so on. After finishing the second synthesis, the design is moved to ICC for placement optimization only because the placement has already been done.

3.4.2 Floorplanning

The floorplan describes the core size, the shape and placement of standard cell rows and routing channels, the constraints of the standard cell placement, and the placement of peripheral I/Os, power, ground, corner and filler pad cells. Floorplanning is the first step in the PnR flow. In this project, this step is performed using Synopsys tool: IC Compiler. Before the floorplan is initialized, the design netlist should be read first and unique instances are created for modules that are instantiated many times. Creating the floor plan, the aspect ratio and the core utilizations options can be set. The aspect ratio can be chosen either to fit within a certain shape in a SoC if that is a requirement. It can also be chosen through different trials such as to minimize the congestion due to signal routes. See figure 2 for the aspect ratio example.

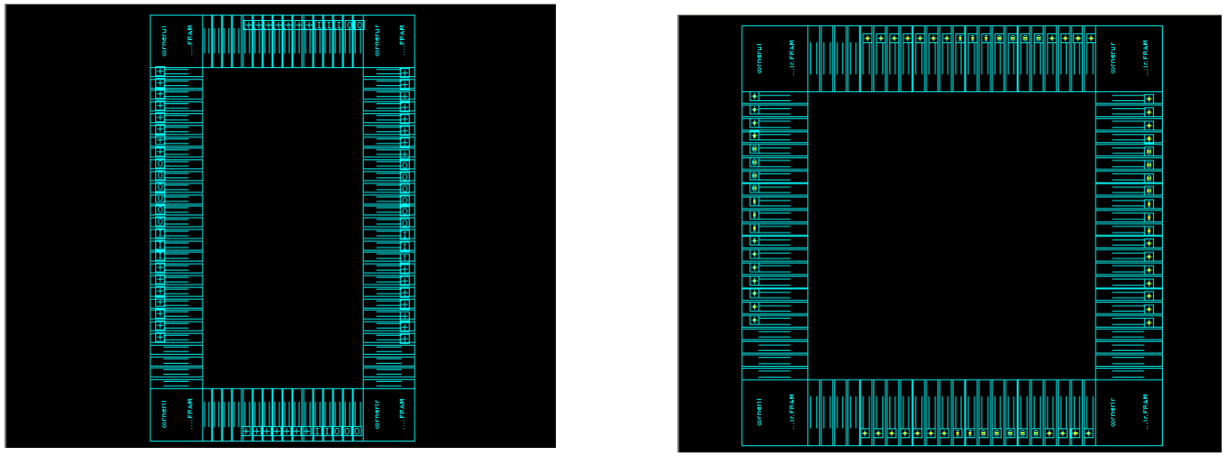

Figure 2. a)Floorplan With No Options Specified. b)Floorplan With Aspect Ratio of 2.

The core utilization represents the ratio of the cell area to the total chip area. It is usually chosen somewhere in between 0.25 and 0.35 in order to allow for successful signal routing without unwanted congestion. Moreover, the spacing left from the input/output pins to the chip core is also determined in the floorplanning step. Aslo the pairing and/or flipping of the standard cell rows can be set, see figure 2 below.

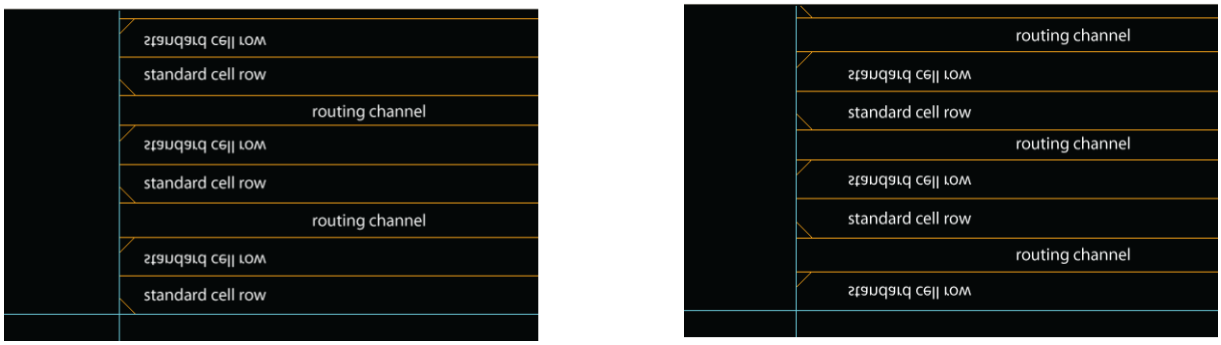

Figure 3. a) Floor Plan With Row Pairing at the Core Bottom. b)Floor Plan With First Row Flipped

3.4.3 Power Grid synthesis

ICC supports both single voltage and multi voltage designs. In this thesis we are concerned with the single voltage power network flow since the design under consideration is a single voltage design. Figure 4 shows the design flow steps for creating the power network of a single voltage design. Before running the power synthesis, the design should be saved. This is an important step since reaching a good power network satisfying the acceptable IR drop constraints might require multiple trials and iterations. Thus, if at any trial the results of the power network were not satisfying, we can simply load the saved design and restart the power synthesis step after modifying the constraints. The second step is defining logic power and ground connections. In this step, logical connections are created between the design's power and ground nets and the power pins of both standard cells and macros. Third, we apply the power rails constraints. Such constraints determine the width, spacing, offset, direction and other parameters required by ICC to create the power network straps and rings. Next, the power ring constraint should be applied. These constraints control the creation of the power rings around the macros and plangroups. Once we have all the constraints set, we need to add temporary power sources for the design which is provided by adding the virtual power pads. Virtual power pads provide the design with additional current source for the power without any additional modifications in the premade floorplan. Different trials can be made using different numbers and arrangements of the virtual power pads to determine the best choice. Then once this is determined, the floorplan shall be modified by inserting actual power pads according to the premade analysis. With all the constraints and virtual pads ready, the power grid synthesis shall be done. After the created power plan is satisfying regarding its IR drop, committing the power plan is done to convert the virtual power rings and straps into actual power and ground metal wires and vias. Finally the

power rails of the standard cells are added to connect the standard cells and a final power grid analysis is done to make a final IR drop check.

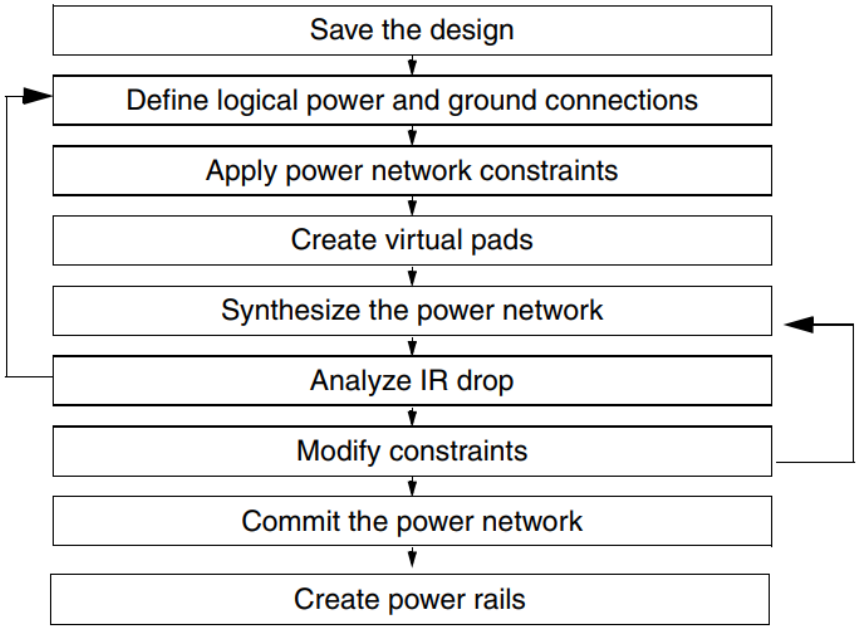

Figure 4. Power Network Synthesis for Single-Voltage Designs [23]

3.4.4 Placement

Instantaneous placement of standard cells and macros is called virtual flat placement. It represents an initial fast placement performed for the purpose of design planning. Virtual flat placement can also be used for plan groups, macros and voltage areas that have not been placed where it helps in deciding sizing, locations, and shapes of the top level physical blocks. It is called virtual placement since it considers the design to be flat temporarily. That is it temporarily ignores the hierarchies. However, after the shapes and locations of the physical blocks are

determined, the design hierarchy is restored and we proceed with the physical design flow block-by-block.

The process of the virtual flat placement can be controlled to satisfy the most crucial design properties such as timing constraints meeting, routing congestion avoidance, pin locations, hierarchical gravity, or scan chain connectivity. Since the virtual flat placement is considered a fast step, different trials could be made to explore various tradeoffs between the design variables before the floorplan finalization.

Virtual flat placement is typically done after the floorplan initialization, but before the macros and the standard cells have been placed. To perform the virtual floorplanning the following should be done first. First, the chip boundary should be defined. Second, the I/O pad cells should be placed. Third, the site rows should be defined. Finally, the timing constraints should be set and should be reasonably achievable. However, the power structure is not necessarily defined at this step, except for the I/O pads.

Virtual flat placement can be arranged into the following steps:

1. Use the set_fp_placement_strategy command to set the virtual flat placement strategy options.
2. Use the set_fp_macro_options command to set the macro placement options.
3. Required constraints setting.
4. Set the blockage, margin, and shielding options.
5.Use the create_fp_placement command to perform virtual flat placement.
6. Analyze the resulting timing and congestion. Use manual editing if required and applicable.
7. Repeat steps 1 through 6 if the results are not good enough.

3.4.5 Clock Tree Synthesis (CTS)

The design contains a great number of synchronous sequential elements due to the presence of a large number of intra-stage registers among other reasons. Hence, there is a need to implement a large clock tree with minimal skew. In this stage, we first check that the design is ready for clock tree synthesis, that is we check that the floorplan and the netlist information are available, the design is placed and the clock constraints are defined. Then, we set the clock tree synthesis options such as the maximum parasitic capacitance, the target skew value, the maximum tree fanout. In addition, we specify the design clock signal and the library gate to be used as a driving cell for the clock port. We further set the options of the clock tree synthesis process to allow for gate relocation and gate resizing as we are targeting maximum clock speed so we need to give the algorithm the maximum flexibility possible. Moreover, we specify the library gates to be used as buffers on the clock tree. Those buffers are to be used later for adjusting hold time violations. Next, the tool compiles the clock tree taking into consideration the options and information that we specified. The clock tree is implemented as an H tree starting from the clock input port. The H tree is one of the best shapes to minimize the clock skew. It depends on a very simple concept which is the equality of all paths length from starting point to all the gates connected to the clock. The clock tree routing is implemented across layers 7 and 8. The last step is to run the clock optimization algorithm to fix the hold time violations across the compiled tree by gate or buffer relocation and sizing. Then, we perform power connections to connect the newly inserted clock buffers and in case any existing cells were relocated. Finally, we run congestion and quality of results (qor) checks to track any issues with timing, power, or design rules due to clock tree synthesis. It is worth mentioning that the qor includes statistics about the

count of cells which indicates the quantity of clock tree buffers inserted during clock tree synthesis.

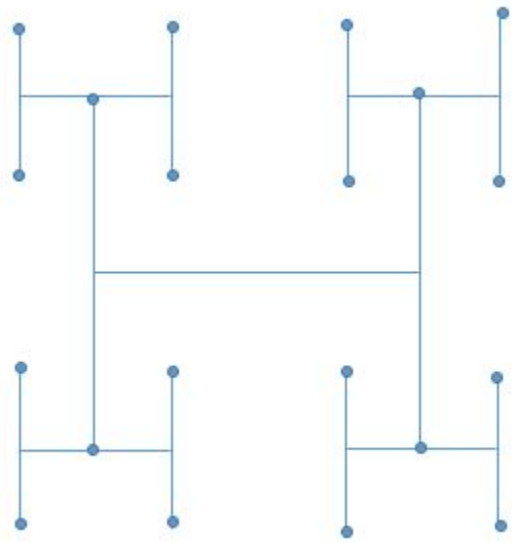

Figure 5. H tree with 16 sink points [24]

3.4.6 Routing

The routing is a complicated, time consuming, and error prone stage. So, we had to do it carefully and accurately to avoid any errors it could cause. We were targeting high speed as in the previous stages. First, we insert a number of spare cells to allow the routing algorithm to use alternative paths in case it fails to connect a specific path using the original cells. The spare cells are only NAND and NOR cells since all binary logic can be built from those two cells or even only one of them. We place only 20 spare cells to avoid high congestion. The cells are evenly distributed across the design area. Then, we check the routeability of the design in terms of pin access points, cell instance wire tracks … etc. Next, we set to routing algorithm options such that the global routing and the track assignment are timing driven, the incremental routing is enabled, and the same net notch rule is checked. We adjust the signal integrity options to minimize the crosstalk and the static noise.

The crosstalk is a common issue and affects the timing of the design. It is when signal drivers are connected to each other through a parasitic capacitor and either charge at the time or when charges and the other discharges at the time. It causes a bump and/or delay in one of the signals. Then, we run the routing algorithm with high effort to obtain the best result available. The routing is performed across the downmost six layers. It was decided not to use layers 7 and 8 and leave space for clock tree synthesis in those layers to avoid high congestion, crosstalk, and design rule violations. After the routing is finished, we run an incremental routing optimization with high effort as well to obtain better timing and to fix all violations.

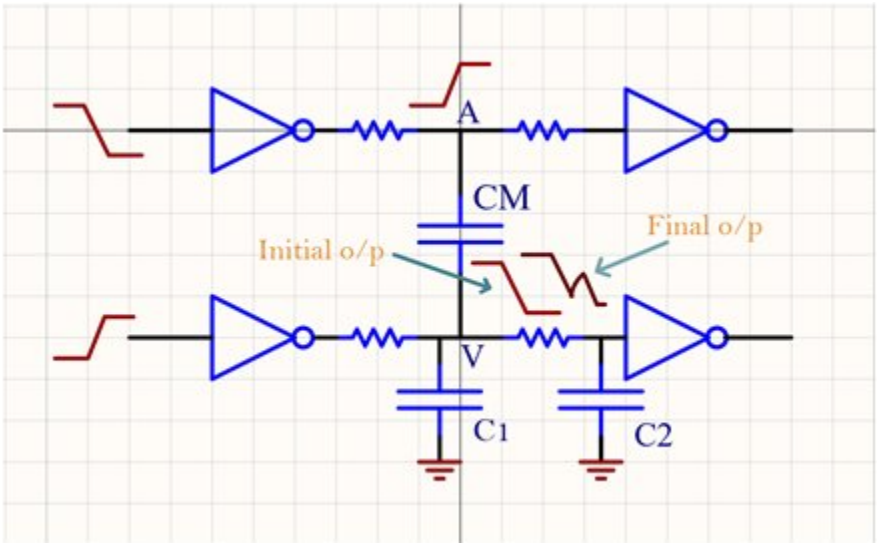

Figure 6. Aggressor and victim switching in the opposite direction [25].

3.4.7 DRC and LVS fixing

After placement, clock tree synthesis and routing, we are faced by Layout Versus Schematic (LVS) and Design Rule Check (DRC) errors. So, a number of commands are run with the aim of eliminating or minimizing those errors. At first, we focused on fixing LVS errors. We run route_eco command which searches for broken nets and fixes them. We set this command to run

20 iterations because the number of fixed errors stabilizes after this number. Further, the command is adjusted such that the algorithm attempts to fix open nets first without breaking connected nets then breaks connected nets if it fails to do so. To further improve LVS errors we run the route_zrt_eco command which,after fixing open nets, works on fixing DRC errors. The command also operates using the same methodology where the open nets are attempted without breaking any nets first then breaks nets in case of failure.

Next, we focused on fixing DRC violations. So, we used the focal_opt command which fixes DRC violations in more aggressive and expensive ways than other commands. Then, we check the LVS errors from ICC GUI. It was found that there are a number of cells not connected to power rails due to absence of vias from the preroute metals to the power rails and the absence of the preroute metals in some cases. That issue was fixed using preroute_standard_cells to fill the empty rows with metal. Then, create_preroute_vias was used to place the missing vias manually in specific locations along the newly placed preroute metal.The other LVS errors were found to be fake errors. The errors were about unused cell pins that were actually not connected in the netlist. For example, there were so many D flip flops that their negated output $\overline{Q}$ was not used. Another example is full adders whose carry out pins were not used. Further analysis is required for the DRC errors but it was not completed due to the unavailability of the IC Validator tool.

3.4.7 Signoff timing estimation and fixing using PrimeTime

If we report timing using ICC, we know that this timing estimation is not so accurate. This is why we need to report timing using Primetime which gives a more accurate estimation of the design timing. First, we extract the RC parasitics from ICC in both the maximum and the minimum estimations. Then, we go to primetime and adjust library corners such that the setup

time check is performed with the slow-slow corner and the hold time check is performed with the fast-fast corner. Using the extracted parasitics, the netlish and the selected corners, Primetime can proceed to the timing estimation step. It is worth mentioning that the minimum estimation of parasitics is used with the hold time violations checking. On the other hand, the maximum estimation of parasitics is used with the setup time checking.

After obtaining the accurate timing estimation, a negative slack is obtained in the setup time estimation. To fix that negative slack, we use the fix_eco_timing command which outputs a list of changes to the design netlist. If those changes are applied, the slack is fixed. We then take the list of changes back to ICC and apply them. Of course, there will be open nets after applying the changes. So, we use the route_eco command once again to fix them. The parasitics of the new design are extracted to be taken once again to Primetime for accurate timing estimation. The cycle of parasitic extraction, estimation, producing change list and fixing is repeated until the slack becomes positive or zero.

Afterwards, the hold time is checked for negative slack as well using the minimum parasitics estimation and the fast fast corner as mentioned above. If negative slack is found, the same cycle is repeated for the hold time violations. Normally, the hold time slack is not very large and is easier to fix than setup time slack. This is evident in the nature of changes applied to fix each of them. The changes for hold time fixing is normally insertion of buffers along the signal path to delay the signal. In setup time fixing, the signal needs to arrive earlier than it does. This is usually achieved by sizing the cells along the signal path to become faster. Sometimes, this is not enough for the signal to arrive early enough.

3.5 Block Diagram

3.5.1 Block diagrams of the RISC-V core.

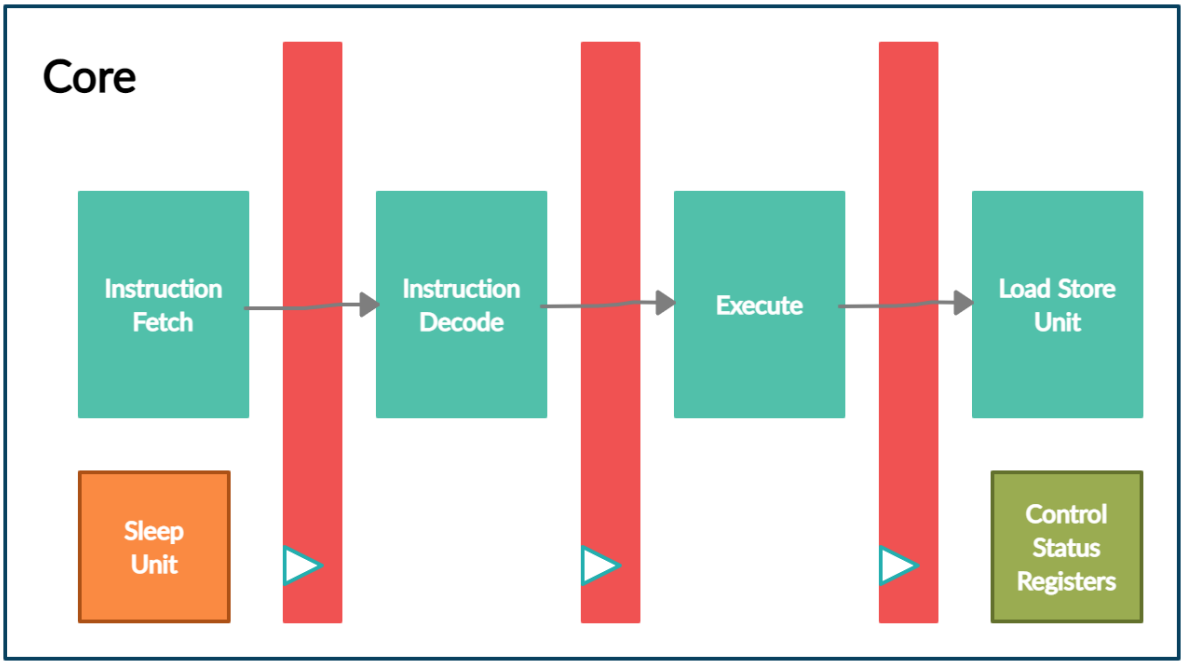

Figure 7. Simplified block diagram of the design

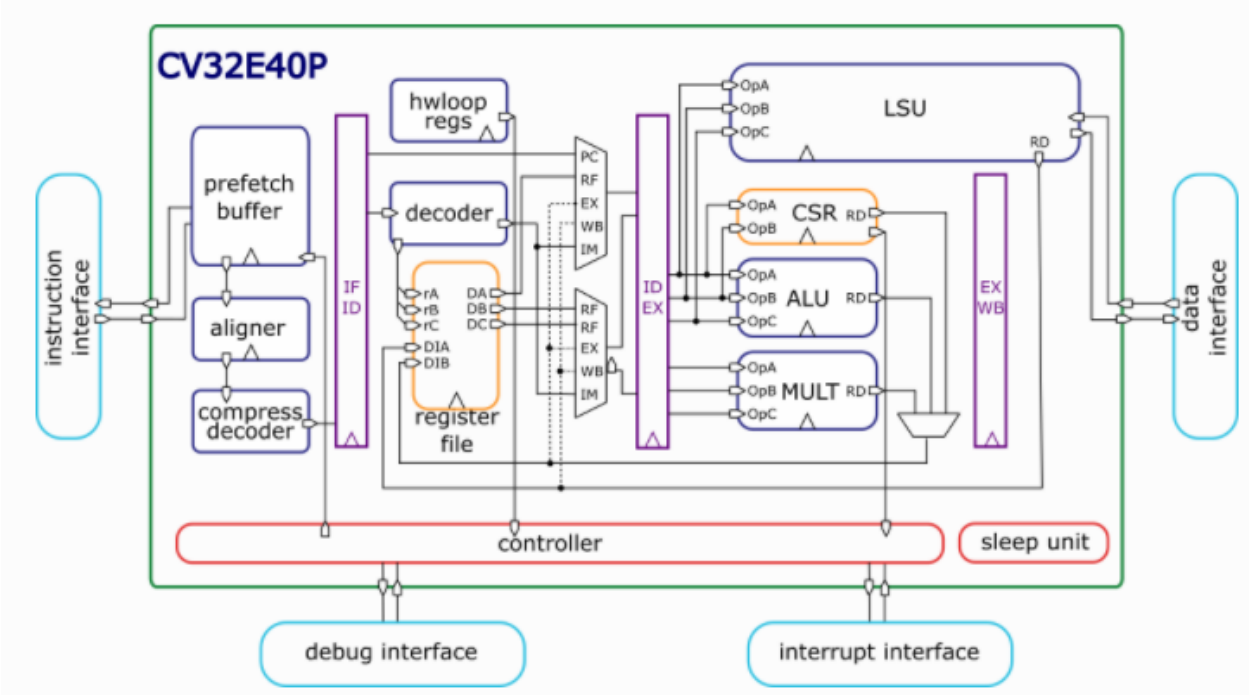

Figure 8. Block Diagram of CV32E40P RISC-V Core [26]

3.5.2 Block diagram of the ASIC flow

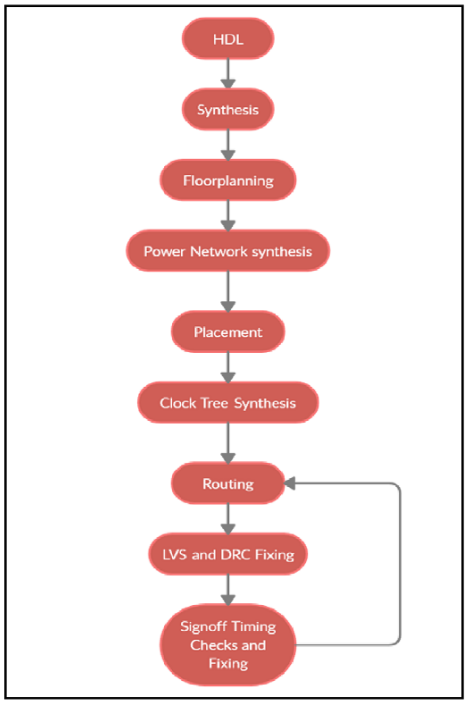

Figure 9. Block Diagram of ASIC flow

4-Project Execution

4.1 Project Tasks and Gantt chart

2 /12
- First meeting

15/12
- Reviewed openpulp RISC-V system
- We checked CNN work and Hero (both are not OK)

1/1
- Chosen RISKY 32
- Finished synthesis

12/1
- Synthesis with different options (compile_ultra) flat/hierarchy
- Optimize for speed

26/1
- Modified RTL for gated clock standard cell
- Done rough floorplanning, power, placement, and CTS
- Checked the different flavors of threshold voltages of different cells in the library
- Done dc_topo synthesis

9/2
- Flat and hierarchy: floorplanning till routing
- Modified CTS script

9/3
- Flat + hierarchy: modified routing script + still LVS&DRC errors
- topo: completed the second pass synthesis

6/4
- Topo: succeeded to extract the design from DC back to ICC
- Modified routing script to fix open and shorts
- Modified the layers for PG, CTS, and the routing layers
- done placement, CTS, and routing with a few LVS errors and slack

20/5
- PT flow completed (reg to reg path)
- Used the change list in ICC using route_eco
- Fixed sourcing SDC problem in primetime

11/6
- Finalizing the project documentation
- Report the highest frequency for all three flows with the corresponding power and area.
- Make a comparison and interpret the results to design the best flow.

4.2 Description of each subprocess

In this section, we shall describe each of the ASIC flow steps mentioned in section 3.5 while mentioning all the details of the steps including the used commands and their detailed options and functions. Figures showing the changes applied to the designs through the intermediate steps are also included as extracted from the perspective tool, either DC or ICC. The upcoming steps are the same for both the hierarchical and flat flows. The topographical flow has some differences that shall be highlighted after the placement section.

4.2.1 Synthesis

The following steps represent the basic synthesis flow:

1. Developing HDL files

As mentioned earlier, the design files inputted to Design Compiler are generally written in VHDL or verilog. Such design description files should be carefully written in order to achieve the best results possible after synthesis. Certain considerations should be taken into account while describing the design such as coding style, design partitioning and data management.

Despite the fact that the HDL writing is not a part of DC flow, it directly affects the synthesis and the optimization process and results.

2. Libraries specification

Link, target, symbol, and synthetic libraries are specified for Design Compiler using the commands link_library, target_library, symbol_library, and synthetic_library respectively.

The most important two libraries are the link and target libraries. These are technology libraries that define the set of cells of a semiconductor vendor and related information, including cell names, cell pin names, pin loading, delay arcs, operating conditions and design rules. Without these two the synthesis process cannot proceed. The symbol library is only needed if using the Design Vision GUI is required. This library defines the symbols used for schematic viewing of

the design in the GUI. In addition, the synthetic_library command is used to specify any specially licensed DesignWare libraries. The standard DesignWare library need not be specified.

3. Reading the design:

Both RTL designs and gate-level netlists can be read by Design Compiler. Verilog and VHDL RTL designs are read by HDL Compiler of Design Compiler. Design Compiler has a specialized netlist reader for Verilog and VHDL gate-level netlists reading. The specialized netlist reader is faster and utilizes less memory than HDL Compiler. Reading design files can be done by Design Compiler using the following ways:

• The *analyze* and *elaborate* commands

• The *read_file* command

• The *read_vhdl* and *read_verilog* commands. These commands are derived from the *read_file -format VHDL* and *read_file -format verilog* commands.

In this project we used the first way which is using the analyze and elaborate commands. The analyze command translates the specified HDL files into an intermediate format and stores the intermediate format in the specified library ready to elaborate as needed to link a full design. This command is given the work library that will receive the analyzed output files using the command option [-library], the format of the input design file using the option [-format] and finally the design file to be analyzed. The *elaborate* command uses the intermediate format to build the design. It is given the design name, and the ;library name to which the work should be mapped using the option [-library].

4. Defining the design environment

It is required that the environment of the design to be synthesized should be modelled. This model includes the external operating conditions (temperature, manufacturing process, and voltage), fanouts, loads, drives, and wire load models. Such modelling directly influences design synthesis process and optimization results. You define the design environment by using the set commands listed below as shown in figure 1.

set_operating_conditions
set_wire_load_model
set_drive
set_driving_cell
set_load
set_fanout_load

5. Set design constraints

Design rules and optimization constraints are used by the design compiler to control the synthesis of the design. Design rules specify technology requirements cannot be violated to ensure that the product will meet the required specifications and work as intended. Such rules should be available in the vendor technology library. Typical design rules add constraints on transition times using set_max_transition command, fanout loads using set_max_fanout command, and capacitances using set_max_capacitance command.

Optimization constraints determine the design targets for timing (clocks, clock skews, input delays, and output delays) and area (maximum area). During optimizations, Design Compiler tries to meet the given goals while meeting the design rules. You define these constraints by using commands such as create_clock, set_clock_latency, set_propagated_clock, set_clock_uncertainty, set_clock_transition, set_input_delay, set_max_area, set_output_delay. For correct optimizations , realistic constraints should be set. In our project, all the required

constraints were already set using the above commands in a constraints file that we only needed to source. We only needed to change the clock constraints in order to discover the minimum clock period that could be achieved in each of the three design flows. We did not need to modify other timing constraints since the constraints were already defined as a function of the clock period. So we only changed the clock period and everything changed accordingly.

6. Select Compile Strategy

There are two basic compile strategies that can be used to optimize hierarchical designs: top down and bottom up. In the top-down strategy, Design Compiler compiles the top-level design and all its sub designs together. Environment settings and constraint settings are thus defined with respect to the top-level design. Despite the fact that this strategy automatically considers the interblock dependencies, the method is non practical for large designs since all designs must exist in memory at the same time which requires a very large memory.

In the bottom-up strategy, individual subdesigns are compiled separately while each has its own constraints. After successful compilation of subdesigns, the designs are assigned the dont_touch attribute which prevents further changes to them during the upcoming compile phases. The compiled subdesigns are gathered to build the designs of the next higher level, and these designs are compiled. This process of compilation continues through the hierarchy until the top-level design is synthesized. Using this method allows for large designs compilation since the Design Compiler is not required to load all the uncompiled subdesigns at once into the memory. The drawback of this step is that at each stage, the interblock constraints should be estimated which may require compilation iterations to get good estimates.

7. Optimize the Design

Invoking the Design Compiler synthesis and optimization processes is done via either the compile command or the compile_ultra command. For the compile command several options are available regarding the degree of optimizations done. The map_effort option can be set to low, medium, or high. In a primary compile, it is only required to get a quick idea of design performance and area. Thus, setting map_effort to low can be sufficient for this purpose. In a normal compile, when performing design exploration, medium map_effort option can be used. The final compile in a design implementation can be done by setting map_effort to high which results in a CPU intensive compile process.

Complie_ultra command on the other hand, performs a high-effort compile process on the design for better quality of results (QoR) .This command targets high-performance designs with very critical timing constraints. It provides a simple approach to achieve critical delay optimization. By default, two ungrouping phases for design hierarchies are included in compile_ultra command. The first phase is done before "Pass1 Mapping" and tries to ungroup small design hierarchies. This phase can be turned off using: set compile_ultra_ungroup_small_hierarchies false command. The second ungrouping phase is done during "Mapping Optimization" and applies a delay-based ungrouping strategy for design hierarchies. Variables can be set to control the second ungrouping phase. To preserve all design hierarchies, use the -no_autoungroup option. In our project, we have used the [-no_auto_ungroup] option in the hierarchical flow to keep the design hierarchies. On the other hand, for the flat flow, we left the default option of auto ungrouping all the hierarchies.

Another important option in the compile_ultra command is the [-spg] option. This option enables physical guidance and congestion optimization. It is used when a floorplanning and placement information is available for Design Compiler which can be used to perform a physically guided synthesis process. Congestion optimization is effective in reducing routing-related congestion. Design Compiler Graphical uses physical guidance to save coarse placement information and pass it to IC Compiler which can begin the implementation flow by optimizing the placement using the place_opt command. Running commands such as create_placement, remove_buffer_tree, or psynopt to create placement is then no longer needed by IC Compiler. Instead, it uses the Design Compiler coarse placement as a starting point which improves placement, runtime and area correlation.

In our project, we used the compile ultra command with all the timing optimization options [-timing_high_effort_script -retime] since we were targeting high speed. The target period was initially 3ns. We also used the [-gate_clock] options to reduce the power consumption whenever possible.

8. Analyze and Resolve Design Problems

In this step, multiple reports are generated to check the design and to generate the required reports showing the results of the synthesis and optimization process. These reports include: area report, power report, cell count report, and most importantly, the timing report.

9. Save the Design Database

The final step is to use the write command to save the post synthesis design.

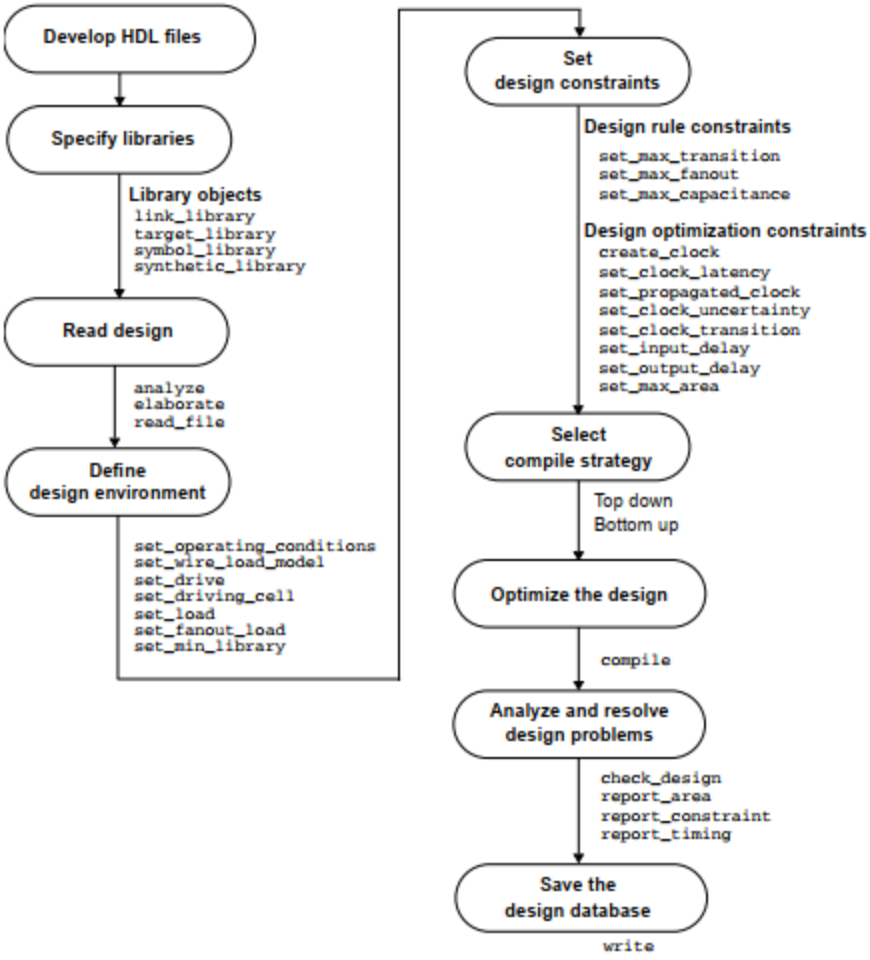

Figure 10. Post Synthesis Steps

4.2.2 Floorplanning

Creating the floorplan in IC Compiler is done via the command create_floorplan. This command has many options including [-control_type] which determines how the command sizes the core area of the floorplan. The control type can either be the aspect ratio, width and height, or boundary. In our case, we chose the control type to be the aspect ratio and we set it to be 2. The core utilization is another option that determines the ratio of the cell area to the core area. It is set to be 0.25 to avoid wire congestion while routing. [-start_first_row] and [-flip_first_row]

options to begin row pairing at the bottom of the core area, and flip the first row at the bottom of the floorplan respectively. The spacing from the io to the core is also determined. See figure 11 for the result of the floorplan step. Course placement is sometimes done in this step using the command create_floorplan_placement.

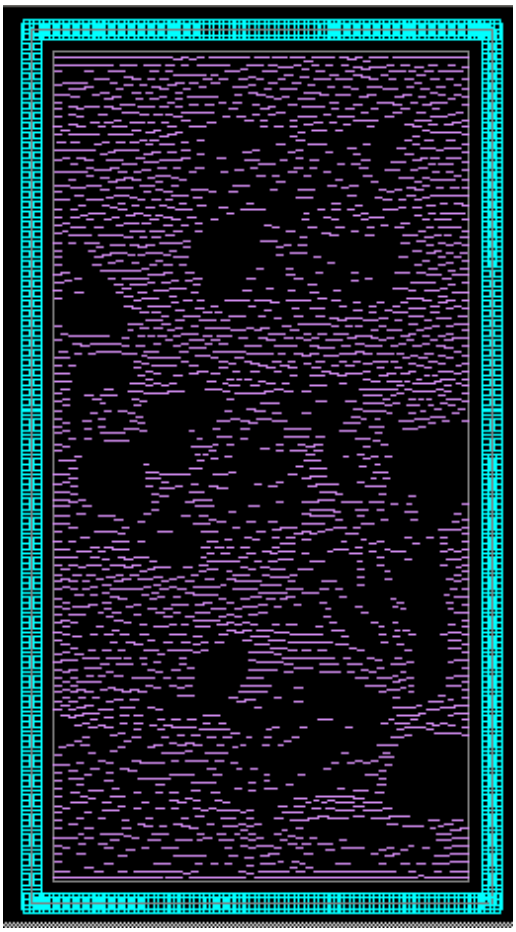

Figure 11. After create_floorplan command and initial coarse placement

4.2.3 Power Grid Synthesis

As mentioned above in section 3.5, the power grid network step is done in many steps. First, the design is saved using the command save_mw_cel such that to allow for multiple iterations whenever required until a satisfactory IR drop is achieved. Next, the power and ground connections are defined using the command derive_pg_connection which connects the power

and ground nets to power and ground pins of standard cells. Then, the power rail constraints are applied using the set_fp_rail_constraints command. This command determines the power straps and power ring constraints including spacing, width, offset, metal layers and strap directions. In our design, we set the power ring constraints such that the power rings are in metal layers 7 upto 10. The ring spacing is 0.8, the ring width is 5 and the ring offset is 0.8. On the other hand, the power strap related constraints are used to set the power strap layers to be metal6 upto metal10 for the hierarchical flow while for the other two flows, the uppermost four layers are set for power straps. The lowermost power layer should be vertical such that to cross the horizontal power rails of the standard cells and thus can be connected by vias. The maximum number of straps is 128 and the minimum number of straps is 20. The spacing is set to minimum in all layers. After setting the constraints, virtual pads are added using the command create_fp_virtual_pad or from ICC GUI. The positions of the virtual pads are set to be enough in number and to be uniformly distributed around the design such that it would mimic the action of actual floorplan power pads. Now, the design is ready for the power network synthesis via the synthesize_fp_rail command. This command creates virtual power straps and rings and analyzes and reports their IR drops. See figure 12 for the design IR drop after synthesis. Once the power grid synthesis step shows satisfactory power plan, the commit_fp_rail command is used to convert the virtual straps and rings to actual straps and rings with defined metal layers. Figure 13 shows the design after the commitment step.

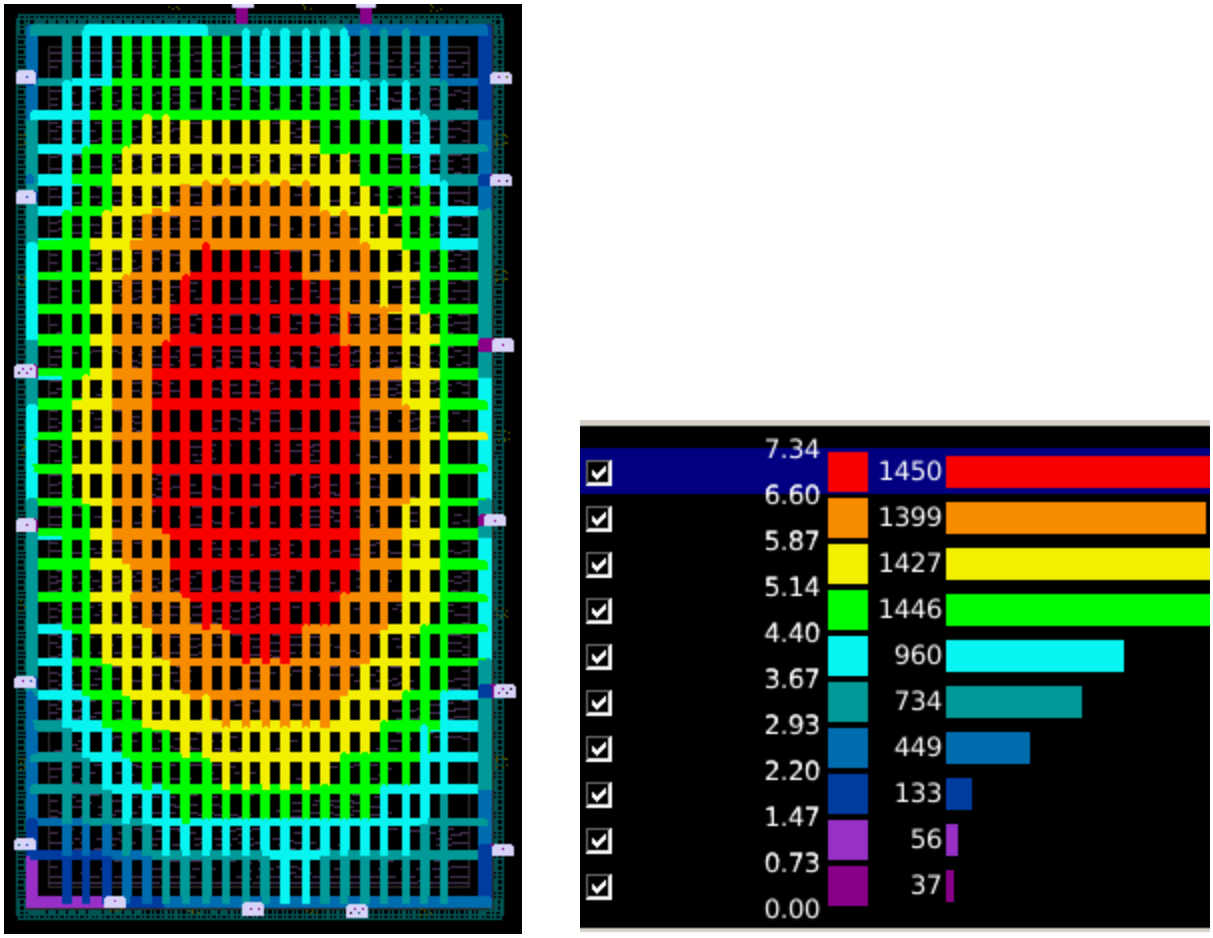

Figure 12. IR drop after synthesis of power network. Maximum IR drom is much less than the maximum threshold set.

Finally the power rails of the standard cells are added using the command preroute_standard_cells. This command adds the metal1 rails that connect power and ground ports of standard cells. The command analyze_fp_rail is then used to make sure that these newly added metal1 rails still meet the IR drop constraints.

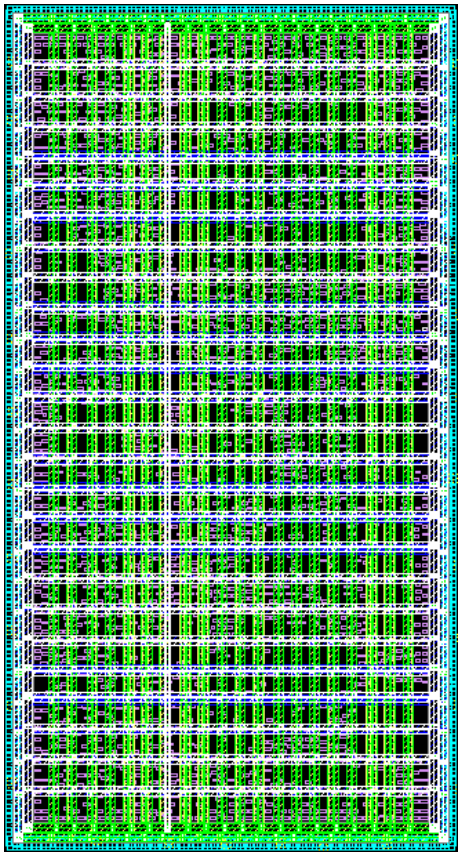

Figure 13. Power grid is committed to the design.

4.2.4 Placement

The main two commands in the placement step are create_floorplan_placement and place_opt. Create_fp_placement command performs a virtual flat placement of standard cells and hard macros. It provides an initial placement for floorplan creation which determines relative locations and shapes of the top-level physical blocks of a flat design with no placed plan groups or voltage areas. The used options for this command in our project are: [-effort high], [-congestion_driven] and [-timing_driven] in order to use extensive CPU computations to place the cells while meeting the timing constraints and avoiding wire congestions. [-incremental all] option can also be used to refine the placement given that an initial placement existed. On the

other hand, place_opt command performs simultaneous placement, routing, and optimization on the current design.

What makes the topographical flow different, however, is that the placement step is not done in IC Compiler as in the case of flat and hierarchical flows. In the topographical flow, a first pass synthesis is done in Design Compiler followed by the floorplanning and power grid synthesis in IC Compiler. After that, a def file and a floorplan file is written in ICC using the write_def and write_floorplan commands respectively. Then, we return back to Design Compiler and extract the physical constraints from the def file using the command extract_physical_constraints. We also read the floorplan using the read_floorplan command. A second pass synthesis is then done using the compile_ultra command with the [-spg] option. This command tells the Design Compiler to create the physically guided cell placement. The final placement is then written again using write_def and write_floorplan commands. The ddc file is also written in this step. We then return to ICC, read the def file and the ddc file. Now we have the cells placed by DC and we only need to perform placement optimizations using place_opt command with the [-spg] option set.

4.2.5 Clock tree Synthesis (CTS)

Clock tree synthesis is a very important and critical stage. It involves checking the design readiness for CTS, specifying the CTS options, compiling the clock tree and optimizing it. The first step is performed using the command check_physical_design -stage pre_clock_opt. This command, depending on which option is used after it, can be used to check the readiness of the design for placement, CTS, or routing. In the CTS stage, the option -stage is given the value pre_clock_opt to indicate that we need to check that the floorplan and the netlist information are

available. Further, the command checks that the cells are placed and the clock constraints are specified.

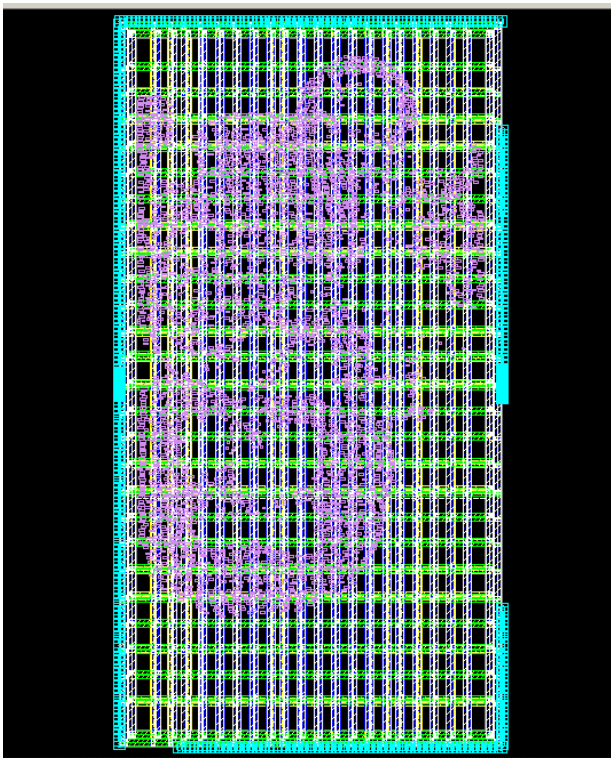

Figure 14. Design before CTS in topographical flow

The next used command is set_driving_cell which is used to specify which buffer cell drives the clock input port. It was decided to choose the BUF_X16 cell which is one of the strongest buffers available in the library yet does not span too large an area. The next step was to specify the target performance metrics of the clock tree using the set_clock_tree_options command as shown in the table below.

Target early delay	0.1
Target skew	0.5
Maximum parasitic capacitance	300

Maximum fanout	10
Maximum transition	0.3

The early delay is the time taken by the clock signal to travel from the clock port to the end of the longest path in the tree. The skew is the maximum time difference between the clock edge at two different points on the tree. The transition time specifies the delay of the buffers and inverters used on the clock tree. Then, the same command was used to allow the CTS algorithm to relocate and resize both gates and clock buffers if needed. In addition, the cell to be used as clock buffers along the tree is specified using the set_clock_tree_references command. The CTS itself is run using the compile_clock_tree. This command also fixes DRC violations after clock tree synthesis. The last command, clock_opt -fix_hold_all_clocks -congestion, is used with the aim of optimizing the clock tree to fix the hold time violations by gate or buffer relocation or sizing. The -congestion option is used to take congestion into consideration while replacing cells for clock optimization.

4.2.6 Routing

As mentioned in section 3.4, the routing stage is delicate and error prone. The first command in this stage is the insert_spare_cells command. This command is used to evenly distribute a number of spare cells across the design area to be used by the routing algorithm if needed. The routing algorithm sometimes finds difficulty in routing the existing cells so it uses alternative routing tracks with spare cells. The insert_spare_cells command is used with only 20 instances of NAND and NOR cells. The option -tie is used to tie their inputs to the ground. Then, the routeability of the design is checked in terms of pin access points, cell instance wire tracks … etc using the command check_physical_design -stage pre_route_opt. Next, we use the

set_route_options command to adjust the routing algorithm options such that the global routing and the track assignment are timing driven, the incremental routing is enabled, and the same net notch rule is checked. We also adjust the signal integrity options to minimize the crosstalk and the static noise. Then, we use the route_auto command with high effort to do the actual routing. The reason for choosing high effort is that we would sacrifice runtime for better results in terms of errors and timing. The route_auto command involves three ordered stages. The first stage is the global routing where the tool creates a three dimensional array of global routing cells and assigns unrouted nets to those cells. The cells are used to estimate the demand and capacity of global routing. The second stage is the track assignment where the tool assigns actual wire tracks within routing cells to the unrouted nets all over the design at once. When this stage is completed, the design is already routed but there are many violations. Those violations are fixed thoroughly in the detailed routing stage. Then, we run the route_opt command, with high effort and incremental mode, to optimize the routing and fix more violations. With the incremental mode enabled, the route_opt command does not redo the routing but only optimizes the existing routing.

4.2.7 DRC and LVS fixing

After routing, we need to fix Layout Versus Schematic (LVS) and Design Rule Check (DRC) errors. We start by fixing LVS errors. The route_eco command is run. This command searches for broken nets and fixes them. It is set to run 20 iterations because the number of fixed errors does not change after this number of iterations. Moreover, the command is adjusted such that it attempts to fix open nets first without breaking connected nets then breaks connected nets if it fails. To fix more LVS errors, we run the route_zrt_eco command which,after fixing open nets,

works on fixing DRC errors. The command also operates using the same methodology of route_eco that we mentioned.

Next, we moved to fixing DRC violations. The focal_opt command was used because it fixes DRC violations in more powerful and costly ways. Then, we view the LVS errors from ICC GUI. It was found that there are some cells unconnected to power rails due to absence of vias from the preroute metals to the power rails which lead to the absence of some preroute metals as shown in the figure 15 below. We used the preroute_standard_cells command to fill the empty rows with metal. The create_preroute_vias command can create vias from any layer to any layer. It was used to place the missing vias manually in well distributed locations along the new preroute metal. Further analysis is required for the DRC errors but it was not achieved due to the unavailability of the IC Validator tool.

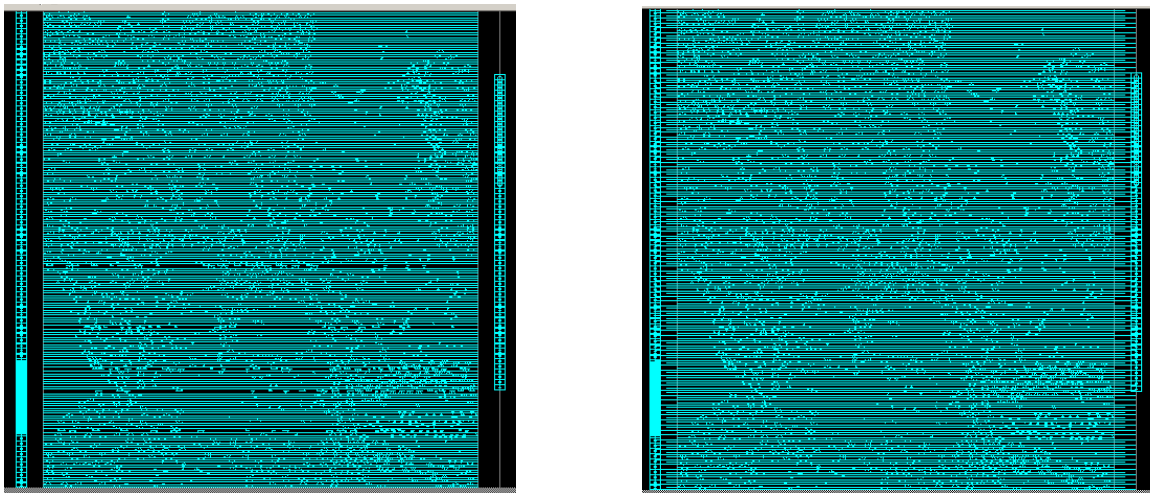

Figure 15. a) Absence of some preroutes b) All preroutes after fixing

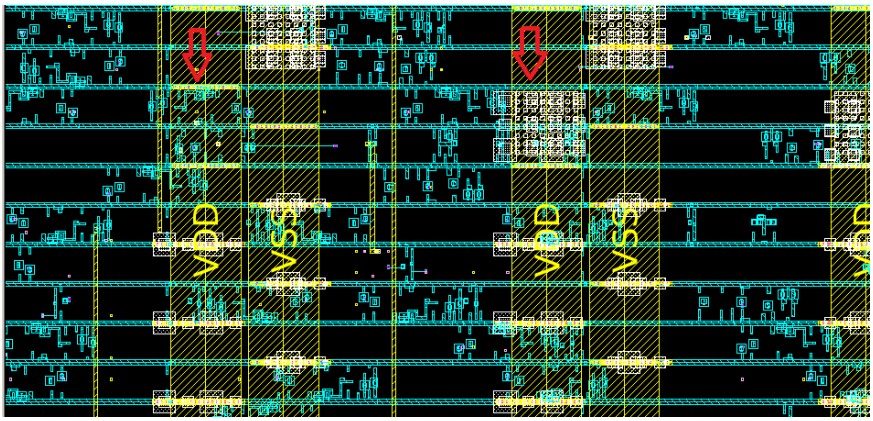

Figure 16. Absence of vias from preroute to power rails

4.2.8 Primetime

As we mentioned in section 3.4, it is crucial to make an accurate estimation of timing which is not available in ICC. So, we go to Primetime for an accurate timing estimation. Before that, we extract the RC parasitics from ICC in both the maximum and the minimum estimations using the extract_rc command which extracts parasitic resistance and capacitance from the routes of the design. Then , we use the write_parasitics command to write the parasitics of the design in two SPEF format files. One file is for the minimum estimation and the other is for the maximum estimation. Then, we go to primetime and adjust library corners such that the setup time check is performed with the slow-slow corner and the hold time check is performed with the fast-fast corner. The extracted parasitics from ICC are sourced using the read_parasitics command. Using the extracted parasitics, the netlish and the selected corners, Primetime can proceed to the timing estimation step with the update_timing command followed by the report_timing command to display the timing information.

After obtaining the accurate timing estimation, a negative slack is obtained in the setup time estimation. To fix that negative slack, we use the fix_eco_timing command which produces a list of changes to the design netlist. If those changes are made, the slack is fixed. The changes for

hold time fixing is insertion of buffers on the signal route to delay the signal. In setup time fixing, the signal needs to arrive earlier. This is usually achieved by resizing the cells on the signal route to make them faster. There will be open nets after applying the changes. So, we use the route_eco command once again to fix them. The cycle of parasitic extraction, estimation, producing change list and fixing is repeated until the slack becomes positive or zero. Finally, the hold time is checked for negative slack. If negative slack occurs, the same cycle is repeated for the hold time violations.

4.3 Project Testing and Evaluation

One of the main aims of this project, in addition to satisfying the constraints, was to compare the three ASIC flows against each other. In this section, we are going to show the obtained results in all flows and attempt to reach a conclusion about which of them is better in terms of total area, total power, timing and congestion.

4.3.1 Topographical flow

As previously explained, the topographical flow advantage over the other flows is the accurate estimation of timing during synthesis step. This accurate estimation allows a higher degree of optimizing the netlist and the placement in an iterative manner to satisfy harder constraints. That is why we expect the topographical flow to meet the timing constraint that we set. We initially identified our target clock period to be 2 ns which the topographical flow easily accomplished. We decided to go even further and try a clock period of 1.8 ns which the topographical flow also achieved. Concerning the area, the topographical flow achieved a total chip area of 136,938. Because of the high frequency, the design exhibited high dynamic power consumption which contributed to a high overall power consumption of 2.42 mW. The congestion and the IR drop requirements were also satisfied. For a fair comparison with other flows, we need to calculate the

power delay product (PDP) for the topographical flow. The PDP for the topographical flow was found to be 4.356 × 10 ^ -12. The LVS errors were all fixed except for 12 signals that were shorted to ground most probably because they were assigned a zero value in the RTL.

.4.3.2 Flat flow

Speed

In comparison with hierarchical flow, flat flow could achieve optimal speed performance. It could achieve exactly **2.3ns** in its clock period; corresponding to approximately 435 MHz for its clock frequency. The reason for this higher speed is that flat design flow is not constrained by the hierarchy of design; in other words, it does not preserve the hierarchy of every punch of blocks. Instead, it makes all design blocks implemented without use of a certain block more than once. Therefore, flat flow has a higher speed level than hierarchical one. Figure 17 shows that exact clock achieved after many optimizations.

```
clock clk_i (rise edge)                                  2.45       2.45
clock network delay (ideal)                              0.00       2.45
id_stage_i/register_file_i/mem_reg[4][31]/CK (DFFS_X1)
                                                         0.00       2.45 r
library setup time                                      -0.03       2.42
data required time                                                  2.42
--------------------------------------------------------------------------
data required time                                                  2.42
data arrival time                                                  -2.42
--------------------------------------------------------------------------
slack (MET)                                                         0.00
```

Figure 17. Clock period of flat design

Area

Since flat flow is based on implementation of all blocks and sub-blocks without preserving hierarchy, the area of the flow is bigger than that of the hierarchical one. The chip area of the flattened design is nearly 0.137mm^2 with area of standard cells of 0.0297mm^2 and the rest of the chip is for the interconnect for cells. Figure 18 and 19 show the area of the whole chip, and figure 20 shows the area of the standard cells used.

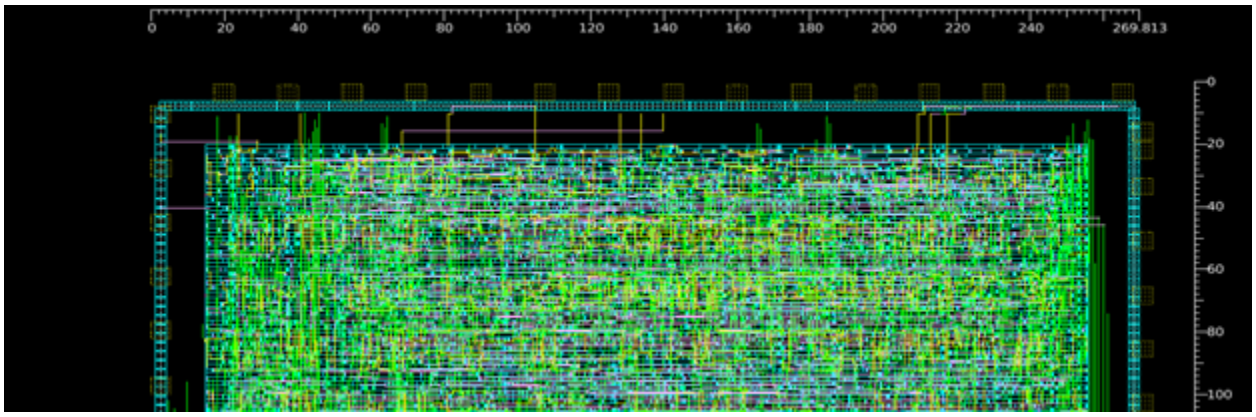

Figure 18. Chip width

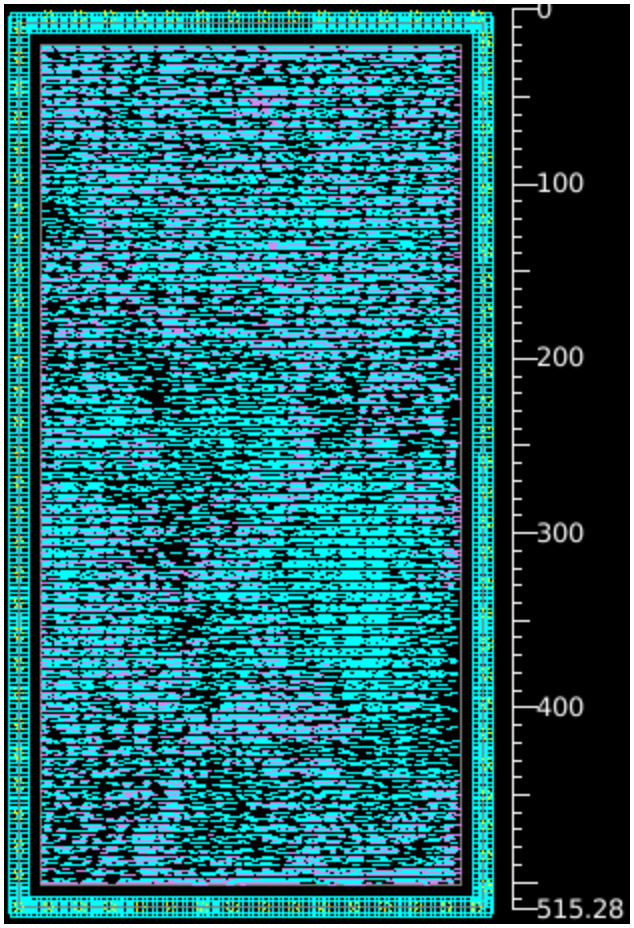

Figure 19. Chip Length

```
Combinational area:              17917.760125
Buf/Inv area:                    2015.216000
Noncombinational area:           11803.218352
Net Interconnect area:             undefined

Total cell area:                 29720.978477
```

Figure 20. Area of standard cells

Power

Power consumption of the flattened design is observed to be higher than that of the hierarchical one. The reason for that is that the flat design implements all design blocks and sub-blocks and there is no room for sharing a certain block more than once in the same design. The overall power consumption of the flat flow is 1.8mW. Figure 21 illustrates the detailed results of the flat design.

```
                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000             0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000             0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000             0.0000  (   0.00%)
clock_network    237.1123         277.5827         2.7474e+03          517.4425  (  29.24%)
register         513.1633          32.9058         3.9547e+04          585.6149  (  33.10%)
sequential         0.0000           0.0000             0.0000            0.0000  (   0.00%)
combinational    164.4967         375.1678         1.2666e+05          666.3162  (  37.66%)
--------------------------------------------------------------------------------------------------
Total            914.7723 uW      685.6562 uW      1.6895e+05 nW      1.7694e+03 uW
1
```

Figure 21. Power Analysis of Flat Design

4.3.3 Hierarchical Flow results

The hierarchical flow is expected to be slower than the other two flows since it does not provide enough optimization in order to keep the design hierarchy. For the timing results of the hierarchical flow, the design could not work at the required period of 2 ns. A large negative slack existed and could not be fixed by the PrimeTime change list. Since the available optimizations techniques could not fix the timing of the design the clock period of the design is increased to determine the smallest period that can be achieved using this flow. The achieved period was 3.3 ns which is larger than the other two flows as expected. The final achieved total area 128,950 square micrometers. Total power achieved was 1.3108 mW.

5- Cost Analysis

- Our project:

We are doing the ASIC design of RISCY as an IP. We used openPDK45n which is an educational open-source library, free to use. The tools used are DC, ICC, and PT, all are educationally licensed. Tools cost is $240 per year According to Synopsys Academic Program, the core can not be contained in a chip because there are no I/O cells to connect it to the outside world during the packaging phase. Also, the design can not be fabricated because openPDK45n can not be fabricated. So, the final design can be considered as a hard IP [27].

- Industrial:

If the target is to do the ASIC design of the RISCY as a chip. The tools needed will be DC, ICC or ICC2, PT, and ICV.

- Tools cost:

The cost depends on the time and the complexity of the design. Also, we may need other tools such as Formality. Yet, on average it will be $30K-$100K. Unofficial estimates are available at [28]

Library cost: The cost depends on the library technology and quality.

- Impact :

There is a demand for processors in security, AI, Graphics, and high-performance computing fields. Science Risc-V processor is open-source and has ints Instruction Set Architectures. So, there is a potential to contribute to many research projects such as Nutshell: A Linux Compatible RISC-V Processor Chinese group. Their target is to have a Linux-compatible RISC-V processor

supporting both the SV39 virtual-memory system and the RV64MAC instruction extension. As a result, European Processor Initiative (EPI) is developing a RISC-V processor which will lead to high-performance and low-power processing units [29] .

Since RISCY is a small processor, it most fits low-power applications. This impacts the IoT field. There is a recent trend which is IoT security. Since most IoT systems' security have limited capabilities, and there is a potential for personal data attack. RISC-V can guarantee data and programs confidentiality, integrity, and availability. Also, it has two advantages over traditional security systems. RISC-V ISA is open instead of the IPs that lead to fragmented and expensive design. In addition, IPs result in rigid and complex software implementation as there are many IP vendors. As a result, RISC-V will lead this field [30].

- Ethics:

all RTL designs have no copyrights and are free to use. Also, the tools are licensed. So, there are not unethical aspects.

- Sustainability:

There are two points that will be considered in this design and shall be considered in any AISC design project which are electromigration and aging. They are part of the full reliability solutions for the entire product lifetime. The two chosen points are under the wear-out failures section [31]

).

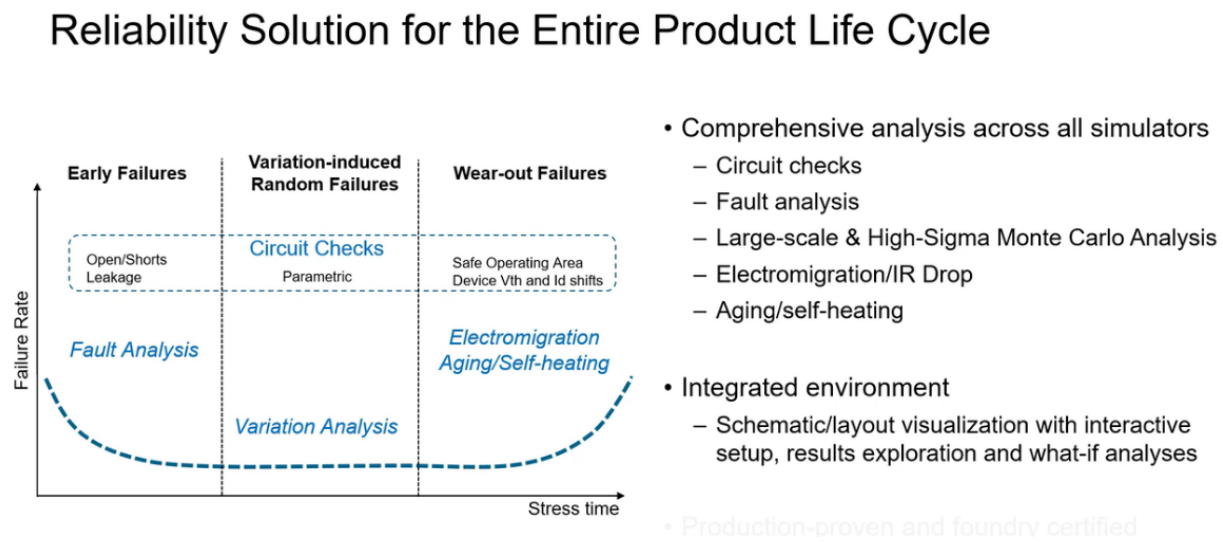

Figure 22. Full product life cycle [31]

Firstly, electromigration is the movement of the atoms based on the circuit current. If it is high, the heat dissipation will break atoms from the material causing vacancies and deposits which

may result in open or short circuits.

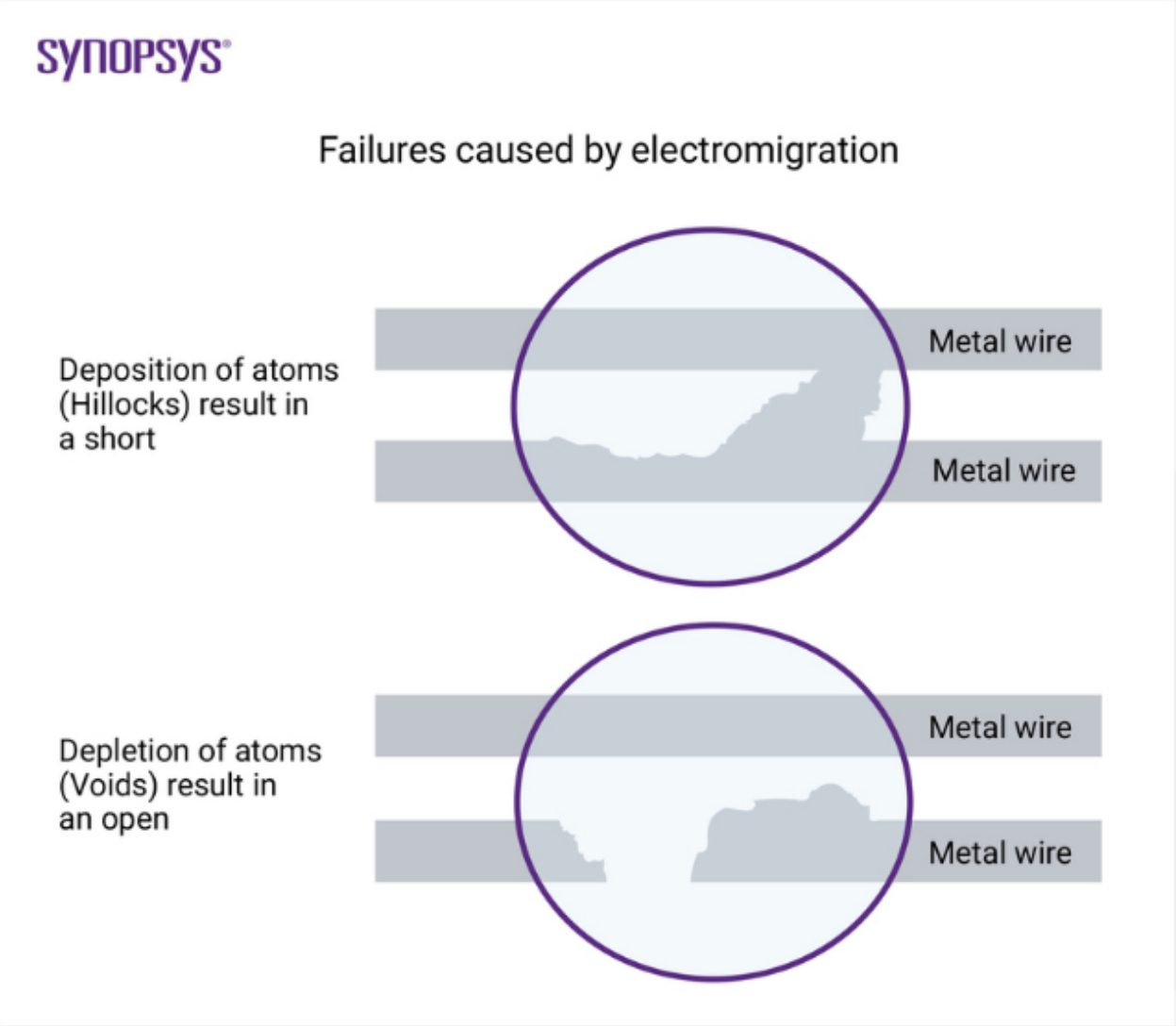

Figure 23. Electromigration [32]

TO solve this problem, the wire width should be large enough to hold the circuit current. There are 4 EM constraints depending on the current type chosen such as absolute, average, peak, and root-mean-square currents. The most critical current is the peak current, so the circuit should be able to handle the circuit max current. Also, EM depends on many factors such as wire material, temperature, and size [32].

Secondly, aging is defined as a time-dependent degradation of the electrical properties in IC. In advanced technologies, the electrical behaviors of transistors deviate from original intended behaviors during the chip's lifetime. This causes the circuit to degrade the chip performance until the chip fails to do its required function.

Aging can be analyzed using many models. Also, Synopsys offers a tool to analyze aging.

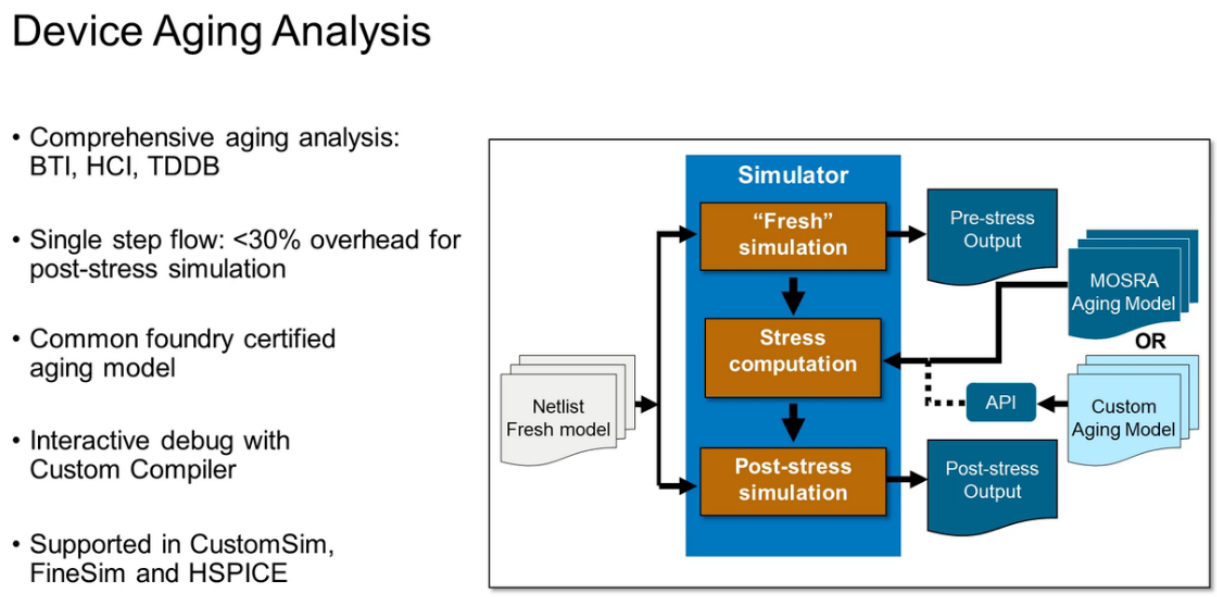

Figure 24. Device aging model [33]

The aging analysis is important to estimate the lifetime of the commercial products from the manufacturing company.

6- Conclusion and Future Work

The topographical flow achieves lowest timing as expected. The hierarchical flow achieves the smallest area. It is intended to package the RISCY hard IP so that it can be fabricated on a chip. Currently, Fabricating the design is not possible because the used library does not have I/O cells. I/O cells are the cells that connect the chip to the outside world. Those cells are required only if the target is to tap out the design. Also, the used FreePDK45 is not an industrial library, so its standard cells can not be fabricated.

So, to proceed to the tap-out phase, the library shall be replaced with an industrial library containing I/O cells. In addition, a packaging tool should be used. Synopsys has 3DIC Compiler tool which can do packaging [34]. To proceed with this phase, the team needs to read the tool manual because it has not been used before by any team member. Also, the library problem should be resolved as it is not common for undergraduate students in an educational project to have access to industrial libraries.

On the other hand, we can use the OpenCell 15nm library, it was reported to have a better optimization than the 45nm library. The main change in the 15nm library is the use of FinFet transistors but it has less number of standard cells. The 15nm has a smaller number of cells so when comparing between the two libraries using the common cells we get a better optimization of the speed, area and power consumption.

More development could be done by using this processor inside the higher project called Convolutional Neural Network for LIDAR Data processing. Its main function would be making

the preprocessing steps on the LIDAR Data before going into the Neural Network. After that the Neural Network will make all the related processes of detecting and selecting objects.

7 - Conclusion:

In conclusion, we could notice that the topographical flow has the best timing with the shortest clock period of 1.8 ns. The flat flow has better timing than the hierarchical flow as expected since the flat flow performs ungrouping of the hierarchies to allow for further optimizations. Regarding the area, the hierarchical flow has the smallest area as expected. Both the flat flow and the topographical flat flow have nearly the same area. When it comes to power, the Hierarchical flow has the lowest power consumption. That is expected since its clock period is the largest. To make a reasonable comparison in the results, we cannot compare the periods or the power only. The power delay product should be considered. It was noticed that the power delay product of the flat flow is the smallest. The detailed results are emphasized in the below table.

Point of Comparison	Flat	Hierarchical	Topographical (flat)
Area (square um)	134,753	128,950	136,938
Timing (min period)	2.3 ns	3.3 ns	1.8 ns
Power	1.8 mW	1.3108 mW	2.42 mW
Power Delay Product	4.14 pJ	4.33 pJ	4.356 pJ

8- Acknowledgements

We would like to extend special thanks to Dr. Mostafa Elshafie for allowing us to use his workstation for running the codes and the communication lab room for meetings. We would also like to thank Dr. Hassan Mostafa and Dr. Amr Helmy for their invaluable support and supervision. Finally, we are truly grateful for Engineer Hoda's guidance and support throughout the project.

4- References

[1] Davide Rossi, Antonio Pullini, Igor Loi, Michael Gautschi, Frank K. Gurkaynak, Andrea Bartolini, Philippe Flatresse, Luca Benini, "A 60 GOPS/W, -1.8 V to 0.9 V body bias ULP cluster in 28 nm UTBB FD-SOI technology", *Journal of Solid-State Electronics, Volume 117, March 2016, Pages 170-184*, DOI**:** 10.1016/j.sse.2015.11.015

[2] Davide Rossi, Antonio Pullini, Igor Loi, Michael Gautschi, Frank K. Gurkaynak, Adam Teman, Jeremy Constantin, Andreas Burg, Ivan Miro-Panades, Edith Beigne, Fabien Clermidy, Philippe Flatresse, Luca Benini, "Energy-Efficient Near-Threshold Parallel Computing: The PULPv2 Cluster", *IEEE Micro, vol. 37, no. 5, pp. 20-31*, DOI**:** 10.1109/MM.2017.3711645

[3] Davide Rossi, Antonio Pullini, Christoph Mueller, Igor Loi, Francesco Conti, Andreas Burg, Luca Benini, Philippe Flatresse, "A Self-Aware Architecture for PVT Compensation and Power Nap in Near Threshold Processors", *IEEE Design & Test, vol. 34, no. 6, pp. 46-53, Dec. 2017*, DOI**:** 10.1109/MDAT.2017.2750907

[4]Michael Gautschi, Michael Schaffner, Frank K. Gurkaynak, Luca Benini, "An Extended Shared Logarithmic Unit for Nonlinear Function Kernel Acceleration in a 65-nm CMOS Multicore Cluster", *IEEE Journal of Solid-State Circuits, Volume: 52, Issue: 1, Jan. 2017, pp 98-112*, DOI**:** 10.1109/JSSC.2016.2626272

[5]Michael Gautschi, Michael Schaffner, Frank K. Gurkaynak, Luca Benini, "A 65nm CMOS 6.4-to-29.2pJ/FLOP@0.8V shared logarithmic floating point unit for acceleration of nonlinear function kernels in a tightly coupled processor cluster", *2016 IEEE International Solid-State Circuits Conference (ISSCC), pp 82 - 83*, DOI**:** 10.1109/ISSCC.2016.7417917

[6]Vincent Camus, Jeremy Schlachter, Christian Enz, Frank K. Gurkaynak, Michael Gautschi, "Approximate 32-bit floating-point unit design with 53% power-area product reduction", *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference, Lausanne, 2016, pp. 465-468.*, DOI**:** 10.1109/ESSCIRC.2016.7598342

[7]Antonio Pullini, Francesco Conti, Davide Rossi, Igor Loi, Michael Gautschi, Luca Benini, "A heterogeneous multi-core system-on-chip for energy efficient brain inspired vision", *IEEE Transactions on Circuits and Systems II: Express Briefs (Volume: PP, Issue: 99), 2017*, DOI**:** 10.1109/TCSII.2017.2652982

[8]Philipp Schoenle, Giovanni Rovere, Florian Glaser, Jonathan Boesser, Noe Brun, Xu Han, Thomas Burger, Schekeb Fateh, Qing Wang, Luca Benini, Qiuting Huang, "A multi-sensor and parallel processing SoC for wearable and implantable telemetry systems", *Proceedings of the 43rd IEEE European Solid State Circuits Conference, (ESSCIRC-2017),*, DOI**:** 10.1109/ESSCIRC.2017.8094564

[9]Philipp Schoenle, Florian Glaser, Thomas Burger, Giovanni Rovere, Luca Benini, Qiuting Huang, "A Multi-Sensor and Parallel Processing SoC for Miniaturized Medical Instrumentation", *IEEE Journal of Solid-State Circuits PP issue:99, pp 1-12*, DOI**:** 10.1109/JSSC.2018.2815653

[10]Florian Zaruba, "Updates on PULPino", 5th RISC-V Workshop Proceedings, Google Quad Campus, Mountain View, California, November 29-30 2016

[11] Antonio Pullini, Davide Rossi, Igor Loi, Alfio Di Mauro, Luca Benini, "Mr.Wolf: A 1 GFLOP/s Energy-Proportional Parallel Ultra Low Power SoC for IoT Edge Processing", *In Proc. European Solid State Circuits Conference (ESSCIRC) 2018, 3-6 Sep 2018, Dresden*, DOI**:** 10.1109/ESSCIRC.2018.8494247

[12]Alfio Di Mauro, Francesco Conti, Davide Schiavone, Davide Rossi, Luca Benini, "Hyperdrive: A Systolically Scalable Binary-Weight CNN Inference Engine for mW IoT End-Nodes Always-On 674uW @4GOP/s Error Resilient Binary Neural Networks With Aggressive SRAM Voltage Scaling on a 22-nm IoT End-Node", *In IEEE Transactions on Circuits and Systems I: Regular Papers, Volume 67, Issue 11, November 2020*, DOI**:** 10.1109/TCSI.2020.3012576

[13]Florian Zaruba, Fabian Schuiki, Stefan Mach, Luca Benini, "The Floating Point Trinity: A Multi-modal Approach to Extreme Energy-Efficiency and Performance", *In Proc. 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Genoa, Italy, 2019*, DOI**:** 10.1109/ICECS46596.2019.8964820

[14]Florian Zaruba, Fabian Schuiki, Luca Benini, "Manticore: A 4096-core RISC-V Chiplet Architecture for Ultra-efficient Floating-point Computing", *Presented at Hot Chips: A Symposium on High Performance Chips (HC32) 2020, arXiv: 2008.06502*

[15]Davide Schiavone, Davide Rossi, Alfio Di Mauro, Frank Gurkaynak, Timothy Saxe, Mao Wang, Ket Chong Yap, Luca Benini, "Arnold: an eFPGA-Augmented RISC-V SoC for Flexible and Low-Power IoT End-Nodes", *arXiv: 2006.14256*

[16] P. Chiu, C. Celio, K. Asanovic, D. Patterson and B. Nikolic, "An Out-of-Order RISC-V Processor with Resilient Low-Voltage Operation in 28NM CMOS", *2018 IEEE Symposium on VLSI Circuits*, 2018. Available: 10.1109/vlsic.2018.8502320 [Accessed 15 April 2021].

[17]Y. Lee et al., "A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators", *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC)*, 2014. Available: 10.1109/esscirc.2014.6942056 [Accessed 15 April 2021].

[18]"EDACafe: ASICs ... the Book", EDACafe, 2021. [Online]. Available: https://www10.edacafe.com/book/ASIC/CH01/CH01.5.php. [Accessed: 11- Jul- 2021].

[19]https://github.com/vincent-camus/carry-cut-backadder/tree/master/lib/NangateOpenCellLibrary_PDKv1_3_v2010_12

[20] "Digital Standard Cell Library SAED_EDK90_CORE DATABOOK."

[21] Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, Greg Yeric, ASAP7: A 7-nm finFET predictive process design kit, Microelectronics Journal, Volume 53,2016,Pages 105-115,ISSN 0026-2692,https://doi.org/10.1016/j.mejo.2016.04.006.(https://www.sciencedirect.com/science/article/pii/S002626921630026X)

[22]"Design Compiler User Guide", Synopsis.

[23]"IC Compiler Design Planning User Guide", Synopsis

[24]"Clock Tree routing Algorithms", VLSI- Physical Design For Freshers, 2021. [Online]. Available: https://www.physicaldesign4u.com/2020/03/clock-tree-routing-algorithms.html. [Accessed: 11- Jul- 2021].

[25]"Optimization of Crosstalk Delta Delay on Clock Nets", Design And Reuse, 2021. [Online]. Available: https://www.design-reuse.com/articles/48514/optimization-of-crosstalk-delta-delay-on-clock-nets.html. [Accessed: 11- Jul- 2021].

[26]"Introduction — CORE-V CV32E40P User Manual documentation", Cv32e40p.readthedocs.io, 2021. [Online]. Available: https://cv32e40p.readthedocs.io/en/latest/intro/. [Accessed: 11- Jul- 2021].

[27]"Synopsys Academic Program Frequently Asked Questions", Synopsys.com, 2021. [Online]. Available: https://www.synopsys.com/company/legal/software-integrity/academic-faqs.html#:~:text=by%2Dcase%20basis.-,How%20Much%20Does%20it%20Cost%20to%20Participate%20in%20the%20Program,%24240%20administrative%20fee%20per%20year. [Accessed: 11- Jul- 2021].

[28]"How much is a single license of mentor graphics, synopsys, or cadence electronic design automation software? - Quora", Quora.com, 2021. [Online]. Available: https://www.quora.com/How-much-is-a-single-license-of-mentor-graphics-synopsys-or-cadence-electronic-design-automation-software%20. [Accessed: 11- Jul- 2021].

[29]"RISC-V Global Forum: Initiate. Innovate. Impact. - RISC-V International", RISC-V International, 2021. [Online]. Available: https://riscv.org/blog/2020/09/risc-v-global-forum-initiate-innovate-impact/. [Accessed: 11- Jul- 2021].

[30]"Strengthening the Internet of Things with a Secure RISC-V IoT Stack - RISC-V International", RISC-V International, 2021. [Online]. Available: https://riscv.org/blog/2020/06/strengthening-the-internet-of-things-with-a-secure-risc-v-iot-stack/. [Accessed: 11- Jul- 2021].

[31]Players.brightcove.net, 2021. [Online]. Available: https://players.brightcove.net/5748441669001/rka4xWwYG_default/index.html?videoId=6186389121001. [Accessed: 11- Jul- 2021].

[32]"What is Electromigration? – Complete Overview | Synopsys", Synopsys.com, 2021. [Online]. Available: https://www.synopsys.com/glossary/what-is-electromigration.html. [Accessed: 11- Jul- 2021].

[33]"Device Aging Analysis using Synopsys Custom Design Platform", Synopsys.com, 2021. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/resources/videos/device-aging-analysis-video-wp.html. [Accessed: 11- Jul- 2021].

[34]"Synopsys and TSMC Accelerate 2.5D/3DIC Designs with Chip-on-Wafer-on-Substrate and Integrated Fan-Out Certified Design Flows", News.synopsys.com, 2021. [Online]. Available: https://news.synopsys.com/2020-08-25-Synopsys-and-TSMC-Accelerate-2-5D-3DIC-Designs-with-Chip-on-Wafer-on-Substrate-and-Integrated-Fan-Out-Certified-Design-Flows. [Accessed: 11- Jul- 2021].

Appendix

1. *Topographical flow Codes*

First Pass Synthesis

```
set hdlin_sverilog_std 2009
set design cv32e40p_core
set_app_var search_path
"/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLDM \
/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Back_End/virtuoso/NangateO
penCellLibrary"
set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"
set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"
sh rm -rf work
sh mkdir -p work
define_design_lib work -path ./work
analyze -library work -format sverilog ../rtl/cv32e40p_core.sv
elaborate cv32e40p_core -lib work
current_design
check_design
source ./cons/cv32e40p_core.sdc
link
compile_ultra -timing_high_effort_script \
        -gate_clock \
            -retime
compile_ultra -incremental
report_area > ./report/synth_area.rpt
report_power > ./report/synth_power.rpt
report_cell > ./report/synth_cells.rpt
report_qor  > ./report/synth_qor.rpt
report_resources > ./report/synth_resources.rpt
report_timing -max_paths 10 > ./report/synth_timing.rpt
write_sdc  output/cv32e40p_core.sdc
define_name_rules  no_case -case_insensitive
change_names -rule no_case -hierarchy
change_names -rule verilog -hierarchy
set verilogout_no_tri       true
set verilogout_equation  false
write -hierarchy -format verilog -output output/cv32e40p_core.v
write -f ddc -hierarchy -output output/cv32e40p_core.ddc
exit
```

Floorplanning & Power Grid

```
set hdlin_sverilog_std 2009
set design cv32e40p_core

sh rm -rf $design

set sc_dir "/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12"
```

```
set_app_var search_path
"/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLDM \
                        /home/nano/Documents/risc/rtl"

set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"
set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"


create_mw_lib   ./${design} \
        -technology $sc_dir/tech/techfile/milkyway/FreePDK45_10m.tf \
                -mw_reference_library $sc_dir/lib/Back_End/mdb \
                -hier_separator {/} \
                -bus_naming_style {[%d]} \
                -open

set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"
set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"
set tech2itf "$sc_dir/tech/rcxt/FreePDK45_10m.map"

set_tlu_plus_files -max_tluplus $tlupmax \
          -min_tluplus $tlupmin \
                  -tech2itf_map $tech2itf


####################################################
import_designs  ../syn/output/${design}.v \
        -format verilog \
                -top ${design} \
                -cel ${design}
source  ../syn/cons/cv32e40p_core.sdc
set_propagated_clock [get_clocks clk_i]
save_mw_cel -as ${design}_1_imported


#############################################
########### 2. Floorplan ####################
#############################################


create_floorplan -control_type aspect_ratio \
         -core_aspect_ratio 2 \
         -core_utilization .25 \
         -start_first_row -flip_first_row \
         -left_io2core 12.4 -bottom_io2core 12.4 -right_io2core 12.4 -top_io2core 12.4

report_ignored_layers
remove_ignored_layers -all
set_ignored_layers -max_routing_layer metal6
save_mw_cel -as ${design}_2_fp

#################################################
########### 3. POWER NETWORK ####################
#################################################
```

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

set_fp_rail_constraints -set_ring -nets  {VDD VSS}              \
                  -horizontal_ring_layer { metal7 metal9 } \
                  -vertical_ring_layer { metal8 metal10 }  \
                  -ring_spacing 0.8                      \
                  -ring_width 5                        \
                  -ring_offset 0.8                      \
                  -extend_strap core_ring
set_fp_rail_constraints -add_layer  -layer metal10 -direction vertical   -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum
set_fp_rail_constraints -add_layer  -layer metal9  -direction horizontal -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum
set_fp_rail_constraints -add_layer  -layer metal8  -direction vertical   -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum
set_fp_rail_constraints -add_layer  -layer metal7  -direction horizontal -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum
set_fp_rail_constraints -add_layer  -layer metal6  -direction vertical   -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum

set_fp_rail_constraints -set_global

## Creating virtual PG pads
############################

create_fp_virtual_pad -net VDD -point {17.3190 510.5095}
create_fp_virtual_pad -net VDD -point {52.3200 510.5095}
create_fp_virtual_pad -net VDD -point {87.4110 510.5095}
create_fp_virtual_pad -net VDD -point {122.4120 510.5095}
create_fp_virtual_pad -net VDD -point {157.4125 510.4185}
create_fp_virtual_pad -net VDD -point {192.5950 510.4185}
create_fp_virtual_pad -net VDD -point {227.5955 510.5095}
create_fp_virtual_pad -net VDD -point {262.7775 510.5095}
create_fp_virtual_pad -net VDD -point {268.3995 492.2835}
create_fp_virtual_pad -net VDD -point {268.2180 480.5865}
create_fp_virtual_pad -net VDD -point {268.2180 445.8575}
create_fp_virtual_pad -net VDD -point {268.2180 411.4010}
create_fp_virtual_pad -net VDD -point {268.2180 376.4000}
create_fp_virtual_pad -net VDD -point {268.2180 342.0340}
create_fp_virtual_pad -net VDD -point {268.2180 307.3055}
create_fp_virtual_pad -net VDD -point {268.3090 272.3955}
create_fp_virtual_pad -net VDD -point {268.2180 237.4855}
create_fp_virtual_pad -net VDD -point {268.2180 203.1195}
create_fp_virtual_pad -net VDD -point {268.2180 168.3000}
create_fp_virtual_pad -net VDD -point {268.2180 133.6620}
create_fp_virtual_pad -net VDD -point {268.2180 98.8425}
create_fp_virtual_pad -net VDD -point {268.2180 64.2950}
create_fp_virtual_pad -net VDD -point {268.2180 29.6570}
create_fp_virtual_pad -net VDD -point {262.5055 -0.0845}
```

```
create_fp_virtual_pad -net VDD -point {227.5955 -0.1750}
create_fp_virtual_pad -net VDD -point {192.5040 -0.0845}
create_fp_virtual_pad -net VDD -point {157.4125 -0.0845}
create_fp_virtual_pad -net VDD -point {122.3215 0.0060}
create_fp_virtual_pad -net VDD -point {87.2300 -0.0845}
create_fp_virtual_pad -net VDD -point {52.2290 -0.0845}
create_fp_virtual_pad -net VDD -point {17.2285 -0.0845}
create_fp_virtual_pad -net VDD -point {0.0000 18.2320}
create_fp_virtual_pad -net VDD -point {0.0000 52.6885}
create_fp_virtual_pad -net VDD -point {0.0000 87.5080}
create_fp_virtual_pad -net VDD -point {0.0000 122.1460}
create_fp_virtual_pad -net VDD -point {-0.0905 156.4215}
create_fp_virtual_pad -net VDD -point {0.0000 191.6035}
create_fp_virtual_pad -net VDD -point {0.0000 226.3320}
create_fp_virtual_pad -net VDD -point {0.0000 260.9700}
create_fp_virtual_pad -net VDD -point {0.0000 295.6990}
create_fp_virtual_pad -net VDD -point {0.0000 330.5185}
create_fp_virtual_pad -net VDD -point {0.0000 364.9750}
create_fp_virtual_pad -net VDD -point {0.0000 399.7945}
create_fp_virtual_pad -net VDD -point {0.0000 434.7045}
create_fp_virtual_pad -net VDD -point {0.0000 469.1615}
create_fp_virtual_pad -net VDD -point {0.0000 503.7995}
create_fp_virtual_pad -net VSS -point {34.8195 510.4185}
create_fp_virtual_pad -net VSS -point {69.8200 510.6905}
create_fp_virtual_pad -net VSS -point {104.8210 510.5095}
create_fp_virtual_pad -net VSS -point {140.0030 510.6000}
create_fp_virtual_pad -net VSS -point {175.0040 510.6000}
create_fp_virtual_pad -net VSS -point {210.0045 510.5095}
create_fp_virtual_pad -net VSS -point {245.0055 510.4185}
create_fp_virtual_pad -net VSS -point {268.2180 498.1775}
create_fp_virtual_pad -net VSS -point {268.2180 463.3580}
create_fp_virtual_pad -net VSS -point {268.2180 428.7200}
create_fp_virtual_pad -net VSS -point {268.2180 393.9915}
create_fp_virtual_pad -net VSS -point {268.2180 359.2625}
create_fp_virtual_pad -net VSS -point {268.2180 324.5340}
create_fp_virtual_pad -net VSS -point {268.2180 289.8955}
create_fp_virtual_pad -net VSS -point {268.2180 255.0765}
create_fp_virtual_pad -net VSS -point {268.3090 220.3475}
create_fp_virtual_pad -net VSS -point {268.3090 185.8000}
create_fp_virtual_pad -net VSS -point {268.3090 150.9810}
create_fp_virtual_pad -net VSS -point {268.2180 116.3430}
create_fp_virtual_pad -net VSS -point {268.2180 81.6140}
create_fp_virtual_pad -net VSS -point {268.2180 46.8855}
create_fp_virtual_pad -net VSS -point {268.2180 12.1565}
create_fp_virtual_pad -net VSS -point {245.0960 -0.0845}
create_fp_virtual_pad -net VSS -point {209.9140 -0.1750}
create_fp_virtual_pad -net VSS -point {174.9130 -0.0845}
create_fp_virtual_pad -net VSS -point {140.0030 -0.0845}
create_fp_virtual_pad -net VSS -point {104.8210 -0.0845}
create_fp_virtual_pad -net VSS -point {69.9110 -0.0845}
create_fp_virtual_pad -net VSS -point {34.6380 -0.0845}
create_fp_virtual_pad -net VSS -point {0.0000 -0.0845}
create_fp_virtual_pad -net VSS -point {0.0000 35.1880}
```

```
create_fp_virtual_pad -net VSS -point {0.0000 70.1890}
create_fp_virtual_pad -net VSS -point {0.0000 104.6455}
create_fp_virtual_pad -net VSS -point {0.0000 139.4650}
create_fp_virtual_pad -net VSS -point {0.0000 174.1940}
create_fp_virtual_pad -net VSS -point {0.0000 208.7410}
create_fp_virtual_pad -net VSS -point {0.0905 243.3790}
create_fp_virtual_pad -net VSS -point {0.0905 278.3800}
create_fp_virtual_pad -net VSS -point {0.0000 312.9275}
create_fp_virtual_pad -net VSS -point {0.0000 347.5655}
create_fp_virtual_pad -net VSS -point {0.0000 382.4755}
create_fp_virtual_pad -net VSS -point {0.0000 417.0230}
create_fp_virtual_pad -net VSS -point {0.0000 451.8425}
create_fp_virtual_pad -net VSS -point {0.0905 486.5710}
# you can create them with gui. Preroute > Create Virtual Power Pad
##########################################################################
##########################################################################
############
create_fp_virtual_pad -net VDD -point {23.2985 518.8505}
create_fp_virtual_pad -net VDD -point {70.8200 518.9530}
create_fp_virtual_pad -net VDD -point {118.3410 518.8505}
create_fp_virtual_pad -net VDD -point {165.7600 518.9530}
create_fp_virtual_pad -net VDD -point {213.3835 518.9530}
create_fp_virtual_pad -net VDD -point {260.9050 518.9530}
create_fp_virtual_pad -net VDD -point {272.6050 482.7220}
create_fp_virtual_pad -net VDD -point {272.6050 435.6115}
create_fp_virtual_pad -net VDD -point {272.6050 388.6035}
create_fp_virtual_pad -net VDD -point {272.6050 341.8010}
create_fp_virtual_pad -net VDD -point {272.6050 294.5875}
create_fp_virtual_pad -net VDD -point {272.6050 247.2720}
create_fp_virtual_pad -net VDD -point {272.5025 200.3665}
create_fp_virtual_pad -net VDD -point {272.5105 153.5530}
create_fp_virtual_pad -net VDD -point {272.5105 106.5435}
create_fp_virtual_pad -net VDD -point {272.5105 59.1235}
create_fp_virtual_pad -net VDD -point {272.6135 12.2170}
create_fp_virtual_pad -net VDD -point {237.0990 -0.0985}
create_fp_virtual_pad -net VDD -point {189.6795 0.0035}
create_fp_virtual_pad -net VDD -point {142.0540 -0.0985}
create_fp_virtual_pad -net VDD -point {94.7370 -0.2015}
create_fp_virtual_pad -net VDD -point {47.1120 -0.2015}
create_fp_virtual_pad -net VDD -point {0.0000 0.0035}
create_fp_virtual_pad -net VDD -point {0.0000 47.5260}
create_fp_virtual_pad -net VDD -point {0.0000 94.8450}
create_fp_virtual_pad -net VDD -point {0.0000 141.6485}
create_fp_virtual_pad -net VDD -point {0.0000 188.6580}
create_fp_virtual_pad -net VDD -point {0.0000 235.8720}
create_fp_virtual_pad -net VDD -point {0.0000 283.0865}
create_fp_virtual_pad -net VDD -point {0.0000 329.9930}
create_fp_virtual_pad -net VDD -point {-0.1025 376.7970}
create_fp_virtual_pad -net VDD -point {0.0000 423.8060}
create_fp_virtual_pad -net VDD -point {0.0000 471.1230}
create_fp_virtual_pad -net VSS -point {0.0000 494.6275}
create_fp_virtual_pad -net VSS -point {47.0090 518.8505}
create_fp_virtual_pad -net VSS -point {94.6340 518.9530}
```

```
create_fp_virtual_pad -net VSS -point {142.0535 518.8505}
create_fp_virtual_pad -net VSS -point {189.5760 518.8505}
create_fp_virtual_pad -net VSS -point {237.2010 518.8505}
create_fp_virtual_pad -net VSS -point {272.5090 506.2260}
create_fp_virtual_pad -net VSS -point {272.5105 459.1135}
create_fp_virtual_pad -net VSS -point {272.5145 412.0000}
create_fp_virtual_pad -net VSS -point {272.5145 365.1955}
create_fp_virtual_pad -net VSS -point {272.5145 318.1850}
create_fp_virtual_pad -net VSS -point {272.5145 271.1750}
create_fp_virtual_pad -net VSS -point {272.5145 223.9600}
create_fp_virtual_pad -net VSS -point {272.5145 176.8470}
create_fp_virtual_pad -net VSS -point {272.5145 129.9395}
create_fp_virtual_pad -net VSS -point {272.5145 83.1350}
create_fp_virtual_pad -net VSS -point {272.5145 35.8170}
create_fp_virtual_pad -net VSS -point {260.8135 -0.2105}
create_fp_virtual_pad -net VSS -point {213.2900 -0.1075}
create_fp_virtual_pad -net VSS -point {165.8695 -0.0050}
create_fp_virtual_pad -net VSS -point {118.3465 -0.0050}
create_fp_virtual_pad -net VSS -point {70.8230 -0.0050}
create_fp_virtual_pad -net VSS -point {23.3000 -0.1075}
create_fp_virtual_pad -net VSS -point {0.0000 24.0130}
create_fp_virtual_pad -net VSS -point {0.0000 71.2285}
create_fp_virtual_pad -net VSS -point {0.0000 118.1360}
create_fp_virtual_pad -net VSS -point {0.0000 165.1460}
create_fp_virtual_pad -net VSS -point {-0.1025 212.2585}
create_fp_virtual_pad -net VSS -point {0.0000 259.2685}
create_fp_virtual_pad -net VSS -point {0.0000 306.4840}
create_fp_virtual_pad -net VSS -point {0.0000 353.4940}
create_fp_virtual_pad -net VSS -point {0.0000 400.4015}
create_fp_virtual_pad -net VSS -point {0.0000 447.5140}
##########################################################################
##########################################################################
############
##########################################################################
##########################################################################
############

synthesize_fp_rail  -nets {VDD VSS} -synthesize_power_plan -target_voltage_drop 22 -voltage_supply 1.1
-power_budget 500

# Generate the real power network
commit_fp_rail

# Set maximum metal layer in DRC for power network connections for standard cells
set_preroute_drc_strategy -max_layer metal6

# Connect standard cells power and ground pins to the power and ground rings and straps
preroute_standard_cells -fill_empty_rows -remove_floating_pieces

# Analyze IR-drop; Modify power network constraints and re-synthesize, as needed.
analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1

add_tap_cell_array -master   TAP \
```

```
            -distance 30  \
            -pattern  stagger_every_other_row

save_mw_cel -as ${design}_3_power
############################################################################
######################OUTPUT TO DC TOPO#####################################
############################################################################
write_floorplan -all /home/nano/Documents/risc/pnr/output/cv32e40p_core.fp
write_def -output /home/nano/Documents/risc/pnr/output/cv32e40p_core.def
############################################################################
```

Second Pass Synthesis

```
set hdlin_sverilog_std 2009
set design cv32e40p_core
set_app_var search_path
"/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLDM \
/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Back_End/virtuoso/NangateO
penCellLibrary"

set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"
set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"

sh rm -rf work
set sc_dir "/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12"
create_mw_lib   ./${design} \
          -technology $sc_dir/tech/techfile/milkyway/FreePDK45_10m.tf \
                    -mw_reference_library $sc_dir/lib/Back_End/mdb \
                    -hier_separator {/} \
                    -bus_naming_style {[%d]} \
                    -open
set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"
set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"
set tech2itf "$sc_dir/tech/rcxt/FreePDK45_10m.map"
set_tlu_plus_files -max_tluplus $tlupmax \
              -min_tluplus $tlupmin \
                        -tech2itf_map $tech2itf
sh mkdir -p work
define_design_lib work -path ./work
analyze -library work -format sverilog ../rtl/cv32e40p_core.sv
elaborate cv32e40p_core -lib work
current_design
extract_physical_constraints /home/nano/Documents/risc/pnr/output/cv32e40p_core.def -verbose
-exact -allow_physical_cells
read_floorplan /home/nano/Documents/risc/pnr/output/cv32e40p_core.fp -echo >
./report/read_floorplan_report.rpt

check_design
source ./cons/cv32e40p_core.sdc
link
compile_ultra -spg \
               -timing_high_effort_script \
```

```
        -gate_clock \
            -retime


report_area > ./report/synth_area.rpt
report_power > ./report/synth_power.rpt
report_cell > ./report/synth_cells.rpt
report_qor  > ./report/synth_qor.rpt
report_resources > ./report/synth_resources.rpt
report_timing -max_paths 10 > ./report/synth_timing.rpt

write_sdc  output/cv32e40p_core.sdc
define_name_rules  no_case -case_insensitive
change_names -rule no_case -hierarchy
change_names -rule verilog -hierarchy
set verilogout_no_tri       true
set verilogout_equation  false

write -hierarchy -format verilog -output output/cv32e40p_core.v
write -f ddc -hierarchy -output output/cv32e40p_core.ddc
write_floorplan -all /home/nano/Documents/risc/syn_topo/output/cv32e40p_core.fp
write_def -output /home/nano/Documents/risc/syn_topo/output/cv32e40p_core.def

exit
```

Routing

```
set hdlin_sverilog_std 2009
set design cv32e40p_core


set sc_dir "/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12"

set_app_var search_path
"/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLDM \
                        /home/nano/Documents/risc/rtl"

set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"
set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"



set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"
set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"
set tech2itf "$sc_dir/tech/rcxt/FreePDK45_10m.map"

set_tlu_plus_files -max_tluplus $tlupmax \
            -min_tluplus $tlupmin \
                    -tech2itf_map $tech2itf

################second time icc#####################
open_mw_lib cv32e40p_core
```

```
read_ddc /home/nano/Documents/risc/syn_topo/output/cv32e40p_core.ddc
read_def /home/nano/Documents/risc/syn_topo/output/cv32e40p_core.def
#####################################################
source  ../syn/cons/cv32e40p_core.sdc
set_propagated_clock [get_clocks clk_i]


########################################
################PLACEMENT################
########################################
place_opt -spg -effort high -congestion #max_cell_density 40%

set tie_pins [get_pins -all -filter "constant_value == 0 || constant_value == 0 && name !~ V* &&
is_hierarchical == false "]
derive_pg_connection    -power_net VDD                  \
                        -ground_net VSS         \
                        -power_pin VDD                  \
                        -ground_pin VSS



if {[sizeof_collection $tie_pins] > 0 } {
        connect_tie_cells -objects $tie_pins \
          -obj_type port_inst \
                -tie_low_lib_cell  LOGIC0_X1 \
                -tie_high_lib_cell LOGIC1_X1
}
save_mw_cel -as ${design}_4_placed



##############################################
########## 5. CTS      ####################
##############################################

check_physical_design -stage pre_clock_opt

check_clock_tree
report_clock_tree

report_qor
report_timing

set_driving_cell -lib_cell BUF_X16 -pin Z [get_ports clk_i]

set_clock_tree_options \
         -clock_trees clk_i \
                -target_early_delay 0.1 \
                -target_skew 0.5 \
                -max_capacitance 300 \
                -max_fanout 10 \
                -max_transition 0.3
```

```
set_clock_tree_options -clock_trees clk_i \
                       -buffer_relocation true \
                       -buffer_sizing true \
                       -gate_relocation true \
                       -gate_sizing true

set_clock_tree_references -references [get_lib_cells */CLKBUF*]

compile_clock_tree

clock_opt -fix_hold_all_clocks -congestion

report_congestion
report_qor

derive_pg_connection -power_net VDD  \
                     -ground_net VSS \
                     -power_pin VDD  \
                     -ground_pin VSS

save_mw_cel -as ${design}_5_cts

##report_timing to see network delay of launch from clk port to reg and capture from reg to clk port and
check if balanced
##create_balanced_group if not
##local skew and global skew
################################################
########### 6. Routing   ######################
################################################

insert_spare_cells -lib_cell {NOR2_X4 NAND2_X4} \
                   -num_instances 20 \
                   -cell_name SPARE_PREFIX_NAME \
                   -tie

set_dont_touch  [all_spare_cells] true
set_attribute [all_spare_cells]  is_soft_fixed true

check_physical_design -stage pre_route_opt

all_ideal_nets
all_high_fanout -nets -threshold 100
check_routeability

set_delay_calculation_options -arnoldi_effort high




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

```
set_si_options -route_xtalk_prevention true \
        -delta_delay true \
        -min_delta_delay true \
        -static_noise true\
        -timing_window true

route_auto -effort high


route_opt -incremental    -effort high  -stage track



derive_pg_connection -power_net VDD  \
            -ground_net VSS \
            -power_pin VDD  \
            -ground_pin VSS


save_mw_cel -as ${design}_6_routed
##############################################
########### 7. Finishing ####################
##############################################
#
#derive_pg_connection -power_net VDD -ground_net VSS -power_pin VDD -ground_pin VSS -create_ports
top -reconnect -preserve_physical_only_pg



insert_stdcell_filler -cell_without_metal {FILLCELL_X32 FILLCELL_X16 FILLCELL_X8 FILLCELL_X4
FILLCELL_X2 FILLCELL_X1} \
        -connect_to_power VDD -connect_to_ground VSS

route_eco -search_repair_loop 20 -utilize_dangling_wires -reroute modified_nets_first_then_others
route_zrt_eco -open_net_driven true -reroute modified_nets_first_then_others
-reuse_existing_global_route true -utilize_dangling_wires true


focal_opt     -drc_nets all -effort high
focal_opt     -drc_pins all -effort high

derive_pg_connection    -power_net VDD                  \
                        -ground_net VSS         \
                        -power_pin VDD                  \
                        -ground_pin VSS
remove_floating_pg -nets VDD -pad_lib_cells {*}
remove_floating_pg -nets VSS -pad_lib_cells {*}

insert_metal_filler -timing_driven -from_metal 1 -to_metal 6

save_mw_cel -as ${design}_7_finished
```

```
save_mw_cel -as ${design}


#via insertion
preroute_standard_cells -fill_empty_rows #-remove_floating_pieces
create_preroute_vias -nets {VDD} \
        -from_layer metal1 -to_layer metal6 \
        -to_object_strap -from_object_std_pin_connection\
        -within {{126.5  68.2} {129.1 68.5}} \
        -mark_as user_defined

create_preroute_vias -nets {VDD} \
        -from_layer metal1 -to_layer metal6 \
        -to_object_strap -from_object_std_pin_connection\
        -within {{80.5  68.2} {83.5 68.5}} \
        -mark_as user_defined

create_preroute_vias -nets {VDD} \
        -from_layer metal1 -to_layer metal6 \
        -to_object_strap -from_object_std_pin_connection\
        -within {{23.1  68.2} {25.8 68.5}} \
        -mark_as user_defined

create_preroute_vias -nets {VDD} \
        -from_layer metal1 -to_layer metal6 \
        -to_object_strap -from_object_std_pin_connection\
        -within {{217.7  68.2} {220.3 68.5}} \
        -mark_as user_defined

create_preroute_vias -nets {VDD} \
        -from_layer metal1 -to_layer metal6 \
        -to_object_strap -from_object_std_pin_connection\
        -within {{172.1  68.2} {174.7 68.5}} \
        -mark_as user_defined

update_clock_latency

report_clock -skew

################################################
########### 8. Checks and Outputs ############
################################################
#PG
verify_pg_nets -pad_pin_connection all
#routing
verify_zrt_route
#LVS
verify_lvs -ignore_floating_port -ignore_floating_net \
      -check_open_locator -check_short_locator
#DRC
report_design -physical
#congestion
report_congestion
```

```
#timing
report_timing



#min and max routing
layers#####################################################################

set_write_stream_options -map_layer $sc_dir/tech/strmout/FreePDK45_10m_gdsout.map \
                -output_filling fill \
                                -child_depth 20 \
                                -output_outdated_fill  \
                                -output_pin  {text geometry}
write_stream -lib $design \
            -format gds\
                        -cells $design\
                        ./output/${design}.gds
define_name_rules  no_case -case_insensitive
change_names -rule no_case -hierarchy
change_names -rule verilog -hierarchy
set verilogout_no_tri       true
set verilogout_equation  false

write_verilog -pg -no_physical_only_cells ./output/${design}_icc.v
write_verilog -no_physical_only_cells ./output/${design}_icc_nopg.v
##write new sdc to be used in pt
##ctrl+shift+N on path in gui to search for clock buffers
extract_rc
write_parasitics -output {./output/${design}.spef}

write_sdc ./output/${design}.sdc
close_mw_cel
close_mw_lib
exit
```

Primetime Hold

```
# Set your top module name as design
set design cv32e40p_core

set                                     link_path                                                                       "*
/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLDM/Nan
gateOpenCellLibrary_ff1p25v0c.db"

read_verilog "../../pnr/output/${design}_icc.v"
#read_verilog "./${design}_icc.v"

current_design $design
link

source ../../pnr/output/${design}.sdc
```

```
read_parasitics ../../pnr/output/${design}.spef.min

update_timing
report_timing -delay_type min
fix_eco_timing -type hold -methods insert_buffer -buffer_list {BUF_X4}
write_changes -format icctcl -output ./eco_hold.tcl
save_session ${design}_min.session

report_constraint -all_violators -significant_digits 4 > ./${design}.min_constr.rpt
report_timing -delay_type min -nworst 40 -significant_digits 4 > ./${design}.min_timing.rpt

write_sdf ./${design}.min.sdf
```

Primetime Setup

```
# Set your top module name as design
set design cv32e40p_core

set link_path  "*
/home/nano/Documents/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLDM/Nan
gateOpenCellLibrary_ss0p95vn40c.db"

read_verilog "../../pnr/output/${design}_icc.v"

current_design $design
link

source ../../pnr/output/${design}.sdc
read_parasitics ../../pnr/output/${design}.spef.max

update_timing
report_timing -delay_type max
fix_eco_timing -type setup -methods size_cell -buffer_list {BUFX4}
write_changes -format icctcl -output ./eco2.tcl
save_session ${design}_max.session

report_constraint -all_violators -significant_digits 4 > ./${design}.max_constr.rpt
report_timing -delay_type max -nworst 40 -significant_digits 4 > ./${design}.max_timing.rpt

write_sdf ./${design}.max.sdf
```

2. Hierarchical flow Codes

2.1 Synthesis

```
# Set your top module name as design

set hdlin_sverilog_std 2009

set design cv32e40p_core



# Set the search_path, link_library, and target_library Synopsys application variables

set_app_var search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLD
M \
                        /home/ahesham/Desktop/RISK_GP/rtl"


set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"
set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"

# Remove any residing work directory from previous runs
sh rm -rf work

# Make the directory work
sh mkdir -p work

# Set the directory work to be the work library for this synthesis run
define_design_lib work -path ./work

# Analyze and elaborate your top module
analyze -library work -format sverilog ../rtl/cv32e40p_pkg.sv
analyze -library work -format sverilog ../rtl/cv32e40p_apu_core_pkg.sv
analyze -library work -format sverilog ../rtl/cv32e40p_fpu_pkg.sv
analyze -library work -format sverilog ../rtl/cv32e40p_tracer_pkg.sv
analyze -library work -format sverilog ../rtl/cv32e40p_sim_clock_gate.sv
analyze -library work -format sverilog ../rtl/cv32e40p_register_file_ff.sv
analyze -library work -format sverilog ../rtl/cv32e40p_decoder.sv
analyze -library work -format sverilog ../rtl/cv32e40p_controller.sv
analyze -library work -format sverilog ../rtl/cv32e40p_int_controller.sv
analyze -library work -format sverilog ../rtl/cv32e40p_alu.sv
analyze -library work -format sverilog ../rtl/cv32e40p_mult.sv
analyze -library work -format sverilog ../rtl/cv32e40p_obi_interface.sv
analyze -library work -format sverilog ../rtl/cv32e40p_prefetch_controller.sv
analyze -library work -format sverilog ../rtl/cv32e40p_fifo.sv
analyze -library work -format sverilog ../rtl/cv32e40p_popcnt.sv
analyze -library work -format sverilog ../rtl/cv32e40p_ff_one.sv
analyze -library work -format sverilog ../rtl/cv32e40p_alu_div.sv
```

```
analyze -library work -format sverilog ../rtl/cv32e40p_register_file_ff.sv
analyze -library work -format sverilog ../rtl/cv32e40p_sim_clock_gate.sv
analyze -library work -format sverilog ../rtl/cv32e40p_hwloop_regs.sv
analyze -library work -format sverilog ../rtl/cv32e40p_apu_disp.sv
analyze -library work -format sverilog ../rtl/${design}.sv

elaborate $design -lib work

# Make sure that the current design is your top module in dc_shell memory
current_design

# Check design for any inconsistencies
check_design

# Read the timing constraints file
source ./cons/cv32e40p_core.sdc

# Resolve references
link

# Synthesize and optimize the gate-level netlist
```

2.2 Place and route

```
set hdlin_sverilog_std 2009
set design cv32e40p_core

sh rm -rf $design
set sc_dir "/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12"


# Set the search_path, link_library, and target_library Synopsys application variables
set_app_var search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLD
M \
                        /home/ahesham/Desktop/RISK_GP(copy)/pnr"



set_app_var link_library "* NangateOpenCellLibrary_ss0p95vn40c.db"
set_app_var target_library "NangateOpenCellLibrary_ss0p95vn40c.db"


create_mw_lib   ./${design} \
        -technology $sc_dir/tech/techfile/milkyway/FreePDK45_10m.tf \
                -mw_reference_library $sc_dir/lib/Back_End/mdb \
                -hier_separator {/} \
                -bus_naming_style {[%d]} \
                -open

set tlupmax "$sc_dir/tech/rcxt/FreePDK45_10m_Cmax.tlup"
set tlupmin "$sc_dir/tech/rcxt/FreePDK45_10m_Cmin.tlup"
```

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

set_tlu_plus_files -max_tluplus $tlupmax \
                -min_tluplus $tlupmin \
                        -tech2itf_map $tech2itf

#import_designs  /home/ahesham/Desktop/risc/syn_topo/output/${design}.v \
#               -format verilog \
#                       -top ${design} \
#                       -cel ${design}

import_designs  /home/ahesham/Desktop/RISK_GP/syn/output/${design}.v \
              -format verilog \
                       -top ${design} \
                       -cel ${design}



source  /home/ahesham/Desktop/RISK_GP/syn/cons/cv32e40p_core.sdc
set_propagated_clock [get_clocks clk_i]


save_mw_cel -as ${design}_1_imported

################################################
########### 2. Floorplan ######################
################################################

## Create Starting Floorplan
############################
create_floorplan -control_type aspect_ratio \
               -core_aspect_ratio 2 \
               -core_utilization .25 \
               -start_first_row -flip_first_row \
               -left_io2core 12.4 -bottom_io2core 12.4 -right_io2core 12.4 -top_io2core 12.4

report_ignored_layers
remove_ignored_layers -all

set_ignored_layers -max_routing_layer metal6
create_fp_placement -effort high -congestion_driven  -timing_driven

#create_fp_placement -effort high -congestion_driven -timing_driven

save_mw_cel -as fop

################################################
########### 3. POWER NETWORK ###################
################################################

derive_pg_connection -power_net VDD  \
                  -ground_net VSS \
                  -power_pin VDD  \
```

```
            -ground_pin VSS


set_fp_rail_constraints -set_ring -nets  {VDD VSS}            \
                -horizontal_ring_layer { metal7 metal9 } \
                -vertical_ring_layer { metal8 metal10 }  \
                -ring_spacing 0.8                   \
                -ring_width 5                      \
                -ring_offset 0.8                    \
                -extend_strap core_ring \



set_fp_rail_constraints -add_layer  -layer metal10 -direction vertical   -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum
set_fp_rail_constraints -add_layer  -layer metal9  -direction horizontal -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum
set_fp_rail_constraints -add_layer  -layer metal8  -direction  vertical  -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum
set_fp_rail_constraints -add_layer  -layer metal7  -direction horizontal -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum
set_fp_rail_constraints -add_layer  -layer metal6  -direction vertical   -max_strap 128 -min_strap 20
-min_width 2.5 -spacing minimum
set_fp_rail_constraints -set_global




## Creating virtual PG pads
############################
# you can create them with gui. Preroute > Create Virtual Power Pad
##########################################################################
##########################################################################
############
          -net VDD -point {49.0610 496.9990}
create_fp_virtual_pad -net VDD -point {119.1480 499.0010}
create_fp_virtual_pad -net VDD -point {187.2325 499.0010}
create_fp_virtual_pad -net VDD -point {0.0000 457.9505}
create_fp_virtual_pad -net VDD -point {261.3245 456.9490}
create_fp_virtual_pad -net VDD -point {0.0000 391.8680}
create_fp_virtual_pad -net VDD -point {261.3245 400.8795}
create_fp_virtual_pad -net VDD -point {262.3255 309.7665}
create_fp_virtual_pad -net VDD -point {-1.0010 295.7490}
create_fp_virtual_pad -net VDD -point {0.0000 210.6435}
create_fp_virtual_pad -net VDD -point {258.3205 213.6470}
create_fp_virtual_pad -net VDD -point {263.3270 119.5300}
create_fp_virtual_pad -net VDD -point {-1.0010 124.5365}
create_fp_virtual_pad -net VDD -point {49.0610 1.3835}
create_fp_virtual_pad -net VDD -point {125.1555 -0.6190}
create_fp_virtual_pad -net VDD -point {182.2260 -1.6200}
create_fp_virtual_pad -net VSS -point {81.0385 498.0000}
create_fp_virtual_pad -net VSS -point {154.0730 498.5000}
```

```
create_fp_virtual_pad -net VSS -point {225.1070 497.5000}
create_fp_virtual_pad -net VSS -point {-0.5000 423.4645}
create_fp_virtual_pad -net VSS -point {261.6240 423.4645}
create_fp_virtual_pad -net VSS -point {-1.5005 345.9280}
create_fp_virtual_pad -net VSS -point {261.6240 348.4290}
create_fp_virtual_pad -net VSS -point {0.4985 252.6375}
create_fp_virtual_pad -net VSS -point {262.8170 260.6170}
create_fp_virtual_pad -net VSS -point {-0.4985 166.8605}
create_fp_virtual_pad -net VSS -point {261.8195 162.8710}
create_fp_virtual_pad -net VSS -point {261.8195 81.0835}
create_fp_virtual_pad -net VSS -point {-0.4985 80.0860}
create_fp_virtual_pad -net VSS -point {91.7990 -0.0540}
create_fp_virtual_pad -net VSS -point {157.2280 -0.0540}
create_fp_virtual_pad -net VSS -point {219.6710 -0.3025}
create_fp_virtual_pad -net VDD -point {262.4610 31.7895}
create_fp_virtual_pad -net VDD -point {-0.7460 35.2725}
create_fp_virtual_pad -net VDD -point {-0.7460 35.2725}
create_fp_virtual_pad -net VDD -point {-0.7460 35.2725}

##########################################################################
##########################################################################
#############

synthesize_fp_rail  -nets {VDD VSS} -synthesize_power_plan -target_voltage_drop 22 -voltage_supply 1.1
-power_budget 500

# Generate the real power network
commit_fp_rail

# Set maximum metal layer in DRC for power network connections for standard cells
set_preroute_drc_strategy -max_layer metal10 -report_fail

# Connect standard cells power and ground pins to the power and ground rings and straps, if there is an
empty rows, add straps. if there are straps not conncted to anything, remove it
preroute_standard_cells -fill_empty_rows -remove_floating_pieces
#-special_via_rule optimize_via_locations

# Analyze IR-drop; Modify power network constraints and re-synthesize, as needed.
analyze_fp_rail  -nets {VDD VSS} -power_budget 500 -voltage_supply 1.1

# Final Floorplan Assessment
# Updates fp placement after PG mesh creation.
create_fp_placement -effort high -incremental all -congestion_driven  -timing_driven

add_tap_cell_array -master   TAP \
                   -distance 30 \
                   -pattern  stagger_every_other_row



verify_pg_nets -pad_pin_connection all
```

```
save_mw_cel -as ${design}_3_power

################################################
########### 4. Placement ######################
################################################
report_ignored_layers
check_physical_design -stage pre_place_opt
check_physical_constraints

#Performs  simultaneous  placement,  routing, and optimization on the design
place_opt -effort high -congestion -cts
check_physical_constraints

psynopt -congestion

check_legality

verify_pg_nets -pad_pin_connection all

# DEFINING POWER/GROUND NETS AND PINS
derive_pg_connection    -power_net VDD                  \
                        -ground_net VSS         \
                        -power_pin VDD                  \
                        -ground_pin VSS

## Tie fixed values
set tie_pins [get_pins -all -filter "constant_value == 0 || constant_value == 0 && name !~ V* &&
is_hierarchical == false "]

derive_pg_connection    -power_net VDD                  \
                        -ground_net VSS         \
                        -tie

#############
if {[sizeof_collection $tie_pins] > 0 } { connect_tie_cells -objects $tie_pins -obj_type port_inst
-tie_low_lib_cell  LOGIC0_X1  -tie_high_lib_cell LOGIC1_X1}

#############
verify_pg_nets -pad_pin_connection all


#remove_floating_pg -nets VSS -pad_lib_cells {*}
#remove_floating_pg -nets VDD -pad_lib_cells {*}


#verify_pg_nets -pad_pin_connection all

save_mw_cel -as ${design}_4_placed


################################################
########### 5. CTS      ######################
################################################
```

```
check_physical_design -stage pre_clock_opt
check_clock_tree
report_clock_tree

set_driving_cell -lib_cell BUF_X16 -pin Z [get_ports clk_i]

define_routing_rule my_route_rule  \
 -widths   {metal7 0.8 metal8 0.8} \
 -spacings {metal7 0.8 metal8 0.8}

set_clock_tree_options \
          -clock_trees clk_i \
                  -target_early_delay 0.1 \
                  -target_skew 0.5 \
                  -max_capacitance 300 \
                  -max_fanout 10 \
                  -max_transition 0.3 \
                  -routing_rule my_route_rule  \
              -layer_list "metal7 metal8"

set_clock_tree_options  -clock_trees clk_i \
                        -buffer_relocation true \
                        -buffer_sizing true \
                        -gate_relocation true \
                        -gate_sizing true \
                        -use_default_routing_for_sinks 1
## To avoid NDR at clock sinks

set_clock_tree_references -references [get_lib_cells */CLKBUF*]

report_clock_tree -settings

## CTS
compile_clock_tree
clock_opt -fix_hold_all_clocks -congestion -operating_condition max

# DEFINING POWER/GROUND NETS AND PINS
derive_pg_connection    -power_net VDD                  \
                        -ground_net VSS         \
                        -power_pin VDD                  \
                        -ground_pin VSS

save_mw_cel -as ${design}_5_cts

###############################################
########### 6. Routing   ######################
###############################################
## Before starting to route, you should add spare cells
insert_spare_cells -lib_cell {NOR2_X1 NAND3_X1 NAND2_X1 INV_X1 MUX2_X1 OAI221_X1 OAI211_X1
OAI22_X1 OAI21_X1 AOI22_X1 SDFFR_X1 BUF_X2 FA_X1 HA_X1  DFFR_X1} \
                  -num_instances 500 \
                  -cell_name SPARE_PREFIX_NAME \
                  -tie
```

```
insert_spare_cells -lib_cell {NOR2_X4 NAND2_X4}                    -num_instances 20
-cell_name SPARE_PREFIX_NAME                    -tie

#Warning: Cell contains tie connections which are not connected to real PG. (MW-349)



set_dont_touch  [all_spare_cells] true
set_attribute [all_spare_cells]  is_soft_fixed true


check_physical_design -stage pre_route_opt;
# dump check_physical_design result to file ./cpd_pre_route_opt_*/index.html
all_ideal_nets
all_high_fanout -nets -threshold 100
check_routeability
#check the tie conncetion warning after this command

set_delay_calculation_options -arnoldi_effort high

set_route_options -groute_timing_driven true \
                  -groute_incremental true \
                  -track_assign_timing_driven true \
                  -same_net_notch check_and_fix


set_si_options -route_xtalk_prevention true\
               -delta_delay true \
               -min_delta_delay true \
               -static_noise true\
               -timing_window true

#set_fix_hold [all_clocks]
#set_prefer -min  [get_lib_cells "*/BUF_X2 */BUF_X1"]
#set_fix_hold_options -preferred_buffer

route_auto -effort high

route_opt -effort high  -stage track -xtalk_reduction -incremental
route_opt -effort high  -incremental

derive_pg_connection    -power_net VDD                  \
                        -ground_net VSS         \
                        -power_pin VDD                  \
                        -ground_pin VSS


save_mw_cel -as ${design}_6_routed

################################################
########### 7. Finishing ######################
################################################
```

```
insert_stdcell_filler -cell_without_metal {FILLCELL_X32 FILLCELL_X16 FILLCELL_X8 FILLCELL_X4
FILLCELL_X2 FILLCELL_X1} \
        -connect_to_power VDD -connect_to_ground VSS



derive_pg_connection    -power_net VDD                  \
                        -ground_net VSS         \
                        -power_pin VDD                  \
                        -ground_pin VSS

save_mw_cel -as ${design}_7_finished

save_mw_cel -as ${design}

################################################
########### 8. Checks and Outputs ############
################################################
#PG
verify_pg_nets -pad_pin_connection all
#routing
verify_zrt_route
#LVS
verify_lvs -ignore_floating_port -ignore_floating_net \
       -check_open_locator -check_short_locator
#DRC
report_design -physical
#congestion
report_congestion
#timing
report_timing

insert_metal_filler -timing_driven -from_metal 1 -to_metal 8
#min and max routing layers

route_eco -search_repair_loop 20 -utilize_dangling_wires -reroute modified_nets_first_then_others

#report_congestion

route_zrt_eco -max_detail_route_iterations 40 -open_net_driven true -reroute
modified_nets_first_then_others -utilize_dangling_wires true

report_congestion

focal_opt    -drc_nets all -effort high
focal_opt    -drc_pins all -effort high



derive_pg_connection    -power_net VDD                  \
                        -ground_net VSS         \
```

```
                        -power_pin VDD                  \
                        -ground_pin VSS

remove_floating_pg -nets VSS -pad_lib_cells {*}
remove_floating_pg -nets VDD -pad_lib_cells {*}


set_write_stream_options -map_layer $sc_dir/tech/strmout/FreePDK45_10m_gdsout.map \
            -output_filling fill \
                        -child_depth 20 \
                        -output_outdated_fill  \
                        -output_pin  {text geometry}

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

 write_sdc design_name.sdc

define_name_rules  no_case -case_insensitive
change_names -rule no_case -hierarchy
change_names -rule verilog -hierarchy
set verilogout_no_tri       true
set verilogout_equation  false


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

extract_rc
#after routing update with route_zrt_eco  use the extract_rc -incremental command to perform
incremental extraction on the changed nets.

write_parasitics -output {./output/cv32e40p_core.spef}


close_mw_cel
close_mw_lib
```

3. Flat flow Codes

2.1 Synthesis

```
set hdlin_sverilog_std 2009
set design cv32e40p_core

set_app_var search_path
"/home/standard_cell_libraries/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Front_End/Liberty/NLD
M \
/home/ahesham/Desktop/NangateOpenCellLibrary_PDKv1_3_v2010_12/lib/Back_End/virtuoso/Nangate
OpenCellLibrary"
```

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

sh rm -rf work
sh mkdir -p work
define_design_lib work -path ./work


analyze -library work -format sverilog ../rtl/cv32e40p_core.sv
elaborate cv32e40p_core -lib work
current_design

check_design
source ./cons/cv32e40p_core.sdc
link
compile_ultra -timing_high_effort_script \
          -gate_clock \
               -retime

compile_ultra -incremental
compile_ultra -incremental
compile_ultra -incremental
compile_ultra -incremental
compile_ultra -incremental


report_area > ./report/synth_area.rpt
report_power > ./report/synth_power.rpt
report_cell > ./report/synth_cells.rpt
report_qor  > ./report/synth_qor.rpt
report_resources > ./report/synth_resources.rpt
report_timing -max_paths 10 > ./report/synth_timing.rpt

write_sdc  output/cv32e40p_core.sdc
define_name_rules  no_case -case_insensitive
change_names -rule no_case -hierarchy
change_names -rule verilog -hierarchy
set verilogout_no_tri        true
set verilogout_equation  false

write -hierarchy -format verilog -output output/cv32e40p_core.v
write -f ddc -hierarchy -output output/cv32e40p_core.ddc

exit
```

2.1 Place and Route (same as Hierarchical flow)