DIGITAL IMPLEMENTATION OF THE RISC-V BASED OPENPULP CORE

By

Hazem Mohamed Fathy

Omar Abd El-Rahman Ismail

Mohamed Saber Mohamed

Mostafa Othman Saad

Ahmed Hassan Maarouf

Ahmed Sayed Mohamed

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
BACHELOR OF SCIENCE
in
Electronics and Communications Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2020

DIGITAL IMPLEMENTATION OF THE RISC-V BASED OPENPULP CORE

By

Hazem Mohamed Fathy
Omar Abd El-Rahman Ismail
Mohamed Saber Mohamed
Mostafa Othman Saad
Ahmed Hassan Maarouf
Ahmed Sayed Mohamed

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the
BACHELOR OF SCIENCE
in
Electronics and Communications Engineering

Under the Supervision of

Dr. Hassan Mostafa	**Dr. Amin Nassar**
Associate Professor	Emeritus Professor
Electronics and Communications Department	Electronics and Communications Department
Faculty of Engineering, Cairo university	Faculty of Engineering, Cairo university

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2020

Contents

List of Figures

List of Tables

List of Abbreviations

Ri5cy	another name of OpenPULP core
IIS	Integrated Systems Laboratory
EEES	Energy effecient Embedded Systems
ASIC	Application specific integrated circuit
FPGA	Field Programmable Gate Arrays
ISA	instruction set architecture
PULP	Parallel Ultra Low Power
SPI	Serial Peripheral Interface
UART	Universal asynchronous receiver-transmitter
I2C	inter-integrated circuit
CPI	camera parallel interface
GPIOs	general purpose inputs/outputs
PWM	Pulse width modulation
LSU	Load Store Unit
PMP	Physical Memory Protection
ALU	Arthematic Logic Unit
FPU	Floating Point Unit
APU	Accelerated Processing Uni
FCSR	floating point status and control register

CSR	status and control register
RTL	Register Transfer Logic
SAED	Synopsys Armenia Education Department
GDSII	data exchange of integrated circuit or IC layout artwork
DC	Design Compiler
WNS	Worst Negative Slack
CMOS	Complementary metal–oxide–semiconductor
PVT	process voltage temperature
QoR	Quality of results
PDK	process design kit
DEF	Design Exchange Format
DRC	Design Rule Check
PnR	Place and Route
PNA	Power Network Analysis
CTS	clock Tree Synthesis
STA	Static Timing Analysis
ICG	integrated clock gating cells
FEC	Formal Equivalence Checking
LVT	Low Voltage Threshold
RVT	Regular Voltage Threshold
HVT	High Voltage Threshold
ECO	Engineering Change Order
MCMM	Multi-corner Multi-mode
OCV	On Chip Variation
PCB	printed circuit board

HDL	Hardware Description Language
VHDL	very high speed integrated circuit Hardware Description Language
EDA	Electronic design automation
LDM	Linear Delay Model
NLDM	non-linear delay mode
WLM	Wire Load Model
P&G	power and Ground
NDR	Non Default Rule
HFNS	High Fanout Net Synthesis
LEC	Logic Equivalence Check
DUA	Design Under Analysis
SDC	Synopsys Design Constraints
ASCII	American Standard Code for Information Interchange
PLL	Phase locked loop
FIFO	First In First Out
TTR	Time to results

Abstract

ASIC Physical Design of the RISC-V based OpenPULP "Open Parallel Ultra-low-power" Core, the aim of OpenPULP core is to satisfy the computational demands of IoT applications requiring flexible processing of data streams. The project aims to deliver the GDSII file by going through the RTL-to-GDSII flow, "SAED 32/28nm" PDK was used through the flow.

The core was implemented with three different synthesis styles (Topographical, Flat, Hierarchical) using Synopsys Design Compiler, followed by all main steps of Place&Route flow using Synopsys IC Compiler, formal equivalence checking is performed after PnR process to ensure that both Pre-layout netlist and Post-layout netlist exhibit the same behavior using Synopsys Formality tool, and then Post-layout STA is done with Synopsys PrimeTime to ensure that design meets all timing requirements at a wide range of PDK operating corners, and then results were compared between the three implementation flows based on area, power and synthesis runtime.

Chapter 1

Introduction

1.1 RISC-V

RISC-V (pronounced "risk-five") is an open, free ISA enabling a new era of processor innovation through open standard collaboration. Born in academia and research, 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 for RISC-V is to become a universal instruction set architecture (ISA):

1. It should suit all sizes of processors, from the tiniest embedded controller to the fastest high-performance computer.

2. It should work well with a wide variety of popular software stacks and programming languages.

3. It should accommodate all implementation technologies: Field-Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), full-custom chips, and even future device technologies.

4. It should be stable, in that the base ISA should not change. More importantly, the ISA cannot be discontinued, as has happened in the past to proprietary ISAs such as the AMD Am29000, the Digital Alpha, the Digital VAX, the Hewlett Packard PA-RISC, the Intel i860, the Intel i960, the Motorola 88000, and the Zilog Z8000.

A 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.

1.1.1 Modular vs. Incremental ISAs

The conventional approach to computer architecture is incremental ISAs, where new processors must implement not only new ISA extensions but also all extensions of the past. The purpose is to maintain backwards binary -compatibility so that binary versions of decades-old programs can still run correctly on the latest processor. This requirement, when combined with marketing appeal of announcing new instructions with a new generation of processors, has let to ISAs that grow substantially in size with age. For example, the 80x86. It dates back to 1978, yet it has added about three instructions per month over its long lifetime.

This convention means that every implementation of the x86-32 must implement the mistakes of the past extensions, even when they no longer make sense. Beyond being recent and open, RISC-V is unusual since, unlike almost all prior ISAs, it is modular. At the core is a base ISA, called RV32I, which runs a full software stack. RV32I is frozen and will never change, which gives compiler writers, operating system developers, and assembly language programmers a stable target. The modularity comes from optional standard extensions that hardware can include or not depending on the needs of the application.

This modularity enables very small and low energy implementation of RISC-V, which can be critical for embedded applications.

1.1.2 RISC-V Instructions

RISC-V instructions are in the following categories:

- Base Integer (RV32I)
- Multiply and Divide (RV32M)
- Single/Double Floating Point (RV32FD)
- Atomic (RV32A)
- Compressed (RV32C)
- Vector (RV32V)
- 64-bit Address (RV64)

RV32I is the category that contains six base instruction formats: R-type for register-register operations; I-type for short immediates and loads; S-type for stores; B-type for conditional branches; U-type for long immediates; J-type for unconditional jumps. **RV32I** registers "**x**" are 32x32 registers, 31 registers (x1 to x31) are used for base integer operations, while 1st register (x0) is always having the value of 0. Dedicating a register to zero is surprisingly large factor in simplifying the RISC-V ISA.

RV32M adds integer multiply and divide instructions to **RV32I**. It has instructions for both signed and unsigned integers: divide (div) and divide unsigned (divu), which place the quotient into the destination register. The multiply is more complicated than divide because the size of the product is the sum of the sizes of the multiplier and the multiplicand; multiplying two 32-bit numbers yields a 64-bit product. To produce a properly signed or unsigned 64-bit product, RISC-V has four multiply instructions. To get the integer 32-bit product – the lower 32-bits of the full product – use “mul”. To get the upper 32 bits of the 64-bit product, use “mulh” if both operands are signed, “mulhu” if both operands are unsigned, and “mulhsu” if one is signed, and the other is unsigned.

Although **RV32F** (single-precision floating point) and **RV32D** (double-precision floating point) are separate, optional instruction set extensions, they are often included together in **RV32FD**, Given single- and double-precision (32- and 64-bit) versions for nearly all floating point instructions. **RV32FD** use 32 separate “f” registers instead of the “x” registers. The main reason for the two sets of registers is that processors can improve performance by doubling the register capacity and bandwidth by having two sets of registers without increasing the space for the register specifier in the cramped RISC-V instruction format. The major impact on the instruction set is to have new instructions to load and store the f registers and to transfer data between the **x** and **f** registers.

RV32A has two types of atomic operations for synchronization:

- Atomic memory operation (AMO)
- Load reserved / store conditional

The AMO instructions atomically perform an operation on an operand in memory and set the destination register to the original memory value. Atomic means there can be no interrupt between the read and the write of memory, nor could other processors modify the memory value between the memory read and write of the AMO instruction. Load reserved and store conditional provide an atomic operation across two instructions. Load reserved reads a word from memory, writes it to the destination register, and records a reservation on that word in memory. Store conditional stores a word at the address in a source register provided there exists a load reservation on that memory address. It writes zero to the destination register if the store succeeded, or a nonzero error code otherwise.

RV32C takes a novel approach: every short instruction must map to one single standard 32-bit RISC-V instruction. Moreover, only the assembler and linker are aware of the 16-bit instructions, and it is up to them to replace a wide instruction with its narrow cousin. The RISC-V architects chose the instructions in the RVC extension to obtain good code compression across a range of programs.

Virtually every integer and floating-point instruction from previous sections has a **RV32V** vector version. There are several types of each vector instruction depending on whether the source operands are all vectors (.vv suffix) or a vector source operand and a scalar source operand (.vs suffix). A scalar suffix means an x or f register is an operand along with vector register (v). **RV32V** adds 32 vector registers, whose names start with "**v**", but the number of elements per vector register varies. That number depends on both the width of the operations and on the amount of memory dedicated to vector registers, which is up to the processor designer.

The ISAs typically add only a few word, doubleword, or long versions of the 32-bit instructions and expand all the registers, including the PC, to 64-bits. Thus, "sub" instruction in **RV64I** subtracts two 64-bit numbers rather than two 32-bit numbers as in **RV32I**. **RV64** is a close but actually different ISA than **RV32**; it adds a few instructions and the base instructions do slightly different things due to the increase in operands size.

1.1.3 RISC-V Data Formats

The RISC-V Architecture recognizes these fundamental data types:

- Signed Integer: 8,16,32 and 64 bits
- Unsigned Integer: 8,16,32 and 64 bits
- Floating-Point: 32 and 64 bits

The widths of data types are:

- Byte: 8 bits
- Halfword: 16 bits
- Word: 32 bits
- Doubleword: 64 bits

1.2 Thesis Objective

RTL-to-GDSII Flow plays an important role in the development process of electronic chips over the world. Due to the huge competition in this field and huge customer demands, the technology in this field is developing at a fast pace towards smaller, faster and more complex devices which is making it harder for a newcomer to cope up with this field without learning and understanding the basics first. CAD tools nowadays offer more than one ASIC design style to satisfiy huge customer demands and improve QoR and TTR of ASIC designs. Objective of this thesis is to demonstrate the impact of different design flows on an open-source RTL Design of OpenPULP Core (RI5CY) using the 32nm CMOS technology.

This objective is achieved by going through RTL-to-GDSII flow starting from three different logic synthesis flows which are hierarchical, flat and topographical synthesis flows, passing after that with all necessary Place And Route steps and Post-layout STA to ensure good timing performance over various operating conditions.

1.3 Thesis Map

This thesis documentation is divided into five chapters and an Appendix, chapter 1 is an introductory chapter that provides a quick background about the new-born RISC-V ISA at the time of this thesis, and also it covers the thesis objective and this thesis map. Chapter 2 demonstrates the structure of the OpenPULP system and its implementation history with a brief description of each system component and also demonstrates the structure of the OpenPULP core known as RI5CY and at the end of this chapter, we provide an idea of the proposed design which will be implemented with three different flows.

Appendix A shows methodology of each flow in general and not depending on a specific design. Chapter 3 describes the attempt to implement OpenPULP core using different synthesis flows based on the methodologies discussed in Appendix A and chapter 4 gathers all the achieved results obtained at each step for every flow of these three flows. Chapter 5 concludes the impact of these flows on the same design with respect to three performance metrics which are area, power and synthesis runtime, and shows a comparison between the famous ARM Corte-M4 with RI5CY core implemented with each of the three flows.

Chapter 2

OpenPULP

The Parallel Ultra Low Power (PULP) is a joint project between the Integrated Systems Laboratory (IIS) of ETH Zurich and the Energy-efficient Embedded Systems (EEES) group of UNIBO to develop an open, scalable Hardware and Software research platform with the goal to break the pJ/op barrier within a power envelope of a few mW.

The PULP platform is a multi-core platform achieving leading-edge energy-efficiency and featuring widely-tunable performance. The aim of PULP is to satisfy the computational demands of IoT applications requiring flexible processing of data streams generated by multiple sensors, such as accelerometers, low-resolution cameras, microphone arrays, vital signs monitors. As opposed to single-core MCUs, a parallel ultra-low-power programmable architecture allows to meet the computational requirements of these applications, without exceeding the power envelope of a few mW typical of miniaturized, battery-powered systems. Moreover, OpenMP, OpenCL and OpenVX are supported on PULP, enabling agile application porting, development, performance tuning and debugging.

The PULP platform have intentionally taken an open source approach from the very onset of the project and so far, PULP have released efficient 32 and 64bit implementations based on the open-source RISC-V instruction set architecture, peripherals and complete systems starting from simple micro-controllers, to the state-of-the-art OpenPULP release which sets a new bar for low-power multicore IoT processors.

2.1 OpenPULP Architecture

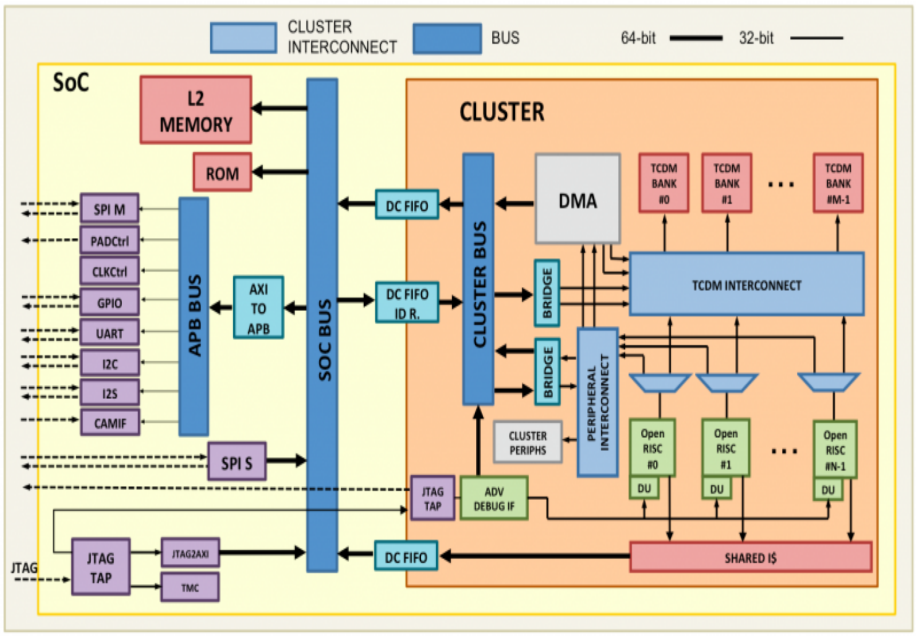


Figure 2.1: PULP Overview

FIGURE 2.1 shows a block diagram of OpenPULP Architecture. OpenPULP – or just PULP – is hierarchical, demand-driven architecture enables ultra-low-power operation by combining:

- A fabric controller (FC) core for control, communications and security functions. This can be viewed like a classic MCU.

- A cluster of 8 cores with an architecture optimized for the execution of vectorized and parallelized algorithms.

All cores and peripherals are power switchable and voltage and frequency adjustable on demand. DC/DC regulators and clock generators with ultra-fast reconfiguration times are integrated. This allows PULP to adapt extremely quickly to the processing/energy requirements of a running application.

All elements share access to an L2 memory area. The cluster cores share access to an L1 (TCDM) memory area and instruction cache. Multiple DMA units allow autonomous, fast, low power transfers between cluster L1 and L2 memory and between L2 memory and external peripherals.

2.1.1 Cores

All 8 cores of the cluster share the RV32IMFCXpulp instruction set architecture, while the fabric controller can be configured as either RV32IMC or RV32IMFCXpulp. The I (integer), C (compressed instruction), M (Multiplication and division) and a portion of the supervisor ISA subsets are supported. These standard instruction sets are extended with specific instructions to optimize the performance of signal processing and machine learning algorithms. These extensions include zero-overhead hardware loops, pointer post/pre-modified memory accesses, instructions mixing control flow with computation (min, max, etc), multiply/subtract and accumulate, vector operations, fixed-point operations, bit manipulation and dot product. All of these instruction extensions are optimized by the compiler or can be used ‘by hand’.

2.1.2 Memory Areas

There are 2 different levels of memory internal to PULP. A larger level 2 area of 512kB which is accessible by all processors and DMA units a smaller (Tightly Coupled Device Memory - TCDM) level 1 area shared by all the cluster cores (128kB). The cluster level 1 memory is banked and connected to the cluster cores via a logarithmic interface that is sized to provide single cycle access in 98% of cases. L2 memory is divided into 4 128kB blocks.

Cluster L1 memory supports test-an-set functionality. The test-and-set is an atomic instruction used to write to a memory location and return its old value as a single atomic (i.e. non-interruptible) operation. If multiple processes access the same memory area and if a process is currently performing a test-and-set, no other process may begin another test-and-set until the first process is done. The instruction caches of the FC (1kB) and cluster (4kB) will automatically cache instructions as needed. The cluster instruction cache is shared between all the cores in the cluster. Generally, the cluster cores will be executing the same area of code on different data hence the shared cluster instruction cache exploits this to reduce memory accesses for loading instructions. The combination of a high-speed shared data and instruction memory in the cluster provides an ideal memory architecture for the execution of code implementing parallelized algorithms. PULP can also access external memory areas over the HyperBus (Flash or RAM) or quad-SPI (Flash) interfaces. We refer to RAM accessed over the HyperBus or quad-SPI interfaces as level 3 memory. Since the energy and performance cost of accessing external RAM over external buses is very high compared to the internal memory, generally this should be avoided as much as possible. In consequence, program code is loaded from external flash at boot into the L2 memory area.

2.1.3 Debug Architecture

PULP contains debug functionalities to help the developer observing/controlling application code execution. Debug functionalities are accessible through JTAG or SPI Slave interfaces using a GDB server. They permit access to the FC RI5CY core debug unit and Cluster RI5CY cores debug units to break program execution. A direct connection from JTAG or SPI Slave to PULP bus architecture also allows access to all the memory mapped registers of the chip.

2.1.4 Data Types Supported

The memories are byte addressable so every single data type whose size is a multiple of bytes can be supported either natively if the number of bytes is less or equal than 4 or through software emulation if it is larger.

2.1.5 Event Units

Two event units (EU) are available in PULP. One for the FC and one for the cluster. The EU allows the RI5CY cores to be put into sleep mode when waiting for an event to occur. In the EUs, the way of treating incoming events can be controlled. The EU can be instructed to react instantly by jumping to an interrupt routine or to delegate the treatment of the event to a software event task controller.

2.1.6 DMA (Direct Memory Access)

The DMA unit allows the transfer of data between L2 and cluster L1 memory areas. 8 channels can be programmed. Channels can be 1D/2D on the L2 memory and 1D on the cluster L1 side.

2.1.7 Micro DMA

The micro DMA (UDMA) provides direct transfer of data between L2 memory and the different interfaces provided in PULP connected to UDMA. It helps in relaxing the execution load of FC RI5CY core. Up to 11 channels can be managed by the UDMA:

- Camera to L2
- I2S0 to L2
- I2S1 to L2
- I2C0 from/to L2
- I2C1 from/to L2
- UART from/to L2
- SPIM0 from/to L2

The width of transfers can be selected between 8, 16 or 32 bits. Up to 128kB can be transferred during a single transaction (8kB for SPIM). In the general case, transactions can be bidirectional but depending on the interface, in some cases only one direction is available.

2.1.8 SPI master (serial peripheral interface)

Up to 2 SPI master interfaces are available:

- One Single/Quad SPI (Master) which is able to communicate at speeds up to 50Mbits/s.
- One Single SPI (Master) which is able to communicate at speeds up to 50Mbits/s.

2.1.9 UART (universal asynchronous receiver-transmitter)

One UART interface is available with up to 1Mbits/s baud rate. No dedicated synchronization (CTS/RTS/DTR/DSR/DCD) signals are provided.

2.1.10 I2C (inter-integrated circuit)

Up to 2 I2C (Inter-Integrated Circuit) are provided in PULP. They support multi-master, multi-slave, single-ended modes. I2C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL), pulled up with resistors.

2.1.11 I2S (digital microphone interface) RX

Up to 2 RX I2S are available for connecting digital audio devices to PULP chip. Up to 4 digital microphones (either PCM or PDM format) can be directly connected to PULP. These are able to communicate at speeds of up to 10Mbits/s.

2.1.12 CPI (camera parallel interface)

The CPI interface is 8 bits wide and can communicate at speeds up to 50MHz. VSYNC, HSYNC and PCLK are provided by the camera.

2.1.13 GPIOs (general purpose inputs/outputs)

Up to 32 digital general purpose I/O's are available. Each I/O can be configured either as an input or output. Interrupts on event can be generated on the rising or the falling or both edges for all I/O's. I/O's can also be configured to act as an external wake up signal.

2.1.14 SPI slave

One SPI slave interface directly connected to PULP bus architecture is available. It is able to access all memory mapped registers and L2 memory. It can communicate at speeds up to 50Mbits/s.

2.1.15 Basic Timers

2 basic timers are available, one connected to the FC and the other to the cluster. They can be configured either as 2 x 32-bit timers or as a single 64-bit timer. The basic timers can either run continuously or trigger just once. Events can be generated using a compare match.

Clock sources of these timers can be the:

- FLL

- FLL with pre-scaler
- 32.768kHz reference clock

2.1.16 Advanced PWM Timers

4 advanced PWM timers are available in the SOC domain. Each of them provides 4 output signal channels that can be used for PWM signal generation with multiple configuration possibilities.

2.1.17 RTC

A real-time clock is available. It provides a set of continuously running counters which can be used with suitable software to provide a clock calendar function. It provides also alarm and a periodic interrupt features. It is clocked by the 32.768 kHz external crystal.

2.1.18 Performance Counters

Each RI5CY cores of the FC and the cluster provide a performance counter. These 32-bit counters can be configured to count the:

1. Total number of cycles (also includes the cycles where the core is sleeping)
2. Number of cycles the core was active (not sleeping)
3. Number of instructions executed
4. Number of load data hazards
5. Number of jump register data hazards

6. Number of cycles waiting for instruction fetches, i.e. number of instructions wasted due to non-ideal caching
7. Number of data memory loads executed. Misaligned accesses are counted twice
8. Number of data memory stores executed. Misaligned accesses are counted twice
9. Number of unconditional jumps (j, jal, jr, jalr)
10. Number of both taken and not taken branches
11. Number of taken branches
12. Number of compressed instructions executed
13. Number of memory loads to EXT executed. Misaligned accesses are counted twice. Every non-L1 access is considered external (cluster only)
14. Number of memory stores to EXT executed. Misaligned accesses are counted twice. Every non-L1 access is considered external (cluster only)
15. Number of cycles used for memory loads to EXT. Every non-L1 access is considered external (cluster only)
16. Number of cycles used for memory stores to EXT. Every non-L1 access is considered external (cluster only)
17. Number of cycles wasted due to L1/log-interconnect contention (cluster only)
18. Number of cycles wasted due to CSR access

2.2 OpenPULP Core

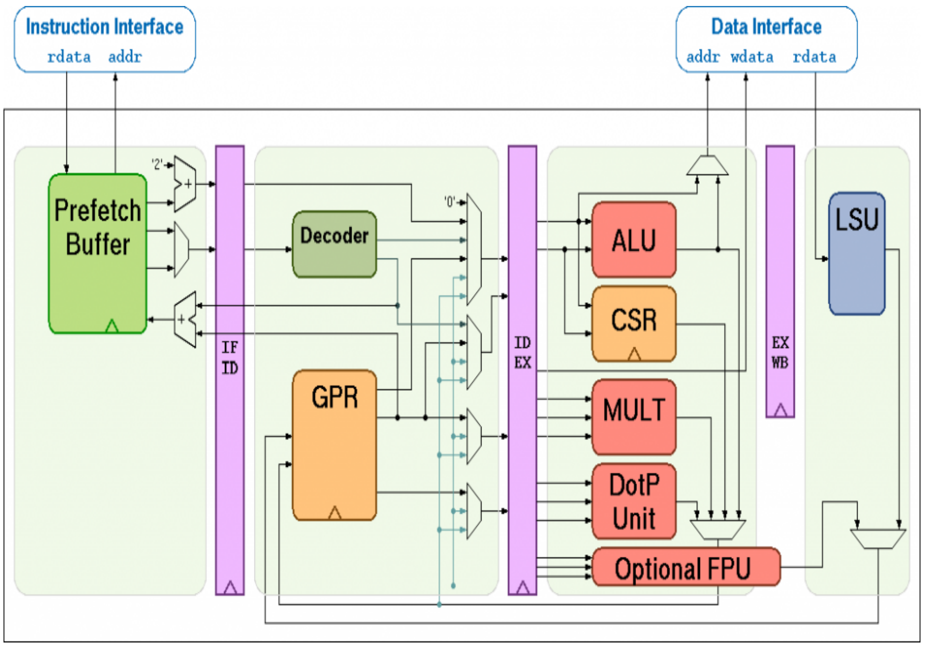


Figure 2.2: OpenPULP Core block diagram

FIGURE 2.2 shows a block diagram of the OpenPULP Core. OpenPULP Core – or RI5CY – is a 4-stage in-order 32b RISC-V processor core. The ISA of RI5CY was extended to support multiple additional instructions including hardware loops, post-increment load and store instructions and additional ALU instructions that are not part of the standard RISC-V ISA.

2.2.1 Supported Instruction Set

RI5CY supports the following instructions:

- Full support for RV32I Base Integer Instruction Set.
- Full support for RV32C Standard Extension for Compressed Instructions.
- Full support for RV32M Integer Multiplication and Division Instruction Set Extension.
- Optional full support for RV32F Single Precision Floating Point Extensions.
- PULP specific extensions:
  - Post-Incrementing load and stores.
  - Multiply-Accumulate extensions.
  - ALU extensions.
  - Hardware Loops.

2.2.2 LSU (Load-Store Unit)

The LSU of the core takes care of accessing the data memory. Load and stores on words (32 bit), half words (16 bit) and bytes (8 bit) are supported.

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

Table 2.1: LSU Signals

The protocol that is used by the LSU to communicate with a memory works as follows:

The LSU provides a valid address in data_addr_o and sets data_req_o high. The memory then answers with a data_gnt_i set high as soon as it is ready to serve the request. This may happen in the same cycle as the request was sent or any number of cycles later. After a grant was received, the address may be changed in the next cycle by the LSU. In addition, the data_wdata_o, data_we_o and data_be_o signals may be changed as it is assumed that the memory has already processed and stored that information. After receiving a grant, the memory answers with a data_rvalid_i set high if data_rdata_i is valid. This may happen one or more cycles after the grant has been received. Note that data_rvalid_i must also be set when a write was performed, although the data_rdata_i has no meaning in this case.

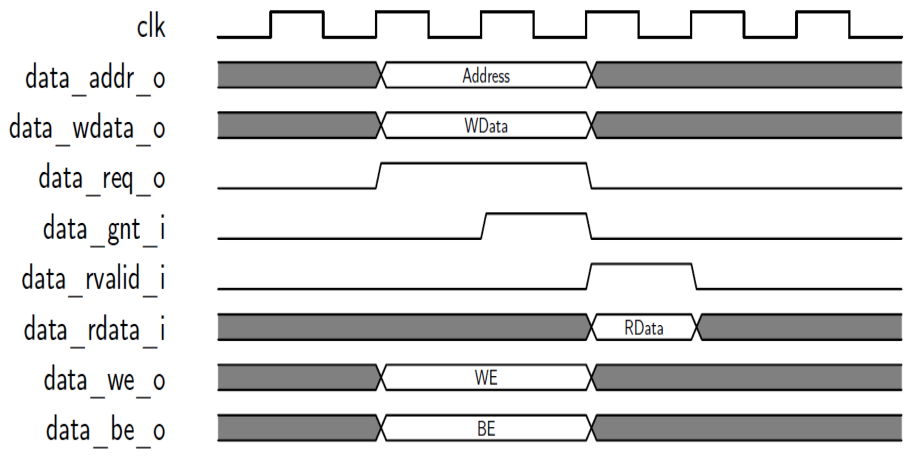


Figure 2.3: Basic Memory Transaction

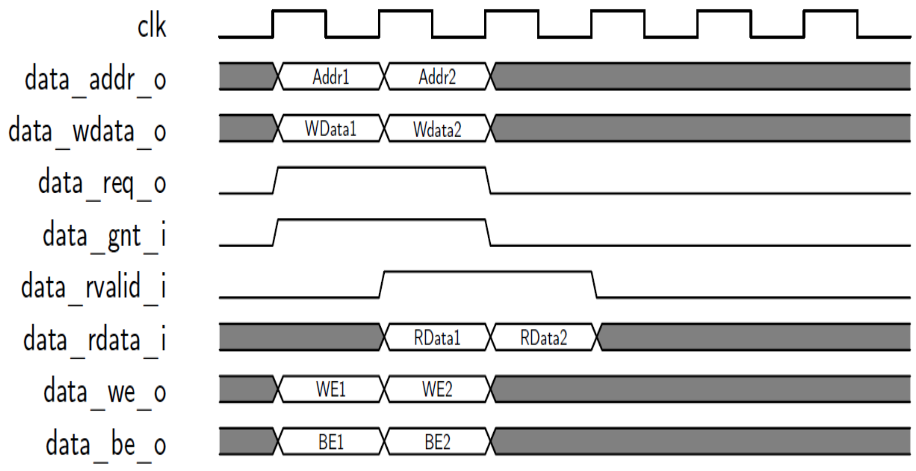


Figure 2.4: Back-to-back Memory Transaction

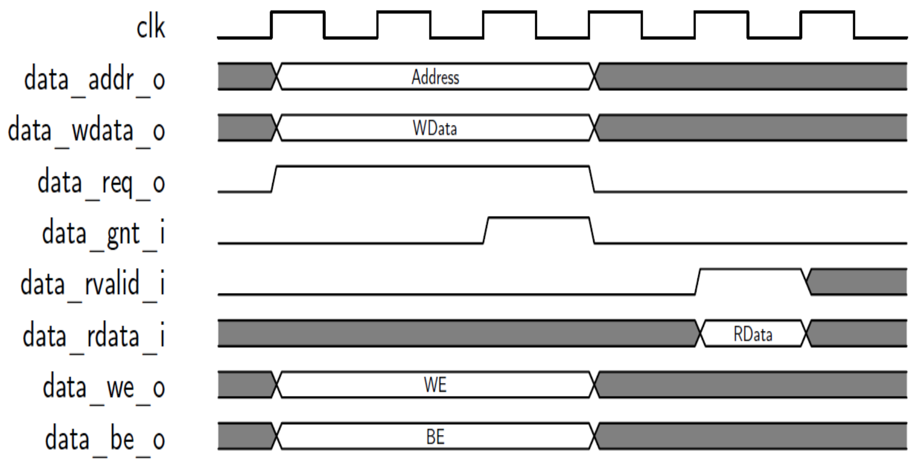


Figure 2.5: Slow Response Memory Transaction

FIGURE 2.3, FIGURE 2.4 and FIGURE 2.5 show example-timing diagrams of LSU's protocol.

2.2.3 Physical Memory Protection (PMP) Unit

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

2.2.4 Post-Incrementing Load and Store Instructions

Post-incrementing load and store instructions perform a load/store operation from/to the data memory while at the same time increasing the base address by the specified offset. For the memory access, the base address without offset is used. Post-incrementing load and stores reduce the number of required instructions to execute code with regular data access patterns, which can typically be found in loops. These post-incrementing load/store instructions allow the address increment to be embedded in the memory access instructions and get rid of separate instructions to handle pointers. Coupled with hardware loop extension, these instructions allow to reduce the loop overhead significantly.

2.2.5 Misaligned Accesses

The LSU is able to perform misaligned accesses, meaning accesses that are not aligned on natural word boundaries. However, it needs to perform two separate word-aligned accesses internally. This means that at least two cycles are needed for misaligned loads and stores.

2.2.6 Instruction Fetch Unit

The instruction fetcher of the core is able to supply one instruction to the ID stage per cycle if the instruction cache or the instruction memory is able to serve one instruction per cycle. The instruction address must be half-word-aligned due to the support of compressed instructions. It is not possible to jump to instruction addresses that have the LSB bit set. For optimal performance and timing closure reasons, a prefetcher is used which fetches instruction from the instruction memory, or instruction cache.

Signal	**Direction**	**Description**
instr_req_o	Output	Request ready, must stay high until instr_gnt_i is high for one cycle
instr_addr_o[31:0]	Output	Address
instr_rdata_i[31:0]	Input	Data read from memory
instr_rvalid_i	Input	instr_rdata_i holds valid data when instr_rvalid_i is high. This signal will be high for exactly one cycle per request
instr_gni_i	Input	The other side accepted the request. instr_addr_o may change in the next cycle

Table 2.2: Instruction Fetch Signals

TABLE 2.2 shows the signals used in the core fetcher.

There are two prefetch flavors available:

1. 32-Bit word prefetcher. It stores the fetched words in a FIFO with three entries.
2. 128-Bit cache line prefetcher. It stores one 128-bit wide cache line plus 32-bit to allow for cross-cache line misaligned instructions.

The protocol used to communicate with the instruction cache or the instruction memory is the same as the protocol used by the LSU.

2.2.7 Multiply-Accumulate

RI5CY uses a single-cycle 32-bit x 32-bit multiplier with a 32-bit result. All instructions of the RISC-V M instruction set extension are supported. The multiplications with upper-word result (MSP of 32-bit x 32-bit multiplication), take 4 cycles to compute. The division and remainder instructions take between 2 and 32 cycles. The number of cycles depends on the operand values. Additionally, RI5CY supports non-standard extensions for multiply-accumulate and half-word multiplications with an optional post-multiplication shift.

2.2.8 PULP ALU Extensions

RI5CY supports advanced ALU operations that allow to perform multiple instructions that are specified in the base instruction set in one single instruction and thus increases efficiency of the core. For example, those instructions include zero-/sign-extension instructions for 8-bit and 16-bit operands, simple bit manipulation/counting instructions and min/max/avg instructions. The ALU does also support saturating, clipping, and normalizing instructions which make fixed-point arithmetic more efficient.

2.2.9 Optional private Floating-Point Unit (FPU)

It is possible to extend the core with a private FPU, which is capable of performing all RISC-V floating-point operations that are defined in the RV32F ISA extensions. The latency of the individual instructions and information where they are computed are summarized in Table 3. FP extensions can be enabled by setting the parameter of the top-level file "riscv_core.sv" to one.
The FPU is divided into three parts:

1. A simple FPU of 10kGE complexity, which computes FP-ADD, FP-SUB

and FP-casts.

2. An iterative FP-DIV/SQRT unit of 7 kGE complexity, which computes FP-DIV/SQRT operations.
3. An FP-FMA unit which takes care of all fused operations. This unit is currently only supported through a Synopsys Design Ware instantiation, or a Xilinx block for FPGA targets.

FP CSR

When using floating-point extensions, the standard specifies a floating-point status and control register (fcsr) which contains the exceptions that occurred since it was last reset and the rounding mode. fflags and frm can be accessed directly or over fcsr which is mapped to those two registers. Since RISCY includes an iterative div/sqrt unit, its precision and latency can be controlled over a custom csr (fprec). This allows faster division / square-root operations at the lower precision. By default, the single-precision equivalents are computed with a latency of 8 cycles.

Floating-point Performance Counters

Some specific performance counters have been implemented to profile FP-kernels.

2.2.10 PULP Hardware Loop Extensions

To increase the efficiency of small loops, RI5CY supports hardware loops. Hardware loops make it possible to execute a piece of code multiple times, without the overhead of branches or updating a counter. Hardware loops involve zero stall cycles for jumping to the first instruction of a loop.

A hardware loop is defined by its start address (pointing to the first instruction in the loop), its end address (pointing to the instruction that will be executed last in the loop) and a counter that is decremented every time the loop body is executed. RI5CY contains two hardware loop register sets to support nested hardware loops, each of them can store these three values in separate flip flops which are mapped in the CSR address space. If the end address of the two hardware loops is identical, loop 0 has higher priority and only the loop counter for hardware loop 0 is decremented. As soon as the counter of loop 0 reaches 1 at an end address, meaning it is decremented to 0 now, loop 1 gets active too. In this case, both counters will be decremented and the core jumps to the start of loop 1.

In order to use hardware loops, the compiler needs to setup the loop beforehand with the following instructions. Note that the minimum loop size is two instructions and the last instruction cannot be any jump or branch instruction. For debugging and context switches, the hardware loop registers are mapped into the CSR address space and thus it is possible to read and write them via csrr and csrw instructions. Since hardware loop registers could be overwritten in when processing interrupts, the registers have to be saved in the interrupt routine together with the general purpose registers.

CSR Address				Hex	Name	Acc.	Description
11:10	**9:8**	**7:6**	**5:0**				
01	11	10	110000	0x7C0	Ipstart[0]	R/W	Hardware Loop 0 Start
01	11	10	110001	0x7C1	Ipendt[0]	R/W	Hardware Loop 0 End
01	11	10	110010	0x7C2	Ipcount[0]	R/W	Hardware Loop 0 Counter
01	11	10	110000	0x7C4	Ipstart[1]	R/W	Hardware Loop 1 Start
01	11	10	110001	0x7C5	Ipendt[1]	R/W	Hardware Loop 1 End
01	11	10	110010	0x7C6	Ipcount[1]	R/W	Hardware Loop 1 Counter

Table 2.3: Hardware-Loop CSR Mapping

2.2.11 Pipeline

RI5CY has a fully independent pipeline, meaning that whenever possible data will propagate through the pipeline and therefore does not suffer from any unneeded stalls. The pipeline design is easily extendable to incorporate out-of-order completion. E.g., it would be possible to complete an instruction that only needs the EX stage before the WB stage, that is currently blocked waiting for an rvalid, is ready. Currently this is not done in RI5CY, but might be added in the future.

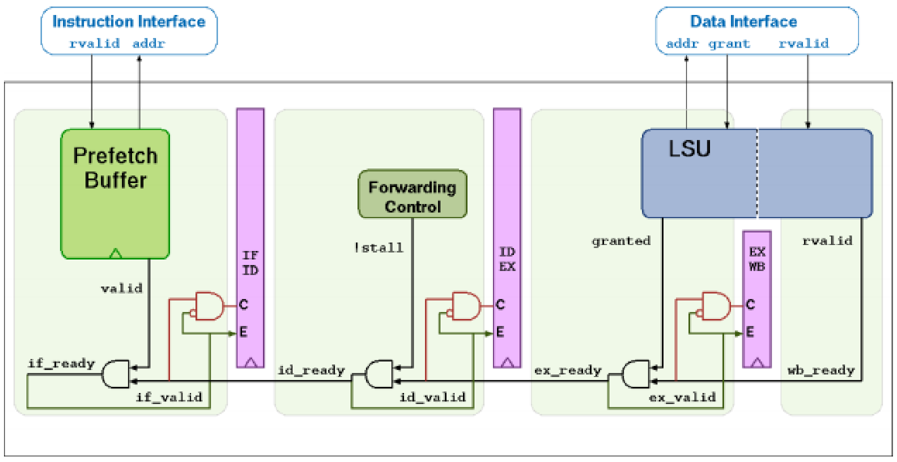


Figure 2.6: RI5CY Pipeline.

FIGURE 2.6 shows the relevant control signals for the pipeline operation. The main control signals, the ready signals of each pipeline stage, are propagating from right to left. Each pipeline stage has two control inputs: an enable and a clear. The enable activates the pipeline stage and the core moves forward by one instruction. The clear removes the instruction from the pipeline stage as it is completed.

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

2.2.12 Register File

RI5CY has 31 _ 32-bit wide registers which form registers x1 to x31. Register x0 is statically bound to 0 and can only be read, it does not contain any sequential logic.

There are two flavors of register file available:

1. Latch-based
2. Flip-flop based

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

Latch-based Register File

The latch-based register file contains manually instantiated clock gating cells to keep the clock inactive when the latches are not written. It is assumed that there is a clock gating cell for the target technology that is wrapped in a module called cluster_clock_gating and has the following ports:

- clk_i: Clock Input
- en_i: Clock Enable Input
- test_en_i: Test Enable Input (activates the clock even though en_i is not set)
- clk_o: Gated Clock Output

FPU Register File

In case the optional FPU is instantiated, the register file is extended with an additional register bank of 32 registers f0-f31. These registers are stacked on top of the existing register file and can be accessed concurrently with the limitation that a maximum of three operands per cycle can be read. Each of the three operands addresses is extended with an fp_reg_sel signal which is generated in the instruction decoder when a FP instruction is decoded. These additional signals determines if the operand is located in the integer or the floating point register file. Forwarding paths, and write-back logic are shared for the integer and floating-point operations and are not replicated.

2.2.13 Control and Status Registers

RI5CY does not implement all control and status registers specified in the RISC-V privileged specifications, but is limited to the registers that were needed for the PULP system. The reason for this is that we wanted to keep the footprint of the core as low as possible and avoid any overhead that we do not explicitly need.

CSR Address				Hex	Name	Acc.	Description
11:10	**9:8**	**7:6**	**5:0**				
00	11	10	000000	0x300	MSTATUS	R/W	Machine Status
00	11	00	000101	0x305	MTVEC	R	Machine Trap-Vector Base Address
00	11	01	000001	0x341	MEPC	R/W	Machine Exception Program Counter
00	11	01	000010	0x342	MCAUSE	R/W	Machine Trap Cause
01	11	00	0xxxxx	0x780-0x79F	PCCRs	R/W	Performance Counter Counter Registers
01	11	10	100000	0x7A0	PCER	R/W	Performance Counter Enable
01	11	10	100001	0x7A1	PCMR	R/W	Performance Counter Mode
01	11	10	110xxx	0x7B0-0x7B7	HWLP	R/W	Hardware Loop Registers
11	00	00	010000	0xC10	PRIVLV	R	Privilege Level
00	00	00	010100	0x014	UHARTID	R	Hardware Thread ID
11	11	00	010100	0xF14	MHARTID	R	Hardware Thread ID
01	11	10	110000	0x7B0	DCSR	R/W	Debug Control and Status
01	11	10	110001	0x7B1	DPC	R/W	Debug PC
01	11	10	110010	0x7B2	DSCRATCH0	R/W	Debug Scratch Register 0
01	11	10	110011	0x7B3	DSCRATCH1	R/W	Debug Scratch Register 1

Table 2.4: Control and Status Register Map

2.2.14 Exceptions and Interrupts

RI5CY supports interrupts, exceptions on illegal instructions and (if enabled) on PMP filtered requests on the data and instruction bus. The base address of the interrupt vector table is given by the mtvec address. As RI5CY supports only vectorized interrupts, the interrupt 0 is reserved for exceptions as illegal instructions, ecall and instruction or data prohibited accesses.

Interrupts

Interrupts can only be enabled/disabled on a global basis and not individually. It is assumed that there is an event/interrupt controller outside of the core that performs masking and buffering of the interrupt lines. The global interrupt enable is done via the CSR register MSTATUS. Multiple interrupts requests are assumed to be handled by event/interrupt controller. When an interrupt is taken, the core gives an acknowledge signal to the event/interrupt controller as well as the interrupt id taken.

Exceptions

The illegal instruction exception, ecall instruction exceptions cannot be disabled and are always active. For PMP exceptions when enabled, every instruction or data requests is filtered by the PMP which can possibly generated LOAD, STORE or FETCH exceptions.

2.2.15 Handling

RI5CY supports SW-assisted nested interrupt/exception handling. Exceptions inside interrupt/exception handlers cause another exception, thus exceptions during the critical part of your exception handlers, i.e. before having saved the MEPC and MESTATUS registers, will cause those register to be overwritten. Interrupts during interrupt/exception handlers are disabled by default, but can be explicitly enabled if desired. Upon executing an mret instruction, the core jumps to the program counter saved in the CSR register MEPC and restores the MPIE value of the register MSTATUS to IE. When entering an interrupt/exception handler, the core sets MEPC to the current program counter and saves the current value of MIE in MPIE of the MSTATUS register.

2.2.16 Debug

RI5CY supports the RISC-V debug specification 0.13 and it implementes the execution based to reuse the existing core pipeline. RI5CY has a debug_req_i input port that is sent by the system Debug Module. Such request makes the core jumps to a specific address location where the Debug Rom is mapped. Such address location is referred as to the parameter DM_HaltAddress. RI5CY implements the debug sets of registers as dpc, dcsr, dscratch0, dscratch1.

2.3 OpenPULP Design History

Application	PULP
Technology	40nm
Manufacturer	TSMC
Type	Research Project
Package	QFN64
Dimensions	3200µm x 3200µm
Gates	1800kGE
Voltage	0.8-1.1 V
Power	153mW @ 1.1 V , 450MHz
Clock	450MHz

Table 2.5: Main Details of "Mr.Wolf"

In 2017, Integrated Systems Laboratory (IIS) of ETH Zurich and the Energy-efficient Embedded Systems (EEES) group of UNIBO have released a full ASIC implementation of the OpenPULP RTL Design that was codenamed "Mr.Wolf". TABLE 2.5 shows the main specifications of Mr.Wolf.

Some features:

- One cluster with eight 32bit RISC-V Cores (RI5CY) supporting ICMF extensions.
- Two IEEE-754 compliant FPUs each shared by four cores.
- A minimal area RISC-V (zero-ri5cy) fabric controller in the SoC module that can be used for basic control operations.

- Integrated LDO to generate all internal voltages.
- 64 Kbyte TCDM + 512 Kbyte L2 Memory

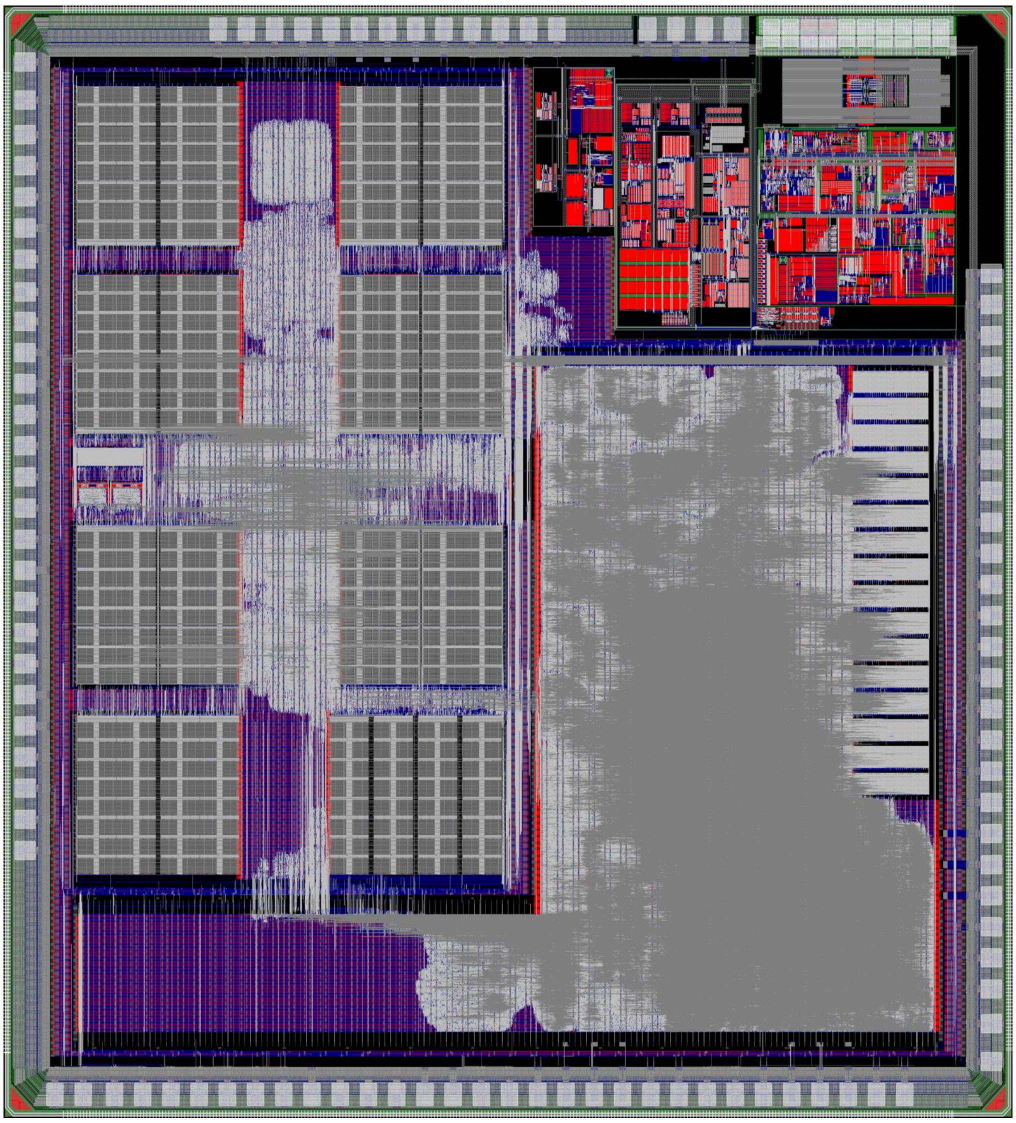


Figure 2.7: Mr.Wolf ASIC Implementation from PULP.

2.4 Proposed Design

Work on this thesis project started targeting a complete RTL-to-GDSII flow of the OpenPULP Core in three different synthesis modes. The first problem is to check the synthesizability of the opensource RTL code of the core by running a quick synthesis to compile RTL files and debug the issues that may prevent synthesis tool to synthesize the design. The second issue was to use custom clock gating cells from the technology library provided rather than structural gating logic, to reduce potential power and area also. This was achieved by manually instantiating the custom clock gating cells from the PDK inside the clock gating module in the RTL. Synopsys educational kit (SAED 32/28nm) CMOS technology is used through RTL-to-GDSII flow.

The RTL modifications can be summarized as follows:

- Using custom clock gating cells from the provided PDK rather than structural gating logic instantiated in the RTL.
- Removing floating-point unit from the RTL, by setting FPU parameter in the top-level file "riscv_core.sv" to '0', since the core is generally parametrized to isolate floating-point unit if needed. This modification was done to reduce performance parameters such as area and power of the core by isolating optional units such as Floating-point unit.

Chapter 3

Core Implementation Flow

3.1 Logic synthesis

3.1.1 Flat Flow

To synthesis the design, the Design Compiler tool from synopsis was used. The Design Compiler is considered the best synthesis tool in the market in this time. In our style, logic synthesis flow consists of choosing constraints, removing levels of hierarchy, synthesis the design, and pre-Layout verification.

Choosing constraints

we choose to design on the worst case process variation which is slow-slow at 0.7 voltage at temperature 125 degree of Celsius. During synthesis we found that the longest path delay is in range of 14 nano-seconds to 15 nano-seconds and with our knowledge of what comes next in layout, we decide to implement our design at 20 nano-seconds clock (50 MHz) to leave a suitable margin when clock network delay and interconnect delays are added. we define the clock latency with 25% of clock period which is an estimation value of clock network delay. we define clock uncertainty with 0.95 nano-seconds which is the uncertainty of the clock edge to encounter for any clock disturbance from the clock source like clock jitter and clock skew. we define the IO delay with 25% of clock period and the output load capacitance to be 20 nano-Farad. we set input transition with 0.1 nano-seconds which specifies a fixed transition time for inputs or inout ports. and also, we define fanout to be 2.5 to ensure that the sum of the fanout load attributes for input pins on nets driven by the specified ports or all nets in the specified design is less than this value. and finally, setting the clock gating style which used in our design to be latch with and gate which will be an integrated circuit found in technology file which compiler will use it in compiling phase as shown in figure 3.1.

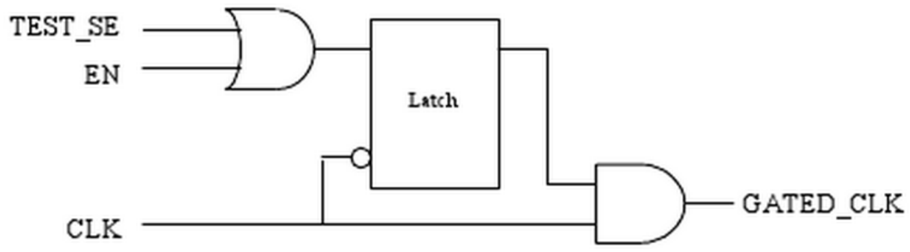


Figure 3.1: clock gating style (Flat Flow)

Removing levels of hierarchy

In this phase we face a challenge this challenge is our design consists of large number of gates that when flatten these gates and removing all levels of hierarchy the tool can not optimize results as the memory available to this big module is not enough to use so we reduce levels of hierarchy as much as we can to allow tool to make optimization.
Now, we will discuss how we remove levels of hierarchy.
the default levels of hierarchy of the design is shown as in figure 3.2.

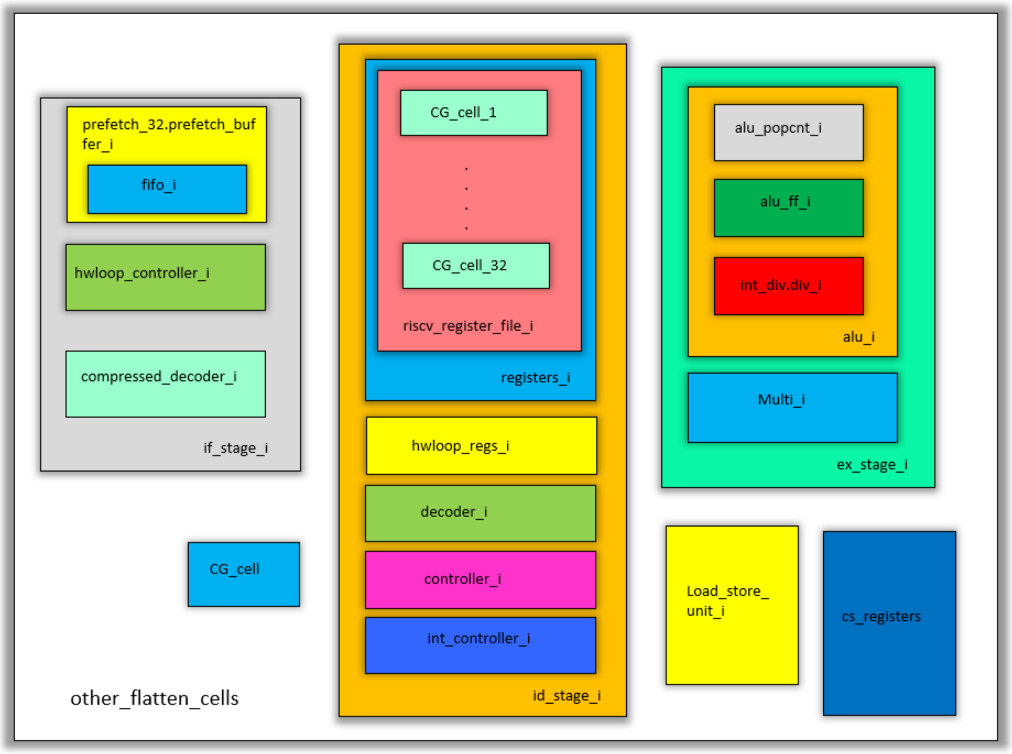


Figure 3.2: default levels of hierarchy (Flat Flow)

by default, ultra high effort compilation removes levels of hierarchy as much as it can so we allow tool to remove levels of hierarchy which help it to optimize results, figure 3.3 shows us how the tool remove part of hierarchy.

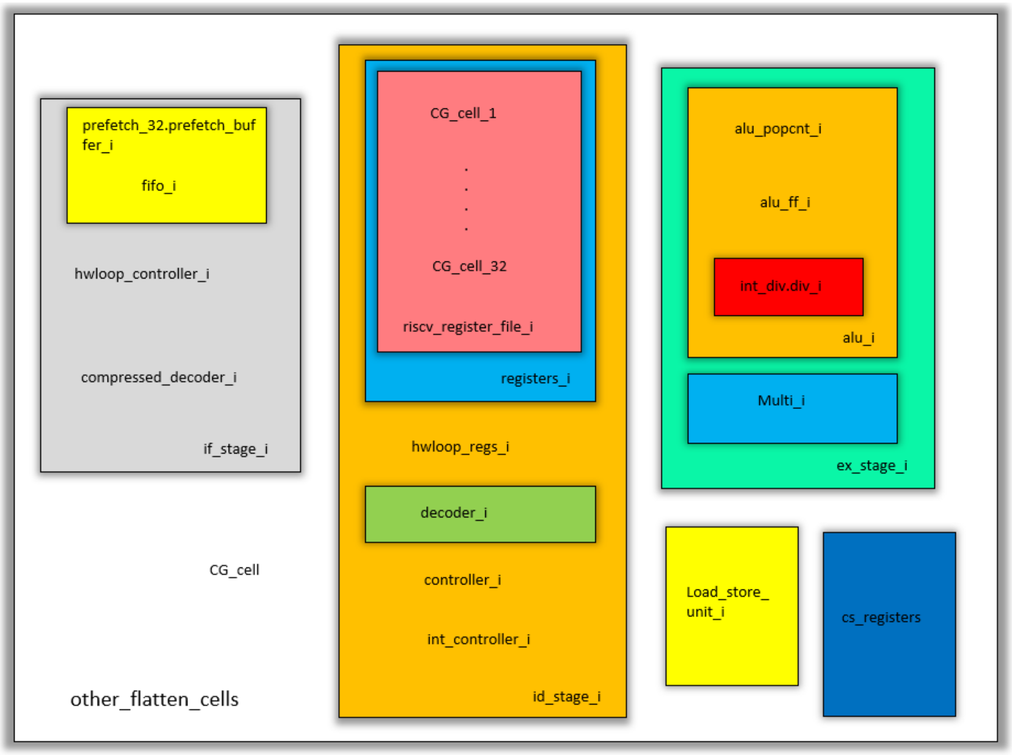


Figure 3.3: after ultra high effort compilation (Flat Flow)

And then we remove levels of part hierarchy by our hand to optimize results more than this case, the removing of levels of hierarchy we done depend on more iterations we done. figure 3.4 shows the final hierarchy we do which gives us good results in timing, area, and power.

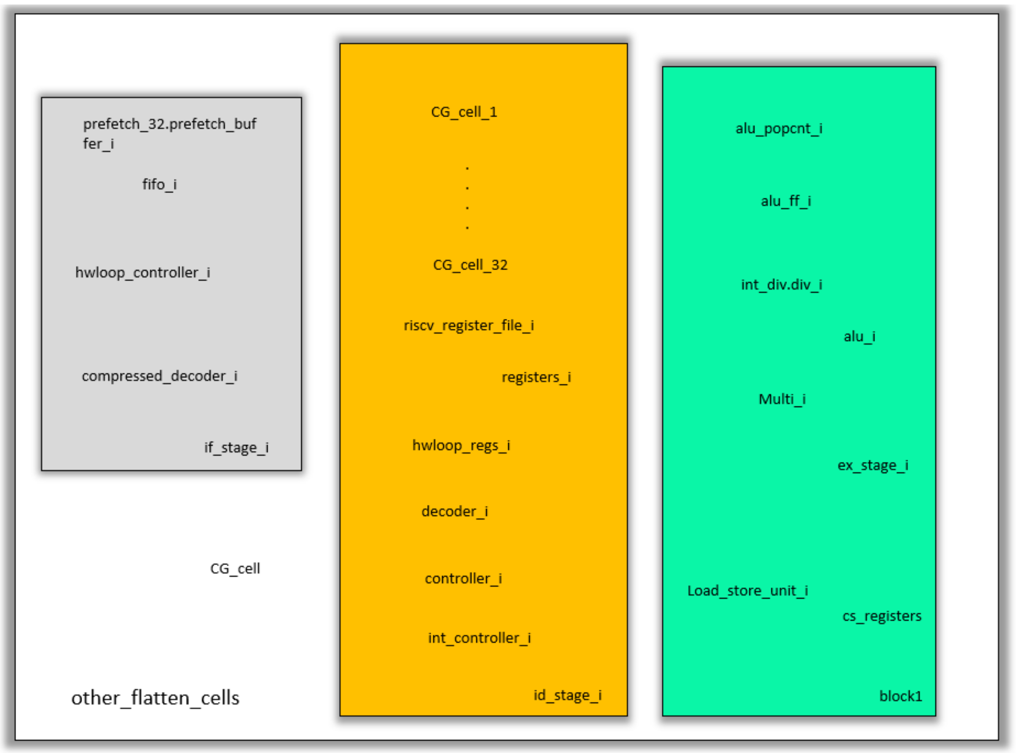


Figure 3.4: final levels of hierarchy (Flat Flow)

As shown in the figure 3.4 only there are 3 levels of hierarchy which no sub-levels of hierarchy found inside each one and other cells are flatten in top module the number of gates in each module are:

- Top module: 1096
- If_stage_i: 2139
- Id_stage_i: 9019
- block1: 10766

Synthesis the design

After choosing the library file and applying all constraints to the design and removing (flatten) levels of hierarchy as much as we can to get good optimization results in timing, area, and power we then compile the design using compile the design with ultra high effort compilation with activation of gate clock option to improve dynamic power of the design. during synthesis process we face a many setup time violation and to solve these violations we use group pathing method which group these violated paths and give it a high priority during optimization and also setting maximum delay value to these paths helps to get good results.

Finally, the gate level netlist is created with no setup and hold time violations at worst case PVT variation.

pre-Layout verification

After synthesis of the design in worst-case operating conditions with no violations, we test the design for best-case operating conditions to verify that it works well or not. Actually, best case operating condition needs a large number of buffers to add to design and as we know after Layout phase the timing results change because of clock tree delay is replaced by an estimated clock latency and parasitics delay of wires, So we decide to minimize the weight of paths slack's for example if the worst hold slack is -3 we minimize it to -2 and all other paths are affected as worst one. This technique helps us to provide the available location of buffers in the Layout phase and to get fewer violations during the post-Layout verification phase for multi-corner multi-mode analysis.

3.1.2 Topographical Flow

First-Pass Synthesis

First-pass synthesis is performed by three main steps:

1. Loading RTL Files
2. Design Constraining
3. Mapping and Optimization

we start by reading RTL files then we define our timing constraints. We define our target frequency based on a few trial runs of increasing clock frequency and tightening design constraints, finally we settle with 50MHz clock frequency that satisfies our timing requirements.

In design constraining phase, we first define uncertainty to the clock period of 5% of clock period to account for some factors such as clock skew and clock jitter that may affect our timing results afterwards, then define clock latency with 30% of the clock period. Clock latency is by definition the propagation delay measured from the clock definition point to the clock pin of sequential cells. We also define the environment constraints for the boundaries of the core such as IO delays with 25% of clock period, capacitive load of 20nF on the output ports and input transition of 0.1ns on the input ports. These external constraints are mandatory so that accurate estimation of the paths delay can be done. We then divide all timing paths in our design to three path groups in addition to the default path group which is the master clock path group, "REGIN" includes all timing paths from input ports to D pins of registers , "REGOUT" includes all timing paths from clock pins of registers to the output ports and "FEEDTHROUGH" which includes all timing paths from input ports to output ports. Path groups are groups of timing endpoints that have a common property. This is a technique used by the tool in order to categorize timing endpoints into several groups so that STA engine can work on the timing requirements of each group separately.

By default , our design has manually instantiated clock gating cells provided by the PDK. In order to run a clock gating synthesis flow , Synopsys Design Compiler requires from user to determine the style of clock gating cells to be chosen from the standard library, Therefore, we define the clock gating style to be used during mapping and optimization phase by the FIGURE shown below.

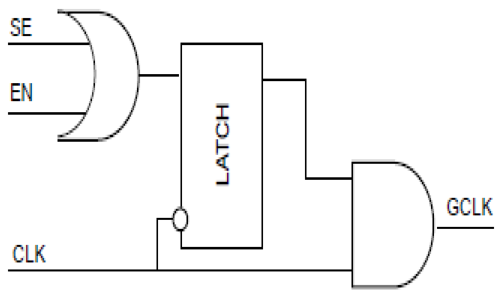


Figure 3.5: Clock gating style used during synthesis (Topographical Flow)

Then we proceed to mapping and optimization step where ultra high effort compilation is performed with some additional techniques such as registers retiming to improve QoR of the core and clock gating to optimize total power.

initial floorplan

intial floorplan including mainly four steps:

1. defining physical constraints.
2. Virtual Placement
3. Power Network Synthesis
4. Optimization for timing.

Initial floorplan is performed using Synopsys IC compiler, we start by reading netlist from the first pass synthesis. based on a few trials we start to define core dimensions (227micron x 454micron) which is the minmum dimensions that satisfies physical constraints and core aspect ratio of 2 that achieves minimum routing congestion.

Our PDK has 9 routable layer, we specifiy minimum routing layer to be Metal2 and maximum routing layer to be Metal8 while Metal1 is used for power rails and internal connections of the standard cells and Metal9 is used for creating the power rings. Virtual placement is then performed to allocate cell placement locations in the core area , then we proceed to power network synthesis in which we create vertical/horizontal power straps and connect them to the standard cells. Power straps are made with Metal4 and Metal5 layers. Maximum VDD IR Drop is 53 mV and maximum VSS IR Drop is 52 mV that don't exceed ten percent of voltage source which is **0.7 volt**. Then we proceed with connecting power network to all cells and commit it.

finally, we proceed to floorplanning timing optimization step and check if any DRC violations before moving to next step. Inital floorplan passed without any timing/DRC violations with overall routing congestion of 2.38%. DEF file that contains all physical information needed for topographical synthesis iteration is now extracted to proceed with second iteration of synthesis.

Second-Pass Synthesis

In second-pass synthesis , we acutally perform topographical-mode synthesis with back annotated cell placement locations and parasitic interconnections that are extracted from initial floorplan. We read RTL files again, SDC file and extracted physical constraints from initial floorplan. Wire load models are not used in this pass because the tool will depend on accurately extracted physical constraints now for static timing analysis rather than timing estimations as WLMs. Ultra high effort compilation is performed again with clock gating techniques and timing high effort scripts that enhances QoR in addition to physical guidance flow that is made especially for Design Compiler in topographical mode. Physical guidance flow enables Design Compiler Graphical to perform enhanced placement that is consistent with the IC Compiler placement commands functionality and enhanced post-placement delay optimization in order to provide a better optimized starting point for physical implementation. As a result, placement is aligned between Design Compiler and IC Compiler, improving runtime, quality of results (QoR), and correlation.

We managed to reduce power, area and get more improvment in timing as shown in the next chapter, Topographical DC compiler taking physical constraints in his consideration made a remarkable utilization in area , power , runtime and QoR, rather than first pass synthesis. We then proceed to the actual place and route flow afterwards in the next sections.

3.1.3 Hierarchical Flow

General logic synthesis flow is preformed as shown in 3.6.

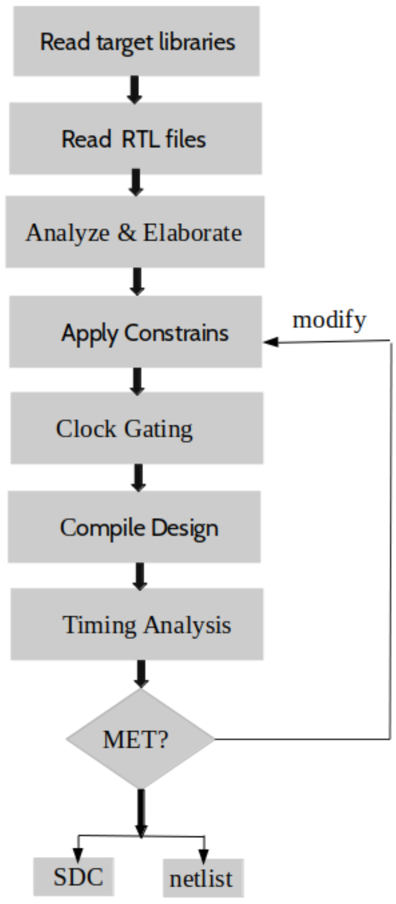


Figure 3.6: Generic logic synthesis (Hierarchical flow)

In the hierarchical flows there are many techniques that may be used such as bottom-up or top down flow , we used both at different stages . The synthesis step is preformed by Synopsys Design Compiler (DC) the most powerful and the more commonly used tool in market.

First-Pass Synthesis

First pass synthesis is done by the following main steps:

1. Reading the RTL (HDL) Files.

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

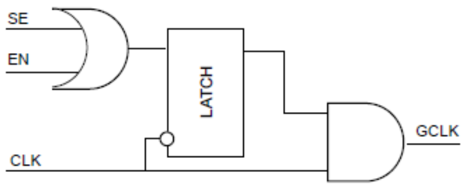
   

   Figure 3.7: Clock gating style that is used by Hierarchical flow

2. Apply Logical Constrains

   constraining design is partitioned into two main sections:

   (a) design rule constraints.
   I/O delays ,clock period ,clock uncertainty ,capactive load , max transition and max fan-out ..,etc

   (b) optimization constrains
   grouping each related paths to guide the tool to do the best optimization and focus on each path group separately , such as regin (in-to-reg) , regout (reg-to-output), inout (inputs-to-outputs) and the default clock path group and leakage power , dynamic power , max area , wire-load modeling and operating condition ...,etc

   our constraints values were as follows :

   - clock period: 20 ns

- clock latency : 7.5 ns (37.5 % of clock period)
- clock uncertainty :0.75 ns
- input transition : 0.1 ns
- output delay : 5 ns (25% of clock period)
- input delay : 6 ns (30 % of clock period)
- output capactive load : 0.02 fF
- operating condition: the worst case (ss0p7v125c)

(c) Compiling and Optimization.

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

First pass netlist in done using top-down approach which is constraining the top module only and as DC by default propagates the top module constrains to the lower hierarchical modules and compiling the whole design under the worst operating condition of a PDK (student version PDK) saed32rvt_ss0p7v125c and results the whole design without any setup timing,hold timing or DRC violations.

Design Planning

Design planning stage consists the following major steps :

1. Design import, floorplanning
2. Plan Group creation
3. Virtual Flat Placement
4. Hierarchical Placement
5. Power Network Synthesis and Analysis
6. Pin Assignment
7. Timing Budgeting

8. Committing the hierarchy

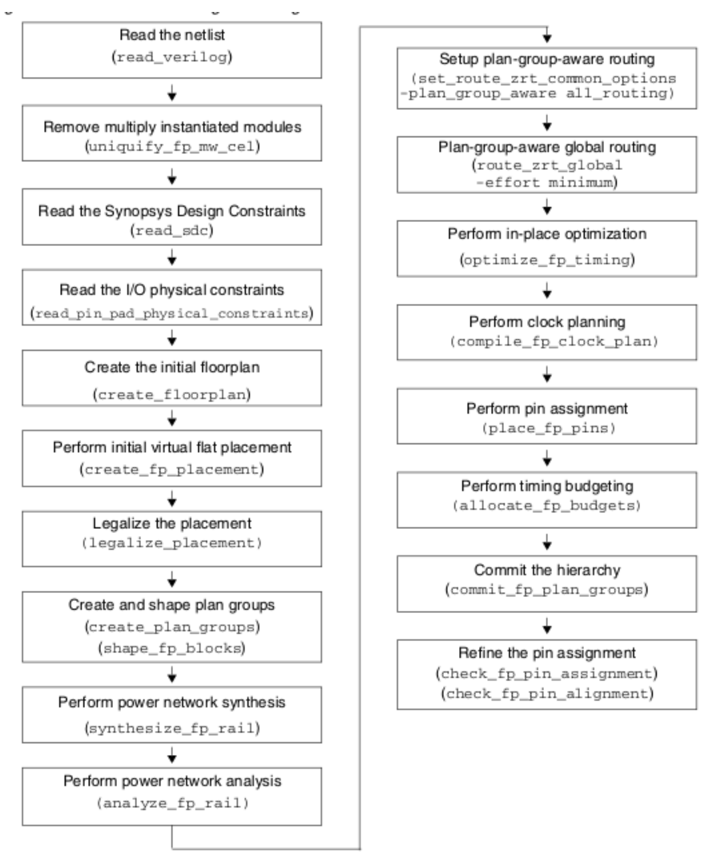

Read the netlist
(read_verilog)
Remove multiply instantiated modules
(uniquify_fp_mw_cel)
Read the Synopsys Design Constraints
(read_sdc)
Read the I/O physical constraints
(read_pin_pad_physical_constraints)
Create the initial floorplan
(create_floorplan)
Perform initial virtual flat placement
(create_fp_placement)
Legalize the placement
(legalize_placement)
Create and shape plan groups
(create_plan_groups)
(shape_fp_blocks)
Perform power network synthesis
(synthesize_fp_rail)
Perform power network analysis
(analyze_fp_rail)
Setup plan-group-aware routing
(set_route_zrt_common_options
-plan_group_aware all_routing)
Plan-group-aware global routing
(route_zrt_global
-effort minimum)
Perform in-place optimization
(optimize_fp_timing)
Perform clock planning
(compile_fp_clock_plan)
Perform pin assignment
(place_fp_pins)
Perform timing budgeting
(allocate_fp_budgets)
Commit the hierarchy
(commit_fp_plan_groups)
Refine the pin assignment
(check_fp_pin_assignment)
(check_fp_pin_alignment)

Figure 3.8: Design Planning steps followed in Hierarchical Flow

- Design Import and Floorplanning :

  at this step we import the full netlist which is generated from the top-down flow and define the physical constrains from the core area by specifying the core aspect ratio and the utilization factor which is define the ration between the estimated interconnections and routing area and the physical cells area, the metal layers used during design planning are from Metal1 to Metal9. Metal layers are divided such as Metal7 and Metal8 are used for power straps, Metal9 is used for core power rings, and Metal3 up to Metal5 are used for clock tree synthesis. While Metal1, Metal2 and Metal6 are used mainly for signals routing.

- Plan Group creation :

  in this step we define the physical hierarchy of the design and the physical constrains of each the decision of this hierarchy in supported by the tool that defines it based on the area and routing considerations

- Virtual Flat Placement :

  Virtual flat placement is the simultaneous placement of standard cells and macros for the whole chip; it is a fast initial flat placement performed for design planning purposes. For designs with macros, plan groups, or voltage areas that have not already been placed, virtual flat placement can help you decide on the locations, sizes, and shapes of the top-level physical blocks. This placement is "virtual" because it temporarily considers the design to be entirely flat, without hierarchy. After you decide on the shapes and locations of the physical blocks, you restore the design hierarchy and proceed with the block-by-block physical design flow.

- Hierarchical Placement :

  this step is also e simultaneous placement of standard cells and macros but for each plan group after shaping the plan groups inside the chip die area

- Power Network Synthesis and Analysis

  the power planning steps is shown in figure 3.9.

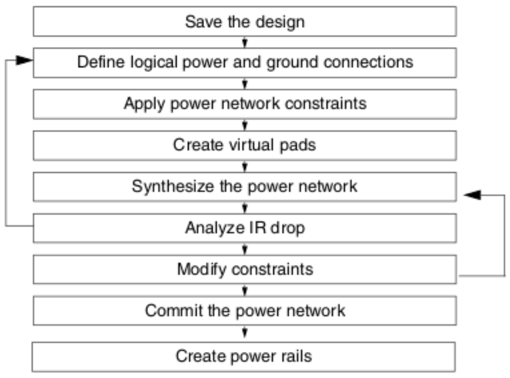


Figure 3.9: Power planning steps in Hierarchical flow

- Pin Assignment
  this step is mainly for use pin assignment to assign pins at the top level of the design, or to assign pins on soft macros and plan groups.For top-level pin assignment, the tool considers the top-level connections to plangroups, macros, and pad locations when it determines where to assign the pins. For block-level pin assignment, the tool considers the cell placement inside the soft macro or plan group when it assigns pin locations. The tool minimizes the wire lengths from the pins to the internal connections within the soft macro or plan group. Use block-level pin assignment in a bottom-up design flow.

- Timing Budgeting
  During the design planning stage, timing budgeting is an important step in achieving timing closure in a physically hierarchical design. The timing budgeting determines the corresponding timing boundary constraints for each top-level soft macro or plan group (block) in a design. If the timing boundary constraints for each block are met when they are implemented, the top-level timing constraints are satisfied.
  Timing budgeting distributes the timing constraints from the top level

to the block level, creating a new constraint set for each of the top-level blocks in the design. Budgeting also allocates slack between blocks for timing paths crossing the block boundaries. Timing budgeting propagates the timing constraints downward one hierarchical level at a time. Timing budgeting also propagates timing exceptions to block-level constraint sets.

- Committing the hierarchy after finalizing the floorplan by converting plan groups into soft macros. Committing the hierarchy creates a new level of physical hierarchy in the virtual flat design by creating CEL views for selected plan groups.
  figure 3.10 shows the plan groups after committing and turns up to be soft macros

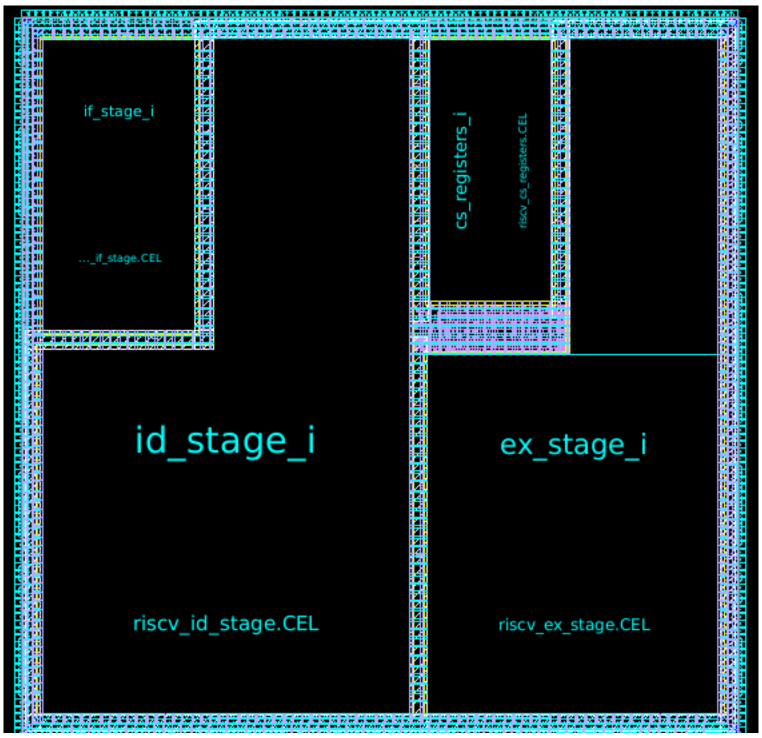


Figure 3.10: Hierarchy Commitment in Hierarchical Flow

Second-Pass Synthesis

Second pass synthesis is done using the bottom-up approach this step is required since the design is optimized in the first path without any correlation between which we will be done in the Design planning and the logic synthesis , so it is required to avoid the timing violations which is appeared in the block level implementation. This step begins with the budgeted constrains for each block and the DEF or the floorplan file for each block which contains all the physical constraints which is need to be consider in the logic synthesis stage. This second pass logic synthesis is preformed by also Synopsys Design Compiler but in the topographical mode with also ultra high effort compilations with clock gating techniques for power reduction and optimizations and also with spg option to consider also the congestion in each blocks as well as in the top module.

note: All the Design Planning steps and the second pass (bottom-up) logic synthesis are preformed by the Synopsys Reference Methodology scripts .

3.2 Place And Route

3.2.1 Flat Flow

After successfully synthesizing the core with the target clock and constraints, we then proceed to place and route the entire core inside the chip area.

Floorplanning

Floor planning is first step to begin with it in the place and route flow where it is a common stage in our three flows. After receiving the clean netlist from synthesis tool, we then start to plan the floor area of the physical standard cells by inserting them inside the boundaries of die area. Boundaries of the core area are defined by the aspect ratio which is the length over the width of die. We chose the aspect to ratio to be 2, so that we can be able to reduce the congestion issue in the vertical-based axis.

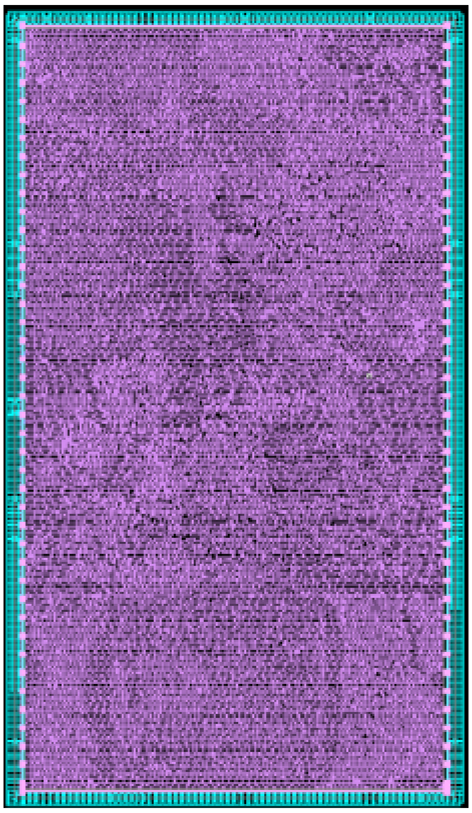


Figure 3.11: Floorplan of Riscv Core (Flat Flow)

Figure 4.7 shows the raw die with 70% in utilization of the core area and aspect ratio of 2. In floorplanning, we have set the virtual placement strategy to be virtual in-place optimization, or virtual-IPO, in order to reduce the congestion when the timing is not critical as we have a good positive slack margin so that we may not need to perform a timing driven placement. The last step in this section is to apply the virtual flat placement which is the core of our design flow with timing-driven and no hierarchy gravity so that the cells can move freely in any location in the core without any constraints.

After we had virtually placed the design in the core area and set the placing strategy, we become ready for establishing the grid of power which is going to feed the standard cells with the needed supply of voltage.

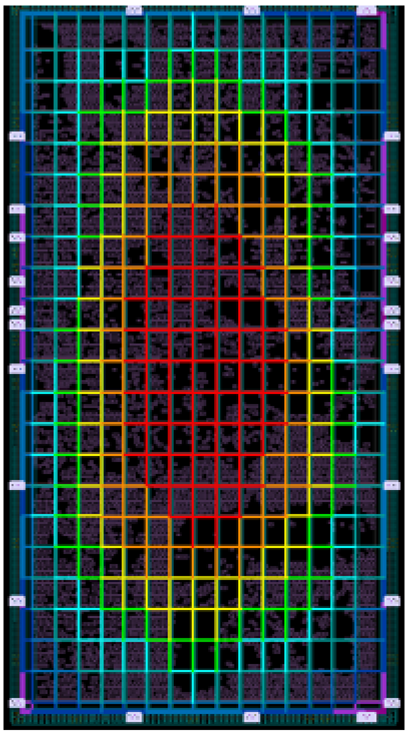


Figure 3.12: PNA voltage drop heat map (Flat Flow)

In order to synthesize the power network, we have to follow some steps. First of all, we have defined the power and ground nets and ports in each standard cell so that the tool can be able to connect later to the grid. After that, we have constrained the power-ground rails and core rings so that they do not exceed 30 rails either in vertical or horizontal directions. We then chose metal 5 and metal 6 layers to build our power grid on them and the maximum voltage drop not to exceed 10% of power supply. The power supply in the worst-case study is 0.7 volt so that the maximum drop must be 0.07 volt. In order to simulate the power flow in the network, the tool needs virtual power pads so that it can calculate the available current flown from them to be able to strictly construct the power grid in virtual ports basis. Figure 3.12 illustrates the voltage drop map after synthesizing the power grid and virtual pads. The red color indicates the worst voltage drop achieved in the network and the outside colors is indicating the drop which is gradually decreasing in all directions outside the interior red-colored circle.

Placement

After performing a virtual flat placement and synthesizing the power network on the chip, we applied a standard coarse placement step. In this step, we performed a congestion-driven placement in order to improve the routing congestion in the design, and the result was an overflow of a few GRC cells which was indicated by approximately 0% of congestion in both vertical and horizontal directions. The timing requirements were also satisfied in this stage, and the design was not optimized in area due to conservation of inserted buffers to clean hold time in some corner case. Therefore, the design was clean in timing and congestion, but the area would be optimized later in the final clean-up stage.

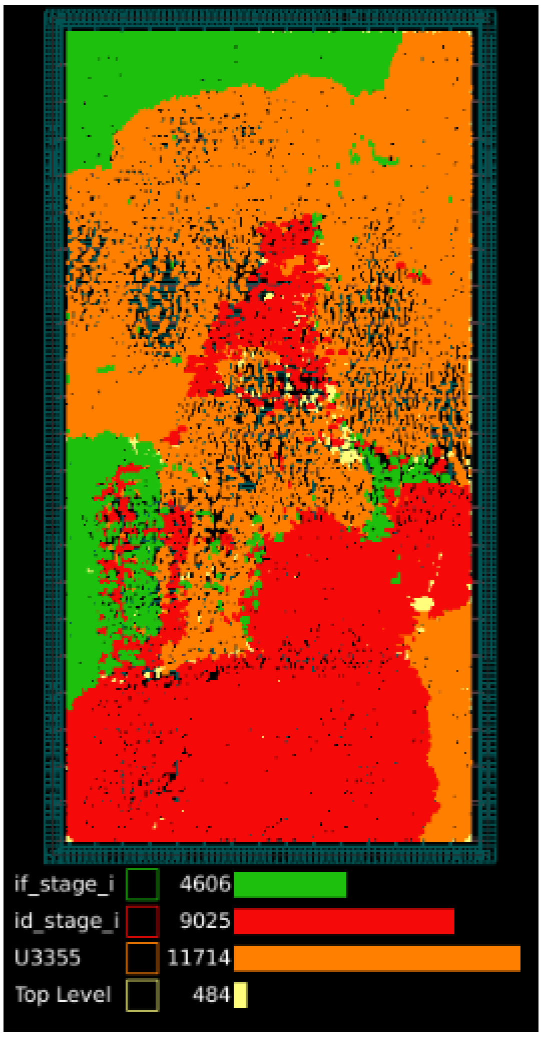


Figure 3.13: Hierarchy map of the core (Flat Flow)

Figure 3.13 illustrates the post-placement hierarchy of the design. There are four colors, each of them indicates a high-level block in the hierarchy. The orange color dominates in the design area since it indicates the most of the flattened blocks in the design.

CTS

After placing the design with no congestion and timing violations, the next step is to build our clock tree. In clock tree synthesis stage, we are going to discuss the clock tree construction process, and its steps. The design contains a single clock source distributed over the whole chip. This stage comprises of three main sub-stages each of them has its own characteristics, but before establishing these three steps, we have to set some configurations and rules in order to build our tree without any problems. We going to discuss these rules and constraints in detail.

First of all, we have set the fanout of the clock nets not to exceed the value of 20 in order to decrease the clock latency and skew as much as possible. The second configuration is the routing rules. We chose to establish our clock network in metal 8 and metal 9 layers with default routing rules for clock sinks, but non-default routing rules for clock nets. We have to widen the wires so as to achieve good latency and decrease the electromigration and crosstalk effects.

In order to help the tool to build the network with some constraints, we gave it some information about timing requirements. The required minimum skew was to be 0.1 nanoseconds and maximum clock transition is 0.2 nanoseconds, and the target clock latency was to be 1 nanosecond.

After all, we have become ready for initializing the first step. This first step is considered as only building the clock tree itself where inserting the buffers with different sizes and different locations so that we can achieve balanced network with minimum skew and latency. This step was done without routing the clock nets so that we can have some liberty to have best results without the overhead of routing the clock nets as routing can take significant time, and we didn't need it in this step.

In this phase, we have encountered some violations like minimum clock pulse violation. This problem occurs due to much buffering in the same clock path since the clock pulse decreases in width when passing through a buffer. One solution to this problem is to reconfigure the whole clock tree many times so that the clock tree has the required minimum pulse width. If this solution fails due to having many buffers which cannot be manageable in this way. We can resort to using inverter-based clock tree. The inverter can decrease the clock pulse width with a smaller amount than buffer. As a result, the tool can handle this violation in a better manner.

The next step is to perform a post-CTS optimization. In most cases, synthesizing the clock tree can cause timing violations in setup and hold manners in addition to design rule violations such as maximum capacitance and maximum transition violations. Therefore, we have to perform an additional synthesis step after successfully build the clock tree. By enable hold time violations fixing in the tool, we can fix them side-by-side with setup violations. We at last have finally clean timing and DRC results.

The last step is routing the clock nets and clock buffers together. After constructing the clock tree and performing post-CTS optimizations, we finally can route the clock wires with relief. Routing the clock nets can affect the setup and timing requirements with small violations due to the finite parasitic resistance and capacitance in the wires. Therefore, we have to do another synthesis step with a separate synthesis engine that is dedicated for timing and DRC violations in any stage in the PNR process.

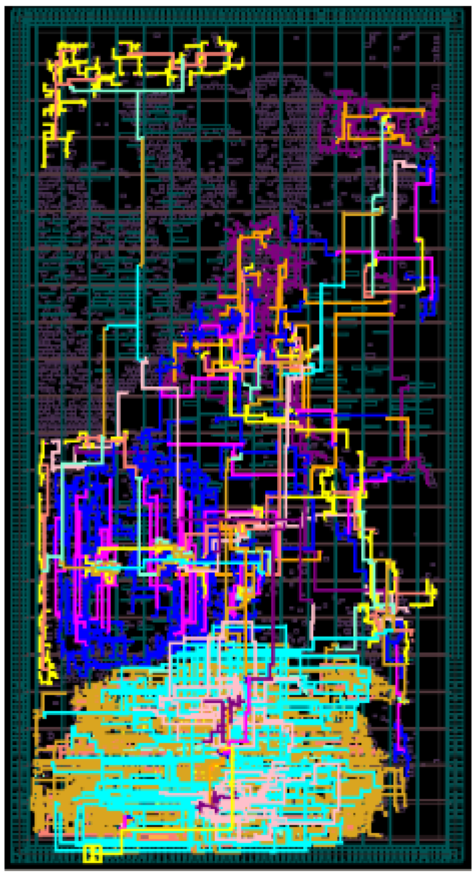


Figure 3.14: Clock Tree Network (Flat Flow)

Figure 3.14 illustrates the clock tree network distributed over the whole chip. As shown in the figure, the density of clock nets in not balanced in the chip due to non-uniform distribution of the sequential devices. There are 20 levels in this tree. Each of them is illustrated by a certain color.

Routing

After establishing the core clock network without any timing and DRC violations, the physical design is now ready for being routed. But before going through the routing stage, we have to check for congestion so that the tool can route with ease. The resultant congestion in post-CTS stage is approximately 0% in both vertical and horizontal directions.

After that, we can enable signal integrity options in order to guarantee a robust routing so that all signals find their way to their sinks without any signal drops. These drops can sBefore performing actual routing, we need to define some rules and configure some options in order to achieve clean routing without any violations. First of all, we defined the delay calculation options so as to calculate the delay of the actual clock and signal nets. There are two methods for the tool to calculate the delay: Elmore and Arnoldi. Arnoldi method is considered more accurate in delay calculation but needs a more powerful computer to perform the calculation without any problem since tool got crashed when using this method. Therefore, we had to use the less accurate one which is Elmore. The second option is to specify the routing layers. We chose to route at maximum of three layers which are metal-2, metal-3 and metal-4 layers. Another routing options that we selected is to perform timing-driven global and detailed routing so that the tool can take timing into account when routing.ignificantly damage the entire chip. In our case, we have chosen to enable crosstalk prevention and static noise reduction.

Consequently, we can forward to our routing steps directly. The routing process comprises of three main phases: global routing, track assignment and detailed routing. Each of them is discussed in detail in appendix chapter in the tail of our document. Another step that comes in final phase is search and repair. This step is dedicated for fixing physical DRC violations when routing is complete. By analyzing the congestion one more time before routing, we can notice that approximately no congestion found in the design. Therefore, we can skip the separate global routing stage. IC compiler tool offers a routing command that can perform the three routing phases by only one-hit command.

First of all, we have run an initial routing phase instead of full routing which is time-consuming and can be useless if the initial routing is done with no problems. After performing this step, we found many DRC violations on nets and timing requirements was not met. Therefore, we had to skip this step and begin a full routing phase. This phase covers all routing issues but has long runtime. After successfully performing the first full routing phase, we found a few timing and physical DRC violations. The tool also offers an incremental routing phase which can fix timing and DRC violations without performing the whole routing of the design from the beginning. We can notice that the tool can fix the setup violations in this stage by adding more length to the wires and adding more vias in order to increase the delay of the timing path. On the other hand, the hold time fixing can be done by shortening the wires and reducing the number of vias as much as possible.

After all, we have got a clean timing and physical DRC results which will be exposed in detail in results chapter.

Chip Finishing

The final step in the place and route is to do the final touches on the chip. The main reason behind this step is the manufacturing process which needs some steps to guarantee error-free chip during manufacturing process. This stage essentially comprises of four elements as follows:

- Wire spreading and widening
- Standard cell filling
- Redundant via insertion
- Metal filling

The concepts and reasons for performing these steps are discussed in detail in the appendix chapter. So, we can only discuss the results of the standard chip finishing phase.

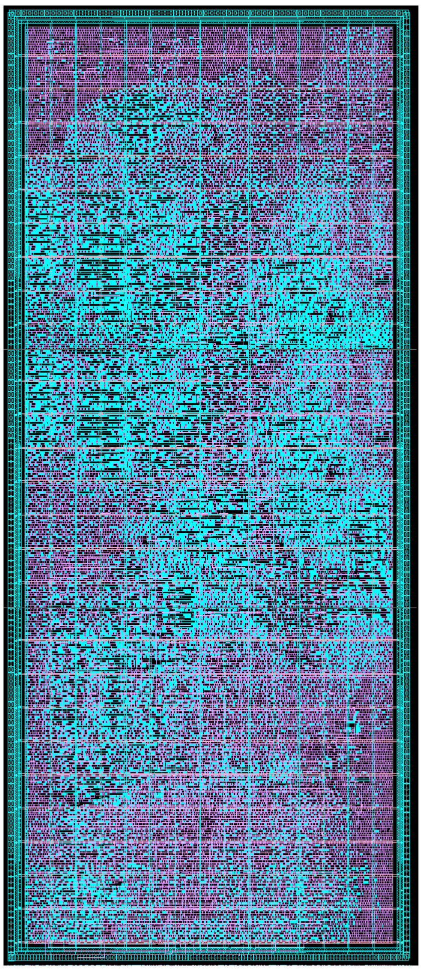


Figure 3.15: Final die shape after chip finishing (Flat Flow)

As shown in Figure 3.15, the die has filled with dummy standard cells and dummy metal wires in order to fill the gaps among the actual cells and wires. The dummy cells appear in the light blue color and actual cells appears in dark violet color.

After performing this stage, we have to perform a final check for the whole design against timing and DRC violations. If any of them appears, we can perform a simple synthesis to recover these errors, then the chip is now ready for being manufactured after performing a post-layout sign-off in an STA tool to accurately check the timing of the entire chip.

3.2.2 Topographical Flow

In this section we describe in more details the attempt to implement the Open-PULP Core based on the flow described in Appendix A, Synopsys IC Compiler tool is used in the implementation.

Floorplanning

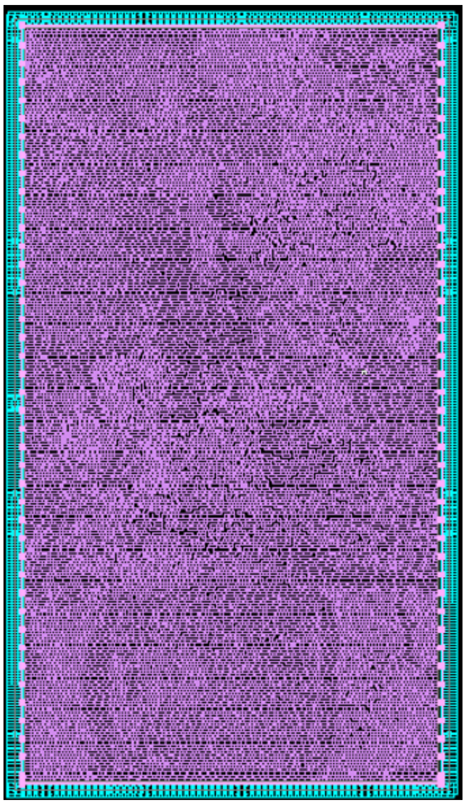


Figure 3.16: Floorplan of the OpenPULP Core (Topographical Flow)

FIGURE 3.16 shows floorplan of the chip. In floorplanning stage, we start by defining Die/Core area, aspect ratio, core utilization and IO2Core margins. Aspect ratio is the ratio between height and width (Height/Width), core utilization means the percentage of total cell area relative to the total area of the chip.Core Height is 238.488micron, and core width is 476.52micron. Total area increased relative to the initial floorplan performed previously and did not decrease, since we added ECO changes from Pre-layout STA such as buffers insertion and cell sizing, which result in an increase in the cell area. Core utilization is defined by 70% , core aspect ratio is "2" and IO2Core margins are 5micron in all sides of the chip. Then we define the metal layers that will be used for routing stage, this is important at floorplanning stage so that PnR tool can perform global routing to estimate routing regions and potential congestion properly. The technology file provided with the PDK contains 9 routable layers, minimum routing layer is chosen to be Metal2 and maximum routing layer is Metal8.

These layers will be used as the following:

- Metal2 and Metal3 for signals routing.
- Metal4 and Metal5 for horizontal/vertical power straps.
- Metal6-Metal8 for clock signal routing as higher metal layers have less resistance than lower layers which means lower interconnections propagation delay.

Virtual placement is then performed to allocate cell placement locations inside the chip core. After that we proceed to power network synthesis stage in which we construct power straps and connect them to standard cells at the bottom of the chip.

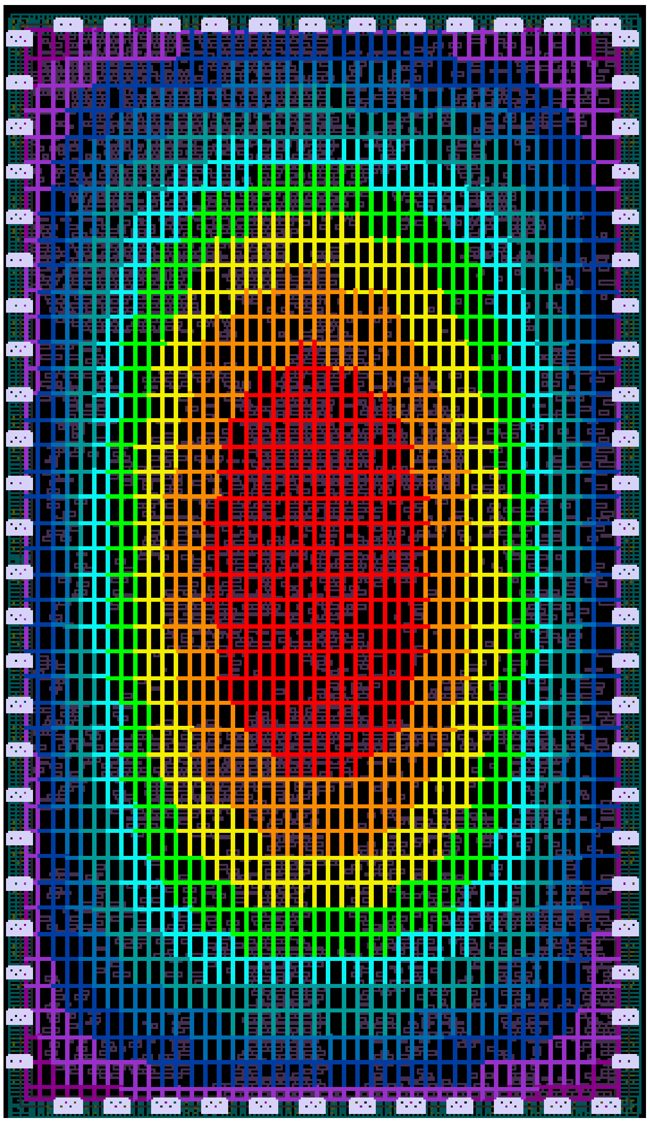


Figure 3.17: PNA Voltage Drop Map (Topographical Flow)

FIGURE 3.17 shows the voltage drop severity across the whole chip. PNA Voltage drop is constrained by the voltage supply value, here voltage supply of 0.7 V is used which constrains the maximum IR Drop value by 70 mV (10% of the voltage supply). We achieved maximum VDD IR Drop of 50 mV and maximum VSS IR Drop of 49.1 mV. These values are the values inside the red region of FIGURE 3.17 which are acceptable as long as they are below 70 mV.

Global routing is then performed, routing congestion value is found to be 2.28% mostly concentrated at horizontal routing resources by 2.2%. While vertical routing congestion is 0.08%.

Placement

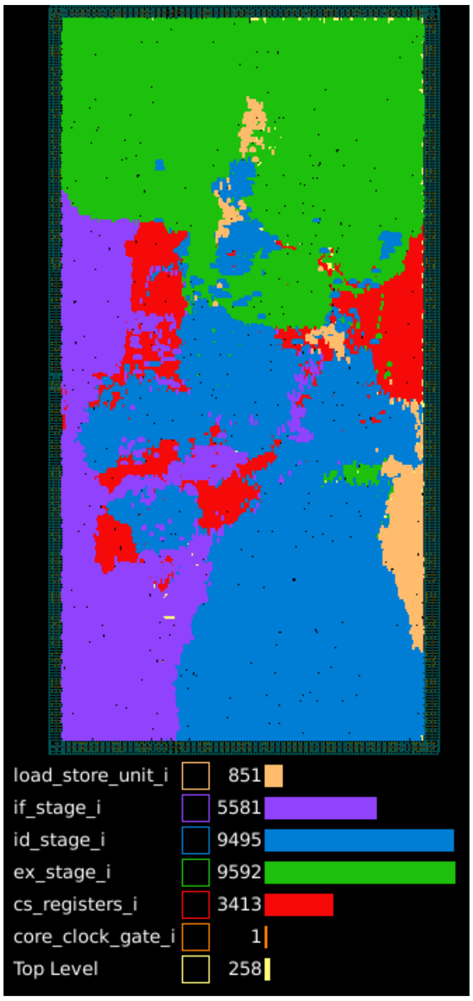


Figure 3.18: Hierarchy map of the core (Topographical Flow)

FIGURE 3.18 shows the virtual cell placement locations defined from floorplanning stage, note that cells from the same block are not tied together, this is with the help of no-hierarchy-gravity placement that results in a better QoR. During placement stage, non-default routing rules are defined for clock nets, these rules basically are double-spacings and double-widths of the clock nets. Non-default rules are often used to "harden" the clock, it means to make the clock routes less sensitive to crosstalk or electromigration (EM) effects, these rules are needed to be taken into consideration at placement stage which is before actual routing of clock nets. Here we define 4x spacings and 4x widths to ensure less sensitivity to crosstalk and electromigration effects. Additional path grouping here is made

for integrated clock gating cells (ICG) enable pins to tighten timing requirements on these pins during placement. As mentioned in previous sections, path groups are groups of timing endpoints that have a common property. This is a technique used by the tool in order to categorize timing endpoints into several groups so that STA engine can work on the timing requirements of each group separately, this can help in more QoR improvements.

CTS

In CTS, Clock network is built either with inverters or buffers. When trying to build the clock network of the core with buffers, max transition violations appeared at the clock pins of sequential cells during Pre-CTS optimizations. Alternative solution was to build it using inverter cells with various sizes provided by the PDK. Buffer based clock tree can introduce asymmetry in the rise and fall delays, and hence the high and low pulse widths of the clock signal, this asymmetry can cause max transition violations at the clock sink pins. To avoid such a problem, we used inverters rather than buffers, advantage of using an inverter based clock tree is that the high pulse width and the low pulse width would be symmetrical.

Another problem appeared after clock nets routing is max capacitance violations on some clock sink pins located in id_stage_i block. Solution for such a problem was to insert an extra stage of inverters at these sink pins to decrease heavy loading on the violating clock nets. This modification results in an increase in the global clock skew, but this is not an issue as long as global clock skew is below maximum clock uncertainty, specified at logic synthesis stage. FIGURE 4.17 shows the clock network of the core, Clock network is built across 26 levels. Each level is colored in a different color as shown.

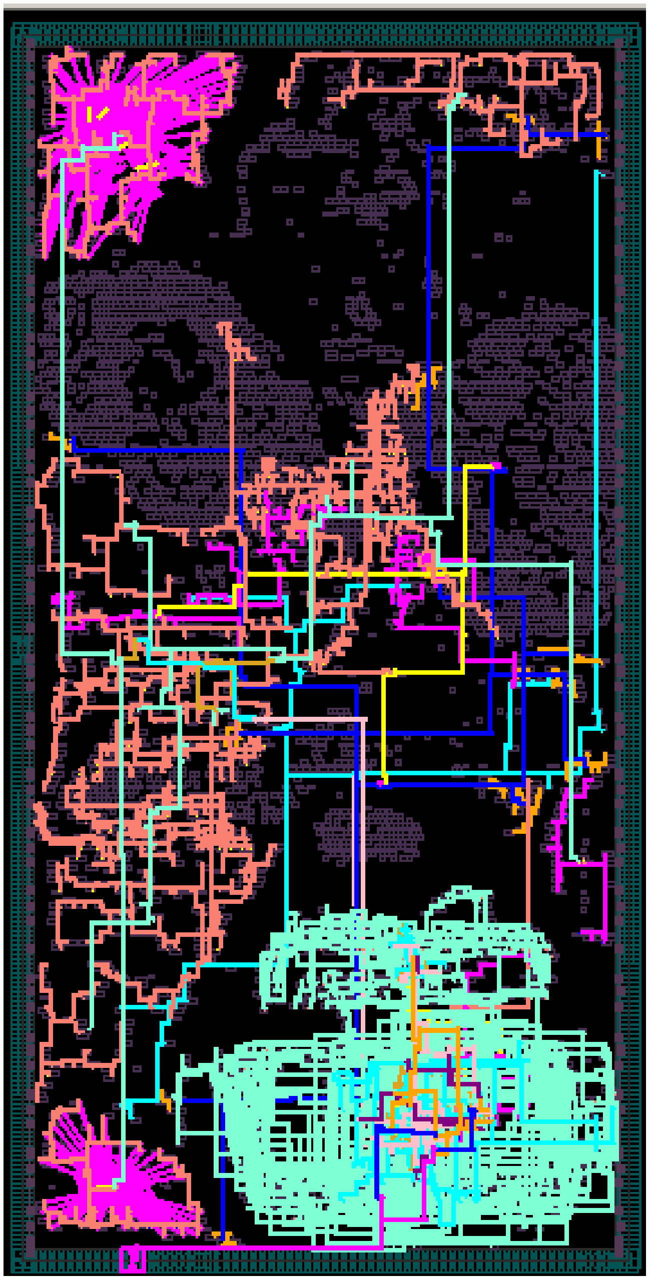


Figure 3.19: Clock Network of the core (Topographical Flow)

Routing

All signals nets are actually routed during Routing stage, clock signals are already routed in CTS stage. Routing is performed on four main steps:

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

Global routing is performed here again to assign nets to specific metal layers and global routing cells. Track assignment assigns each net to a specific track and lays down the actual metal traces and detail routing actually route the specified nets from previous steps. Search and repair is a method to solve any DRC violations that appeared from previous steps through multiple loops using progressively larger SBox sizes.

Chip Finishing

Chip finishing consists of four basic steps:

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

Metal layers spreading and widening is performed to protect minimum-width interconnecions from opens and minimum-spacing interconnections from shorts, this could happen because of any random particles that could fall on the chip and damage it during fabrication process. But spreading and widening may introduce new timing violations, to avoid such a problem we may prevent spreading and widening from timing-critical nets by setting setup threshold of 2ns and hold threshold of 0.2ns in our case. These thresholds mean if any timing path that has a setup margin or hold margin below or equal to these values, no spreading or widening is performed on any net of these paths.

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

Defects in vias is a serious issue and needs to be taken into account, **Redundant vias insertion** is an insertion of extra vias beside the original vias because if any via of both original via and extra via is defective , connection between metal layers will not fail as another contact will still connect metal layers together.

Metal filling is performed to protect metal interconnections in low metal density regions from over-etching during fabrication process. Timing driven metal filling specifically is performed to preserve timing on critical nets, metal fill near critical nets on the same layer, upper layer, and lower layer are removed or trimmed. FIGURE 3.20 shows the core after chip finishing step

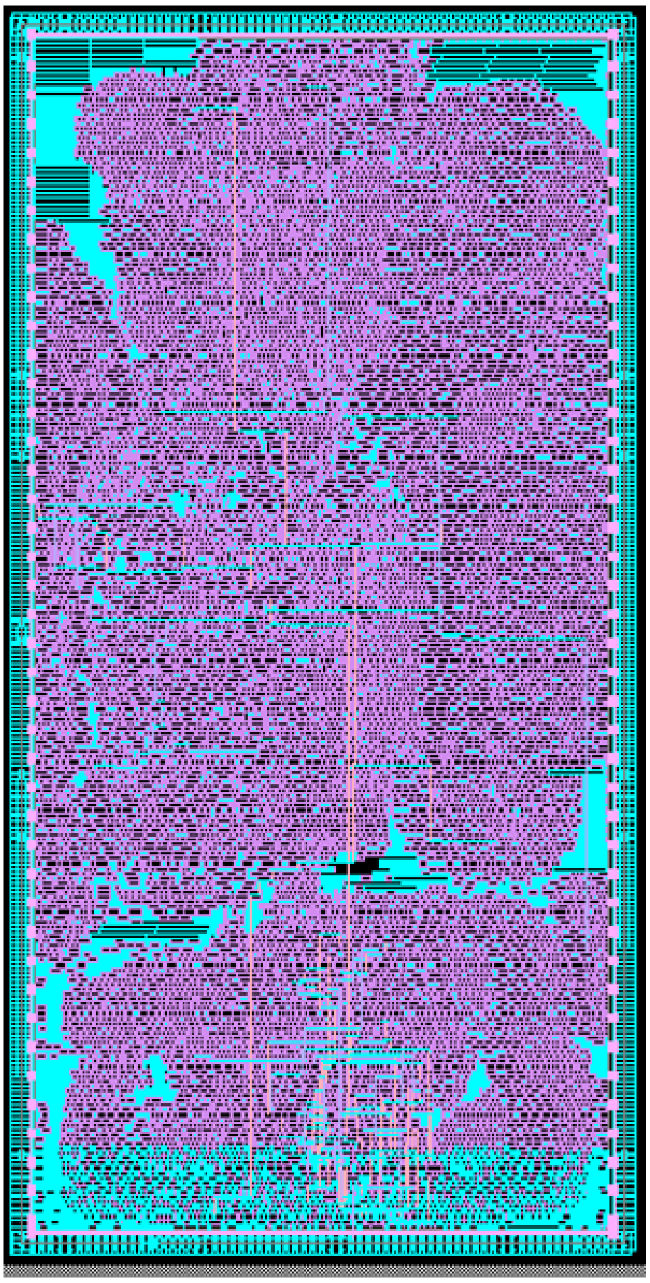


Figure 3.20: OpenPULP Core after chip finishing (Topographical Flow)

3.2.3 Hierarchical Flow

After the second pass logic synthesis ,we also tried to do full hierarchical flow in IC Compiler , but unfortunately the design has timing violations issues in the top level integration so we turns to a plan B which is to preserve the hierarchic y in the plan groups and complete the other PnR flow as the baseline flow .

Floorplanning

figure 3.21 shows the shape of plan groups .

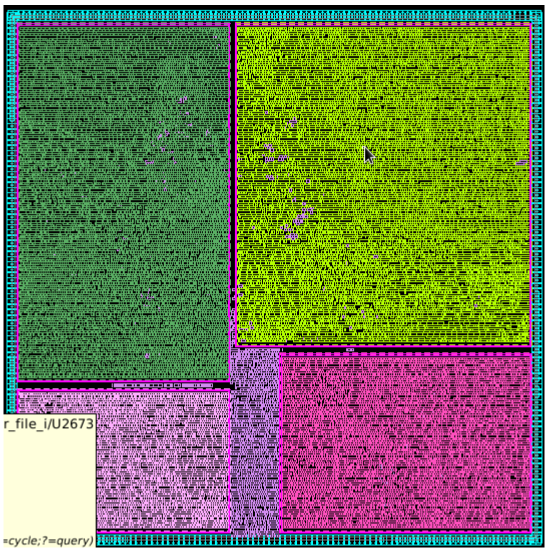


Figure 3.21: Plan Groups in Hierarchical Flow

Power Network Analysis

Figure 3.22 shows the power network synthesis map, we have synthesized the power network at voltage supply 1.5 Volt with maximum IR Drop of 10% of voltage supply value. Power network was synthesized successfully according to the constraints applied.

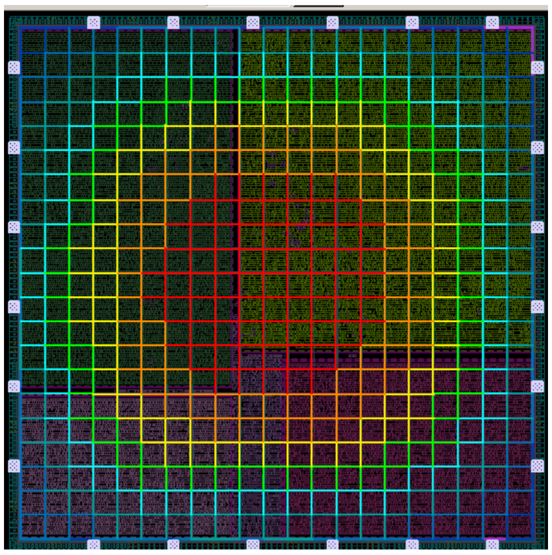


Figure 3.22: Power network synthesis map in Hierarchical Flow

Placement

As shown in figure 3.23, we can see that standard cells locations is preserved based upon the hierarchy commited from synthesis stage.

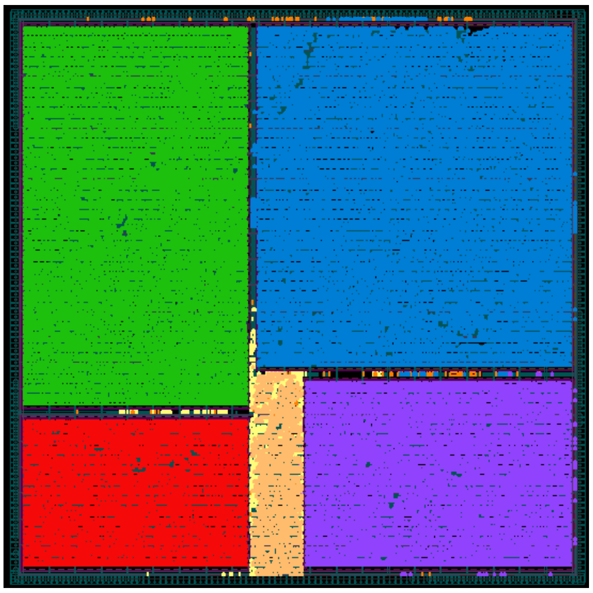


Figure 3.23: Hierarchy Placement in Hierarchical Flow

CTS

figure 3.24 shows the clock tree distribution across the design .

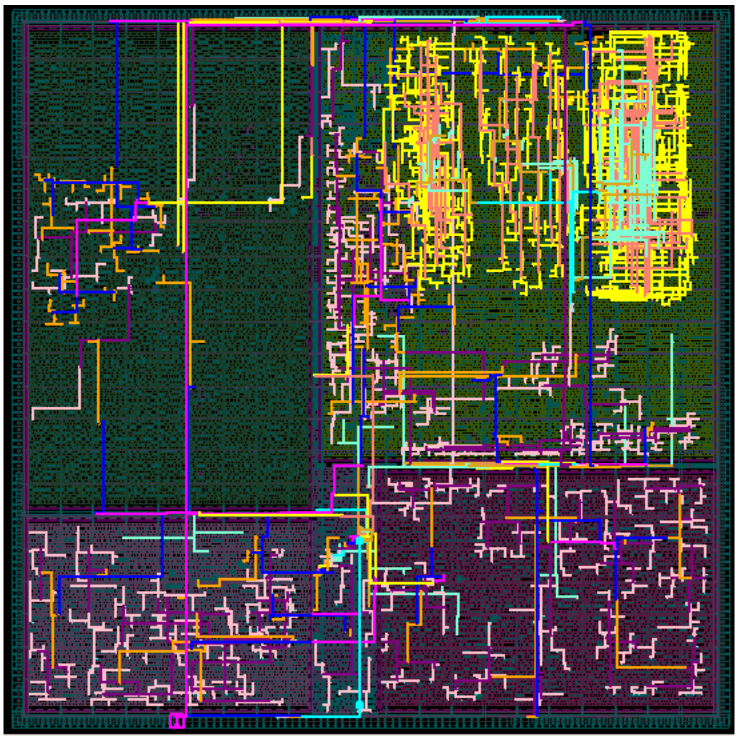


Figure 3.24: Clock tree network in Hierarchical Flow

Chip Finishing

figure 3.25 shows the chip in its final layout, several steps is made here such as standard cells filling to improve chip yield and to make the chip more uniform in density. Also metal filling is done here with taking into consideration critica-timing nets to be avoided so that no timing violations are being introduced. Metal spreading and widening is also made to prevent problems of over-etching and any shorts/opens in interconnections that could lead to a layout failure.

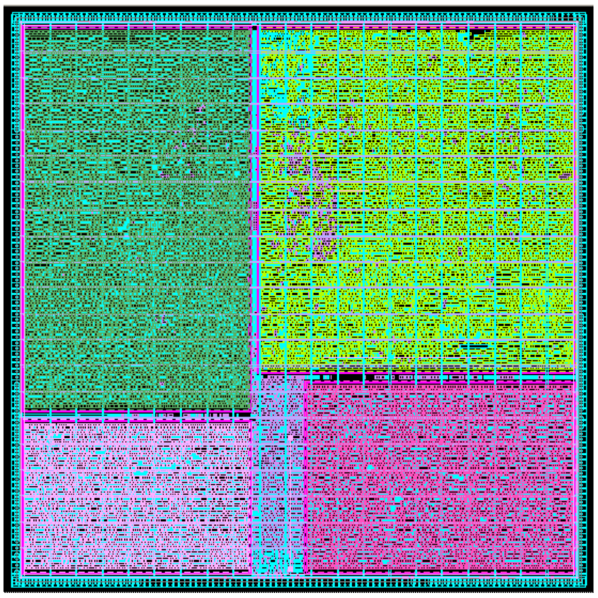
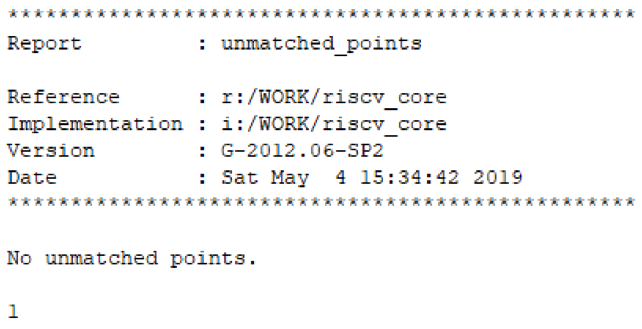


Figure 3.25: Chip finishing in Hierarchical Flow

3.3 Formal Equivalence Checking

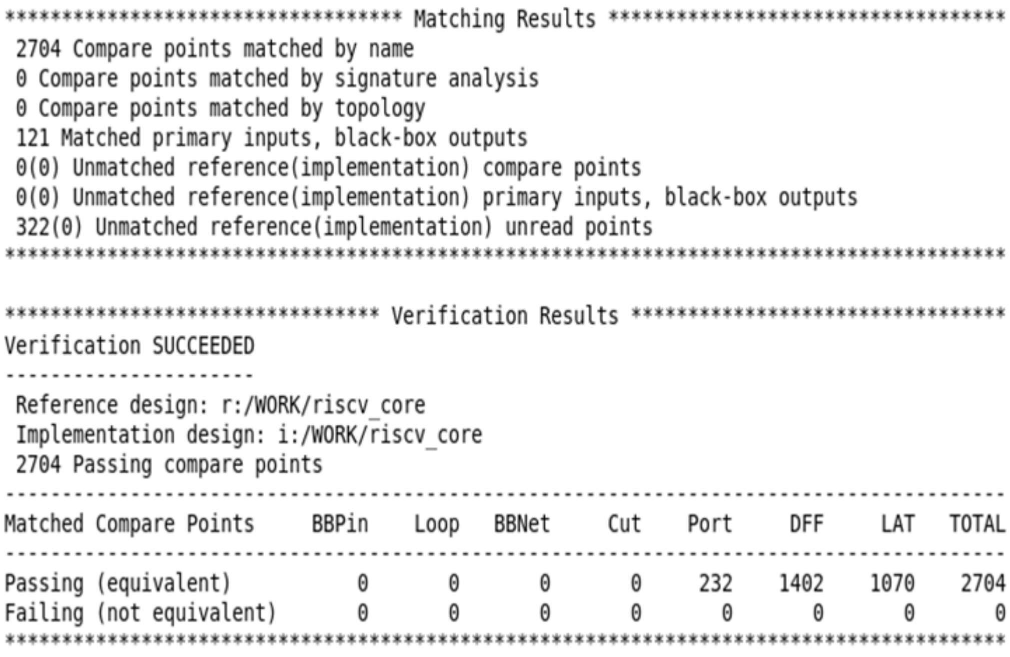
Equivalence checking is performed using Synopsys Formality tool. Verification can be done between RTL-Gate Netlist, Gate Netlist-Gate netlist and RTL-RTL. In this section, we will verify Post-layout netlist versus Pre-layout netlist to make sure they exhibit exactly the same behavior, first we perform compare points matching so that formality tool can perform formal verification properly. For all styles There is zero unmatched points which means that the environment is ready for verification as shown in Figure 3.26. Gate-to-Gate netlist verification is now performed.



```
**************************************************
Report         : unmatched_points

Reference      : r:/WORK/riscv_core
Implementation : i:/WORK/riscv_core
Version        : G-2012.06-SP2
Date           : Sat May  4 15:34:42 2019
**************************************************

No unmatched points.

1
```

Figure 3.26: Compare points Matching Report

3.3.1 Flat Flow



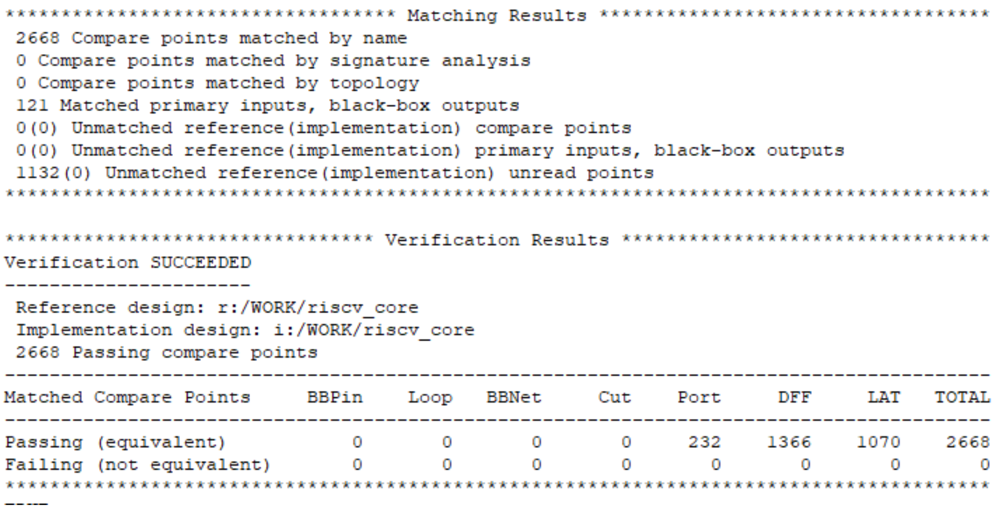
```
*********************************** Matching Results ***********************************
 2704 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 121 Matched primary inputs, black-box outputs
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
 322(0) Unmatched reference(implementation) unread points
****************************************************************************************

********************************* Verification Results *********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/riscv_core
 Implementation design: i:/WORK/riscv_core
 2704 Passing compare points
----------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
----------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0     232    1402    1070    2704
Failing (not equivalent)       0       0       0       0       0       0       0       0
****************************************************************************************
```

Figure 3.27: Formal Equivalence Report (Flat Flow)

FIGURE 3.27 shows that all compare points with total number of 2704 passed verification successfully.

3.3.2 Topographical Flow

Gate-to-Gate netlist verification is now performed. FIGURE 3.28 shows that all compare points with total number of 2668 passed verification successfully.

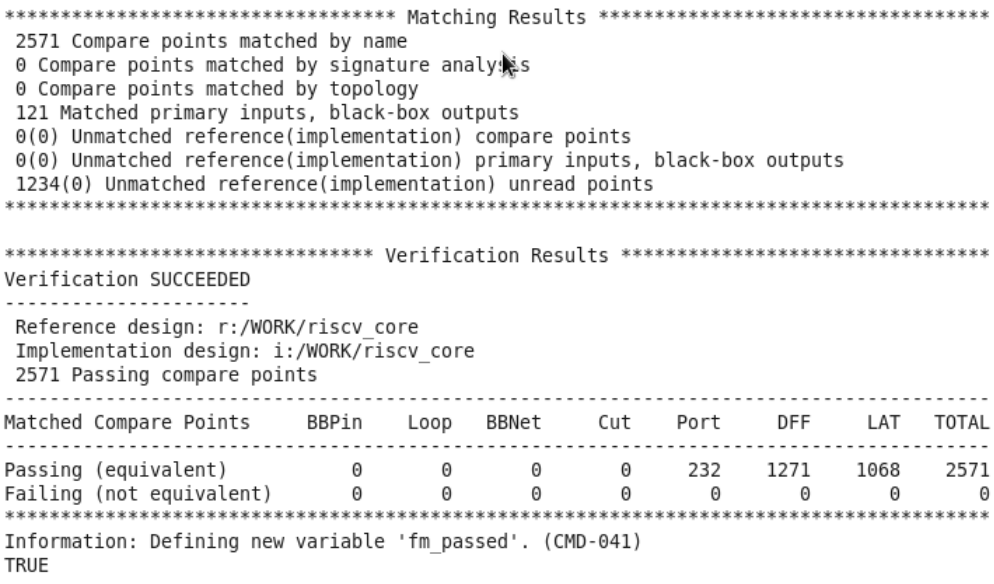


```
*********************************** Matching Results ***********************************
 2668 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 121 Matched primary inputs, black-box outputs
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
 1132(0) Unmatched reference(implementation) unread points
****************************************************************************************

********************************* Verification Results *********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/riscv_core
 Implementation design: i:/WORK/riscv_core
 2668 Passing compare points
----------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
----------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0     232    1366    1070    2668
Failing (not equivalent)       0       0       0       0       0       0       0       0
****************************************************************************************
----
```

Figure 3.28: Formal Equivalence Report (Topographical Flow)

3.3.3 Hierarchical Flow

Gate-to-Gate netlist verification is now performed. figure 3.29 shows that all compare points with total number of 2571 passed verification successfully.



```
*********************************** Matching Results ***********************************
 2571 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 121 Matched primary inputs, black-box outputs
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
 1234(0) Unmatched reference(implementation) unread points
****************************************************************************************

******************************** Verification Results ********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/riscv_core
 Implementation design: i:/WORK/riscv_core
 2571 Passing compare points
----------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
----------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0     232    1271    1068    2571
Failing (not equivalent)       0       0       0       0       0       0       0       0
****************************************************************************************
Information: Defining new variable 'fm_passed'. (CMD-041)
TRUE
```

Figure 3.29: Formality Equivalence Report in Hierarchical flow

3.4 Post-layout STA

3.4.1 Flat Flow

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

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

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

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

The last affecting factor is the MOSFET threshold voltage. There are three main types of threshold voltage based on MOS devices, and they are low voltage threshold (LVT), regular voltage threshold (RVT), high voltage threshold (HVT). This factor is also significantly affecting the delay of the standard cells. The low threshold-based cells have the lowest delay, and the highest one has the highest propagation delay. In our case, we have only regular voltage threshold-based libraries. This type of libraries has a variety of the previously mentioned operating conditions.

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

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

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

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

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

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

After finishing the setup successfully, we have stepped forward to analyze the design against the different cases. In most cases, the design had clean setup timing, and the problem was at the hold time. There are two was ways to solve such problems. The first one is to fix them in the layout tool and forward the design to layout tool to verify the results. The other one is to fix them in STA tool itself and to write the changes that have done to the design, and then forward them the layout tool to make these changes on the design, and finally the tool can write a new DDC-based file that contains the changes. That new file can be forwarded again to the STA tool for further analysis.

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

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

3.4.2 Topographical Flow

Post-layout static timing analysis is mandatory to verify performance of the design over several operating conditions. In this section we will describe briefly the procedure followed to verify OpenPULP Core over more than twenty operating conditions to ensure proper functionality without any violations.

Corner Name	**Process**	**Power Supply (V)**	**Temperature °C**
ss0p7v125c	Slow - Slow	0.7	125
ss0p75v125c	Slow - Slow	0.75	125
ss0p95v125c	Slow - Slow	0.95	125
ss0p95v25c	Slow - Slow	0.95	25
ss0p95vn40c	Slow - Slow	0.95	-40
tt1p05v125c	Typical - Typical	1.05	125
tt1p05v25c	Typical - Typical	1.05	25
tt1p05vn40c	Typical - Typical	1.05	-40
tt0p85v125c	Typical - Typical	0.85	125
tt0p85v25c	Typical - Typical	0.85	25
tt0p85vn40c	Typical - Typical	0.85	-40
tt0p78v125c	Typical - Typical	0.78	125
tt0p78v25c	Typical - Typical	0.78	25
tt0p78vn40c	Typical - Typical	0.78	-40
ff1p16v25c	Fast - Fast	1.16	25
ff1p16vn40c	Fast - Fast	1.16	-40
ff0p95v125c	Fast - Fast	0.95	125
ff0p95v25c	Fast - Fast	0.95	25
ff0p95vn40c	Fast - Fast	0.95	-40
ff0p85v125c	Fast - Fast	0.85	125
ff0p85v25c	Fast - Fast	0.85	25
ff0p85vn40c	Fast - Fast	0.85	-40

Table 3.1: Verified Corner Cases

TABLE 3.1 shows all the verified cases out of total cases of 27. The remaining five cases have DRC violations that require to go back and solve them during PnR stage, this requires to go through what is called Multi-corner Multi-mode PnR flow (MCMM). But the available resources (computer machine) couldn't help at the time of this thesis. However, the core can typically operate properly on 22 cases out of 27. The type of operating condition analysis used during STA is On-Chip variations (OCV) , it specifies that the minimum and maximum operating conditions for each corner case represent, respectively, the lower and upper bounds of the maximum variation of operating conditions on the chip. All maximum path delays use the maximum operating condition, and all minimum path delays use the minimum operating condition. Cells and net delays are derated by early derating factor of -10% and late derating factor by +10%. Long path delays (for example, data paths and launch clock path for setup checks or capture clock paths for hold checks) are derated by +10% , and short path delays (for example, capture clock paths for setup checks or data paths and launch clock paths for hold checks) are derated by -10%.

3.4.3 Hierarchical Flow

Post-layout STA is really important to confirm the layout design. Sometimes bad wires, I/Os and some other effects will change the parameters in circuits and the real performance badly. The parasitic capacitances extracted according to how your layout is designed might be critical in affecting the actual performance of your design. In order to get an idea of how the design would work from your layout, you should perform a post-layout STA from the extracted view. The procedure is identical to that for running STA verification on your logical netlist. So this helps you to get idea about how much deviation you're getting from both the results. General tolerance of 5% is accepted. If both pre-layout and post-layout STA matches exactly(or with the tolerance) then you're good to go. Our implementation of the core is analyzed using OCV analysis type, and also under time derateing factor by -10 % for the minimum paths and the longest paths have a derate factor by +10% to more restrict the design to meet timing requirements under the extreme condition.

Chapter 4

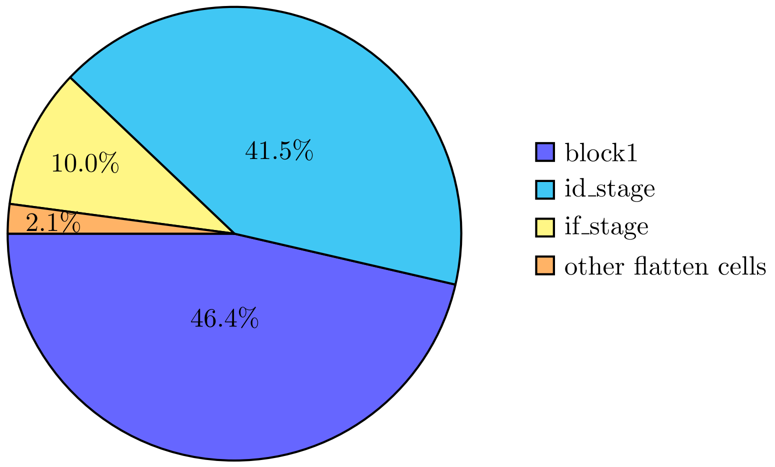
Results

4.1 Synthesis results

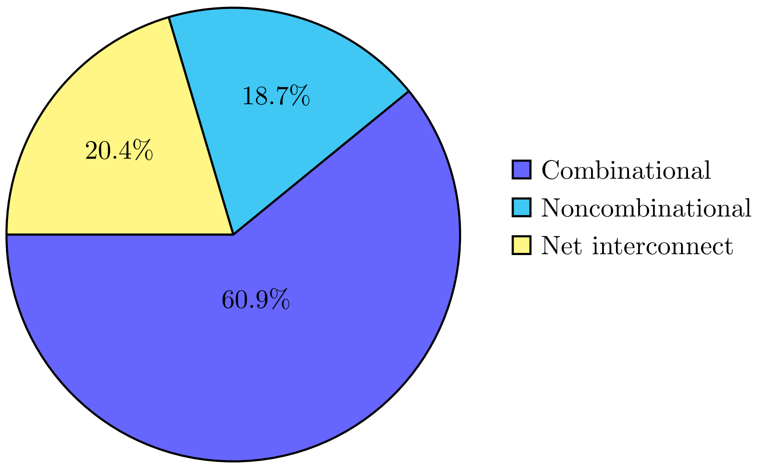
4.1.1 Flat Flow

Area results

	Initial results	After removing levels of hierarchy	After optimization
total cell area	69677.898602	66889.96648	65878.953995
net interconnect area	15983.773588	16765.1871	16876.197741
total area	85661.672189	83655.15358	82755.151736



Cell Area percentage distribution of Top level hierarchy



Total Area percentage distribution

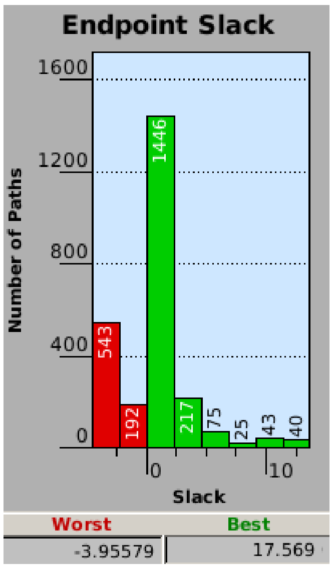
power results

	Initial results	After removing levels of hierarchy	After optimization
leakage power	2.0278 mw	1.9476 mw	1.9376 mw
switching power	17.9883 micron-watt	18.1487 micron-watt	18.3331 micron-watt
total power	2.0458 mw	1.96575 mw	1.9559 mw

As, shown in power table the final result indicates that the Leakage power which is the power consumed in transistor due to the constant current from Vdd to ground and its value is 1.9376 mw which is 99.06% of total power.
The switching power or dynamic power which is the power consmed in transistors due to switching from 1 to 0 or from 0 to 1 and its value is 18.3331 micron-watt which is 0.937% of total power.

timing results

1- Initial results:



setup slack of endpoints

2- After removing levels of hierarchy:

Here we solve violations by using group pathing method and setting maximum delayfor a certain paths.

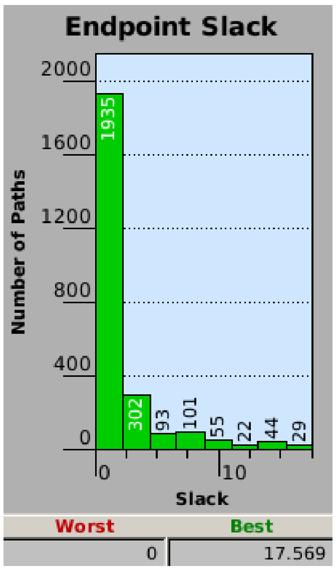


Figure 4.1: setup slack of endpoints (Flat Flow)

Here in figure 4.1 we succeed to make all paths met with the constraints but the timing range will be not enough when adding parasitic delay of wires after place and route phase, so we decide to optimize path's delays more than that. In figure4.2 the setup slacks of all paths (2581 path) are representet as:

- first column consists of 1935 path and is form 0 to 2.2
- second column consists of 302 path and is from 2.2 to 4.4
- third column consists of 93 path and is from 4.4 to 6.6
- forth column consists of 101 path and is from 6.6 to 8.8
- fifth column consists of 55 path and is from 8.8 to 11
- sixth column consists of 22 path and is from 11 to 13.2
- seventh column consists of 44 path and is from 13.2 to 15.4
- eighth column consists of 29 path and is from 15.4 to 17.6

3- After optimization:

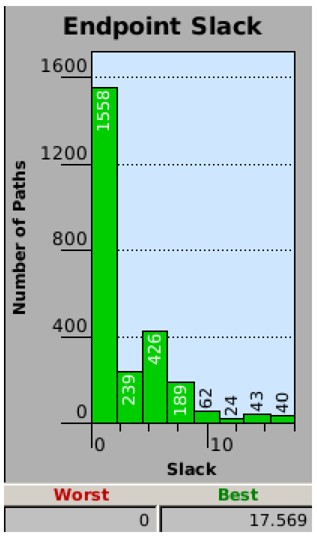


Figure 4.2: setup slack of endpoints after optimization (Flat Flow)

As shown in figure 4.2 the ranges of slack margin goes to right that is mean the delay of paths going smaller and more acceptable.In figure4.2 the setup slacks of all paths (2581 path) are representet as:

- first column consists of 1558 path and is form 0 to 2.2
- second column consists of 239 path and is from 2.2 to 4.4
- third column consists of 426 path and is from 4.4 to 6.6
- forth column consists of 189 path and is from 6.6 to 8.8
- fifth column consists of 62 path and is from 8.8 to 11
- sixth column consists of 24 path and is from 11 to 13.2
- seventh column consists of 43 path and is from 13.2 to 15.4
- eighth column consists of 40 path and is from 15.4 to 17.6

Note: There is 992 paths with slack 0 these paths are paths of latchs of register file as we use latch base register file, the tool use time borrowing technique to these paths for more timing optimization.

The time borrowing technique, which is also called cycle stealing, occurs at a latch. In a latch, one edge of the clock makes the latch transparent, that is, it opens the latch so that output of the latch is the same as the data input, this clock edge is called the opening edge. The second edge of the clock closes the latch, that is, any change on the data input is no longer available at the output of the latch, this clock edge is called the closing edge.

Typically, the data should be ready at a latch input before the active edge of the clock. However, since a latch is transparent when the clock is active, the data can arrive later than the active clock edge, that is, it can borrow time from the next cycle. If such time is borrowed, the time available for the following stage (latch to another sequential cell) is reduced.

we will represent more details of worst 10 paths except paths of latchs:

endpoint	data arrival time(ns)	data required time(ns)	slack (ns)
R_161	23.04	23.09	0.0401
U3355/U2944_ex_stage_i_R_160	23.04	23.09	0.040
instr_addr_o[18]	13.94	14.00	0.058
R_87/D	22.99	23.05	0.059
U3355/U2944_ex_stage_i_R_86/D	22.99	23.05	0.059
instr_addr_o[26]	13.94	14.00	0.059
id_stage_i/mult_dot_op_b_ex_o_reg_13_/D	22.98	23.04	0.060
id_stage_i/mult_operand_b_ex_o_reg_13_/D	22.98	23.04	0.060
id_stage_i/alu_operand_b_ex_o_reg_13_/D	22.98	23.04	0.060
instr_addr_o[11]	13.94	14.00	0.062

Table 4.1: worst SETUP slack of 10 paths after synthesis (Flat Flow)

note: U3355 represent block1 in hierarchy levels.

Hold time slacks of all paths (2581 path):

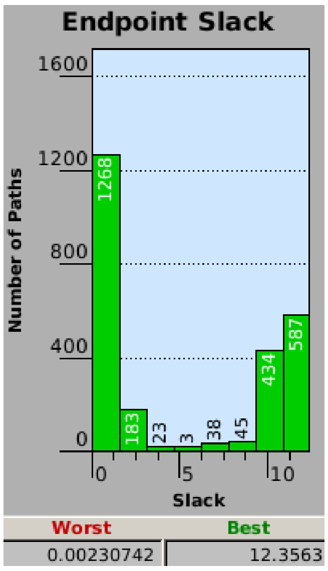


Figure 4.3: hold slack of endpoints after optimization (Flat Flow)

As shown in figure 4.3 here there is no hold time violations in worst case. we will represent more details of worst 10 paths:

endpoint	data arrival time time(ns)	data required time(ns)	slack (ns)
U3355/U2944_load_store_unit_i_rdata_q_reg_27_/D	5.1323	5.13	0.0023
U3355/U2944_load_store_unit_i_rdata_q_reg_29_/D	5.15	5.14	0.0047
U3355/U2944_load_store_unit_i_rdata_q_reg_26_/D	5.15	5.14	0.0047
if_stage_i/prefetch_32_prefetch_buffer_i_instr_addr_q_reg_22_/D	5.12	5.11	0.0089
U3355/U2944_load_store_unit_i_rdata_q_reg_24_/D	5.15	5.14	0.0102
if_stage_i/prefetch_32_prefetch_buffer_i_instr_addr_q_reg_31_/D	5.12	5.11	0.011626
id_stage_i/registers_i_riscv_register_file_i_wdata_b_q_reg_26_/D	5.16	5.15	0.01787
U3355/cs_registers_i_dscratch1_q_reg_21_/D	5.14	5.12	0.01812
U3355/U2944_load_store_unit_i_rdata_q_reg_25_/D	5.16	5.14	0.02187
U3355/cs_registers_i_dscratch1_q_reg_0_/D	5.14	5.12	0.02227

Table 4.2: worst HOLD slack of 10 paths after synthesis (Flat Flow)

note: U3355 represent block1 in hierarchy levels.

Synthesis Runtime:

The overall synthesis run time of Flat Flow is **3926.50** seconds which is about **1 hour and 5 minutes**.

4.1.2 Topographical Flow

Results of two-pass topographical flow will be in two categories:

1. First-Pass Synthesis Results
2. Second-Pass Synthesis Results

First-Pass Synthesis

Endpoint	Req time (ns)	Arrival time (ns)	Slack
ex_stage_imult_iDP_OP_101J6_125_8146R_194_IP	26.58	25.20	1.38
ex_stage_imult_iDP_OP_101J6_125_8146R_195	26.54	25.11	1.43
id_stage_imult_operand_b_ex_o_reg[24]	26.51	24.91	1.60
id_stage_ialu_operand_b_ex_o_reg[24]	26.51	24.91	1.60
id_stage_imult_operand_b_ex_o_reg[8]	26.53	24.89	1.65
id_stage_imult_dot_op_b_ex_o_reg[8]	26.53	24.89	1.65
id_stage_ialu_operand_b_ex_o_reg[8]	26.53	24.89	1.65
id_stage_imult_dot_op_b_ex_o_reg[24]	26.52	24.85	1.67
id_stage_ialu_operand_b_ex_o_reg[31]	26.53	24.85	1.68
id_stage_imult_dot_op_b_ex_o_reg[31]	26.53	24.85	1.68

Table 4.3: worst 10 setup paths after synthesis (Topographical Flow)

Endpoint	Req time (ns)	Arrival time (ns)	Slack
id_stage_iimm_vec_ext_ex_o_reg[1]	7.79	7.62	0.17
ex_stage_ialu_iint_div div_iCnt_DP_reg[0]	7.96	7.64	0.32
id_stage_idata_type_ex_o_reg[0]	8.00	7.64	0.37
cs_registers_iPCCR_q_reg[0][2]	8.00	7.62	0.38
id_stage_imult_operator_ex_o_reg[1]	8.00	7.61	0.39
cs_registers_iPCCR_q_reg[0][28]	8.03	7.62	0.41
cs_registers_iPCCR_q_reg[0][24]	8.03	7.62	0.41
cs_registers_iPCCR_q_reg[0][20]	8.03	7.62	0.41
cs_registers_iPCCR_q_reg[0][18]	8.03	7.62	0.41
cs_registers_iPCCR_q_reg[0][8]	8.03	7.62	0.41

Table 4.4: worst 10 hold paths after synthesis (Topographical Flow)

TABLEs 4.3 and 4.4 show the most critical 10 setup/hold timing paths in the design.

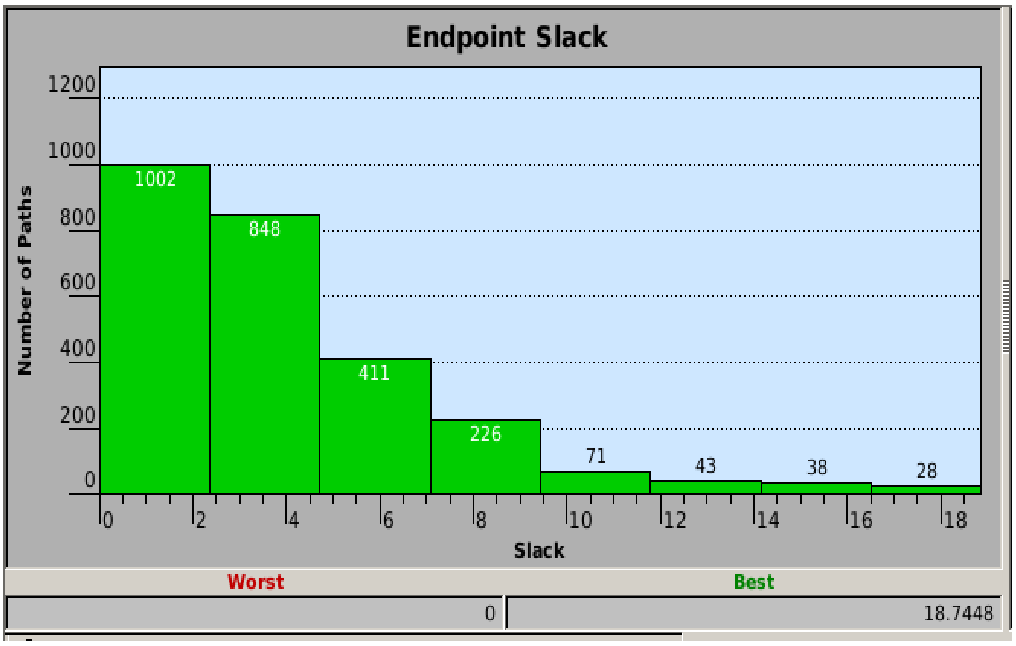


Figure 4.4: histogram for setup paths (Topographical Flow)

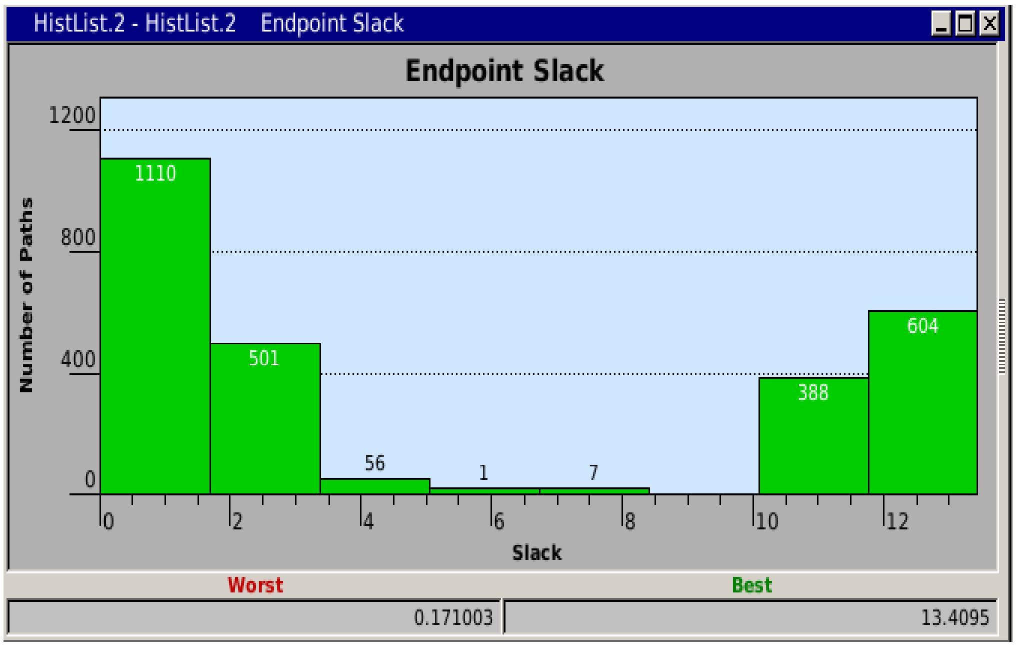


Figure 4.5: histogram for hold paths (Topographical Flow)

FIGUREs 4.4 and 4.5 show design setup/hold timing status. As latches are level sensitive, time borrowing is the property of a latch by virtue of which a path ending at a latch can borrow time from the next path in pipeline such that the overall time of the two paths remains the same. The time borrowed by the latch from next stage in pipeline is, then, subtracted from the next path's time.

Area Results

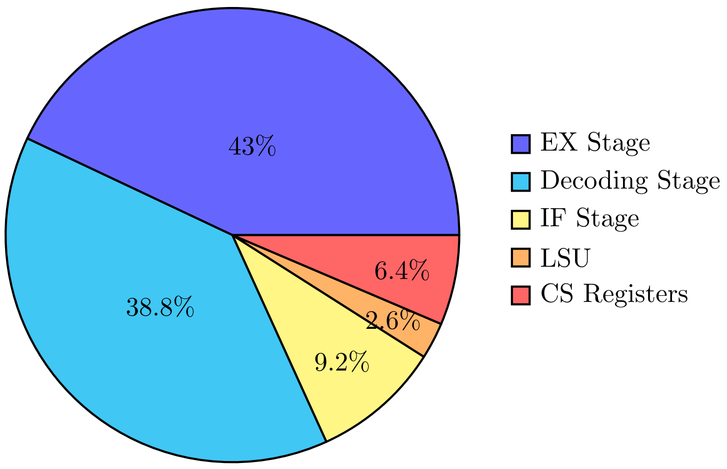


Figure 4.6: Cell Area Distribution across hierarchy (Topographical Flow)

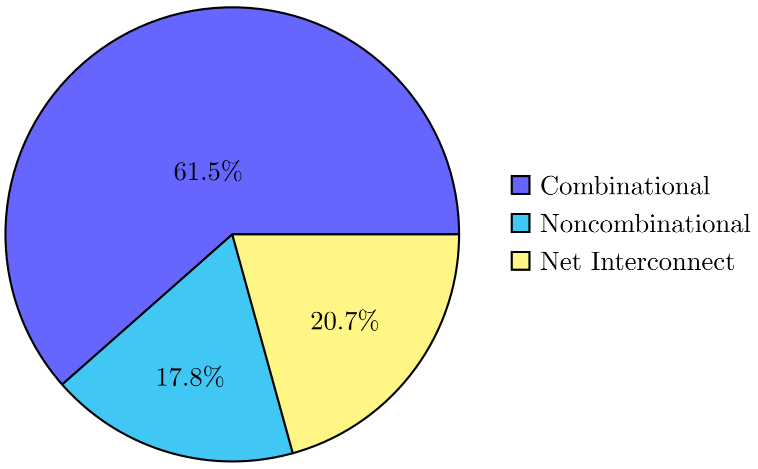
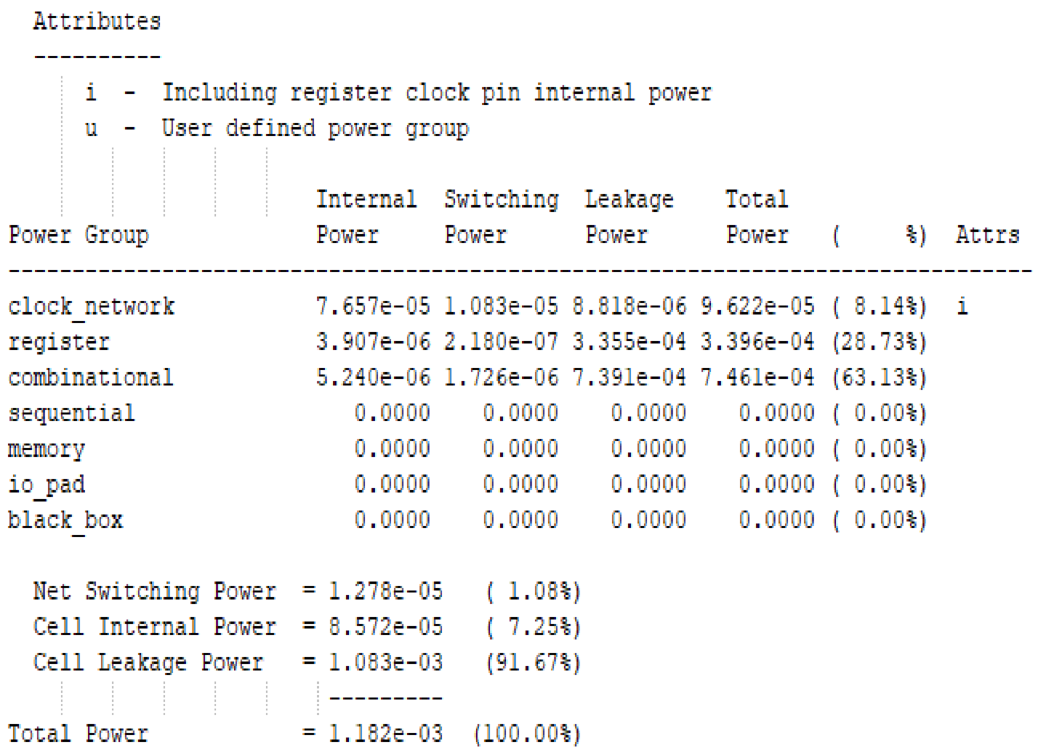


Figure 4.7: Total Area Distribution (Topographical Flow)

FIGURE 4.27 shows distribution of cell area across hierarchy blocks, we can notice that both EX stage and Decoding stage are two major blocks compared to other hierarchy blocks. FIGURE 4.28 also shows design area classified into combinational,non-combinational and interconnections occupied area.

Power Results



```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           7.657e-05 1.083e-05 8.818e-06 9.622e-05 ( 8.14%)  i
register                3.907e-06 2.180e-07 3.355e-04 3.396e-04 (28.73%)
combinational           5.240e-06 1.726e-06 7.391e-04 7.461e-04 (63.13%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 1.278e-05   ( 1.08%)
  Cell Internal Power  = 8.572e-05   ( 7.25%)
  Cell Leakage Power   = 1.083e-03   (91.67%)
                         ---------
Total Power            = 1.182e-03  (100.00%)
```

Figure 4.8: Total Power Report (Topographical Flow)

Total power is measured to be 2.14 mW , major power component is cell leakage power as shown in FIGURE 4.8. Solution for such an increase in cell leakage power is to switch from RVT cells to HVT cells as an example, however, increase in cell delays will occur.

Synthesis Runtime

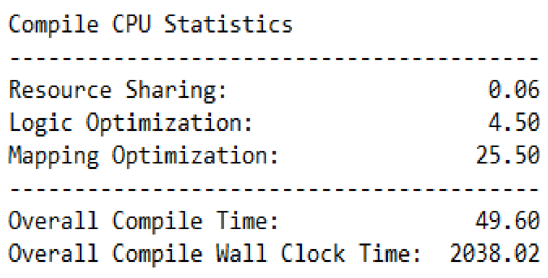

```
Compile CPU Statistics
-----------------------------------------
Resource Sharing:                    0.06
Logic Optimization:                  4.50
Mapping Optimization:               25.50
-----------------------------------------
Overall Compile Time:               49.60
Overall Compile Wall Clock Time:  2038.02
```

Figure 4.9: Synthesis Runtime Report (Topographical Flow)

Synthesis runtime is measured to be 2038 seconds equivalent to about 33 minutes as shown in FIGURE 4.9.

Second-Pass Synthesis

Endpoint	**Req time (ns)**	**Arrival time (ns)**	**Slack**
id_stage_imult_dot_op_b_ex_o_reg_18	26.45	26.45	0.001
id_stage_ialu_operand_b_ex_o_reg_18	26.45	26.45	0.001
id_stage_imult_operand_b_ex_o_reg_18	26.45	26.45	0.001
id_stage_imult_dot_op_b_ex_o_reg_16	26.46	26.45	0.01
id_stage_imult_operand_b_ex_o_reg_16	26.46	26.45	0.01
id_stage_ialu_operand_b_ex_o_reg_16	26.46	26.45	0.01
id_stage_imult_operand_b_ex_o_reg_25	26.52	26.49	0.03
id_stage_imult_dot_op_b_ex_o_reg_25	26.52	26.49	0.03
id_stage_ialu_operand_b_ex_o_reg_25	26.52	26.49	0.03
id_stage_imult_operand_b_ex_o_reg_26	26.53	26.50	0.03

Table 4.5: worst 10 setup paths in second pass synthesis (Topographical Flow)

Endpoint	**Req time (ns)**	**Arrival time (ns)**	**Slack**
id_stage_ihwloop_regs_ihwlp_counter_q_reg_0_0	7.93	7.64	0.29
id_stage_ihwloop_regs_ihwlp_counter_q_reg_1_0	7.94	7.64	0.30
ex_stage_iregfile_waddr_lsu_reg_0	7.92	7.61	0.32
ex_stage_iregfile_waddr_lsu_reg_1	7.92	7.60	0.32
ex_stage_ialu_iint_div_div_iCnt_DP_reg_0	7.96	7.64	0.33
ex_stage_iregfile_waddr_lsu_reg_4	7.93	7.60	0.33
ex_stage_iregfile_waddr_lsu_reg_2	7.93	7.60	0.33
ex_stage_iregfile_waddr_lsu_reg_3	7.94	7.60	0.34
ex_stage_ialu_iint_div_div_iResReg_DP_reg_22	8.01	7.64	0.36
id_stage_iint_controller_iexc_ctrl_cs_reg_0	8.00	7.62	0.37

Table 4.6: worst 10 hold paths in second pass synthesis (Topographical Flow)

TABLEs 4.5 and 4.6 show the most critical 10 setup/hold timing paths in the design. Latch-based paths are not shown in these tables since their setup slack margin is 0 due to time-borrowing effect.

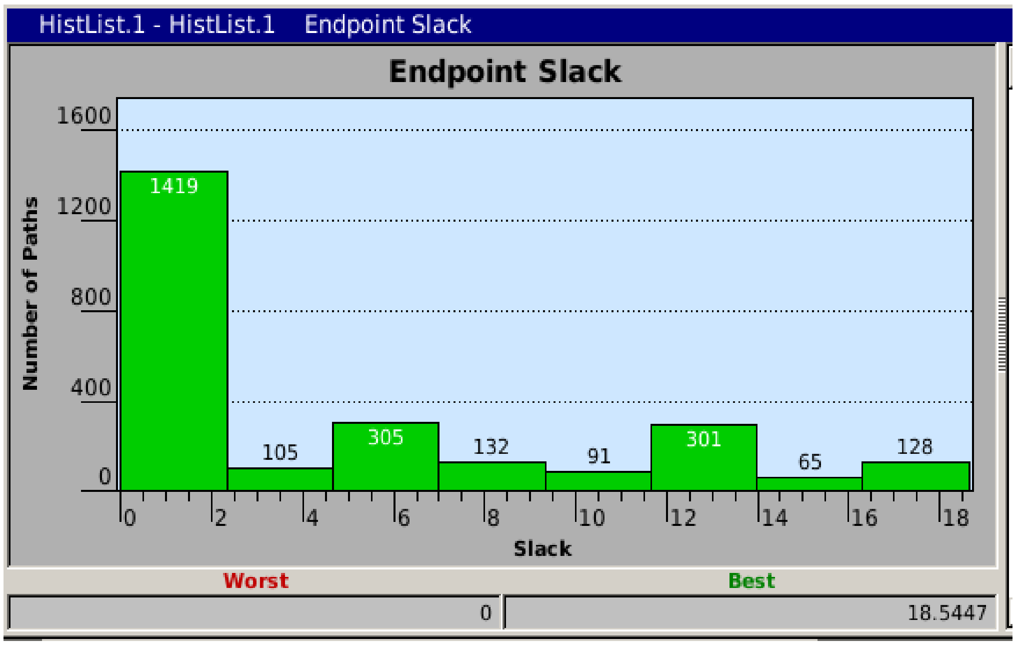


Figure 4.10: histogram for setup paths in second pass synthesis (Topographical Flow)

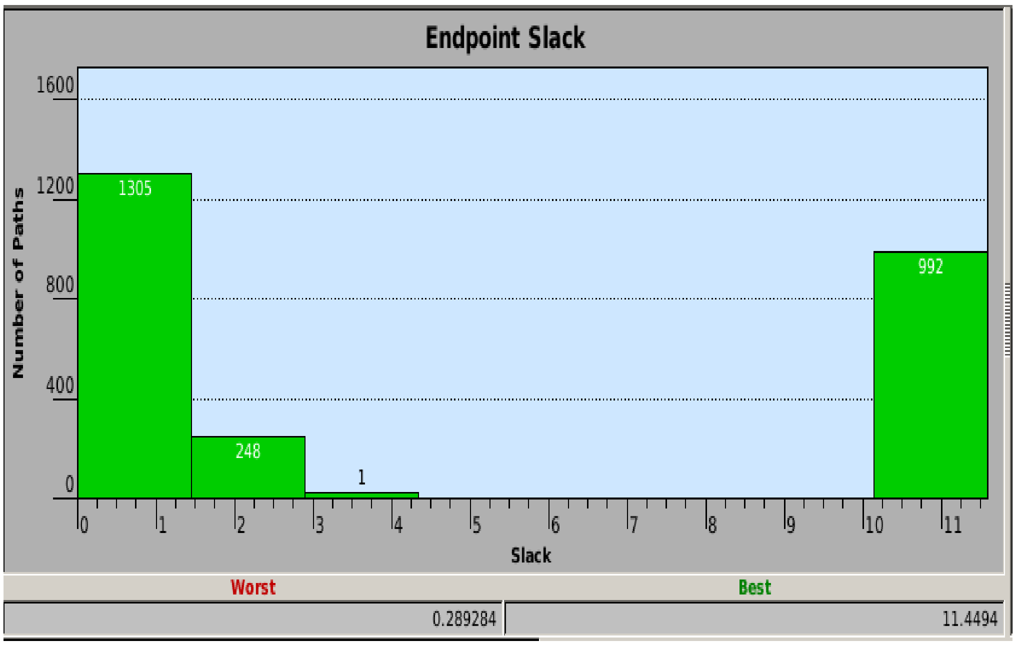


Figure 4.11: histogram for hold paths in second pass synthesis (Topographical Flow)

FIGUREs 4.10 and 4.11 show design setup/hold timing status. From setup timing histogram we can notice more uniform distribution of paths along slack range relative to the first iteration's setup timing histogram.

Area Results

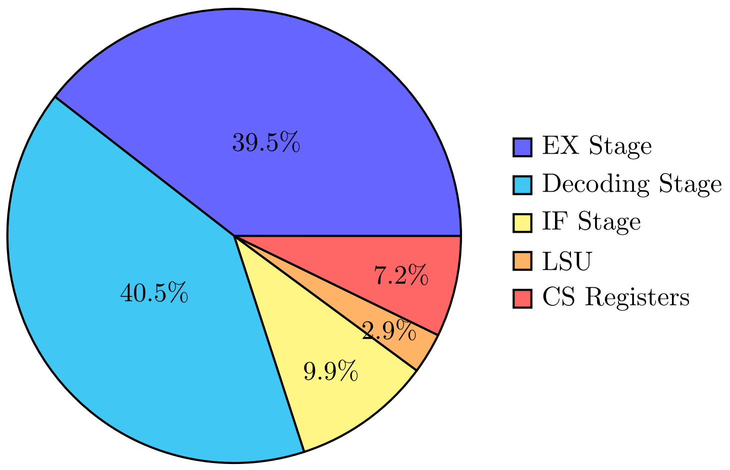


Figure 4.12: Cell Area Distribution across hierarchy in second pass synthesis (Topographical Flow)

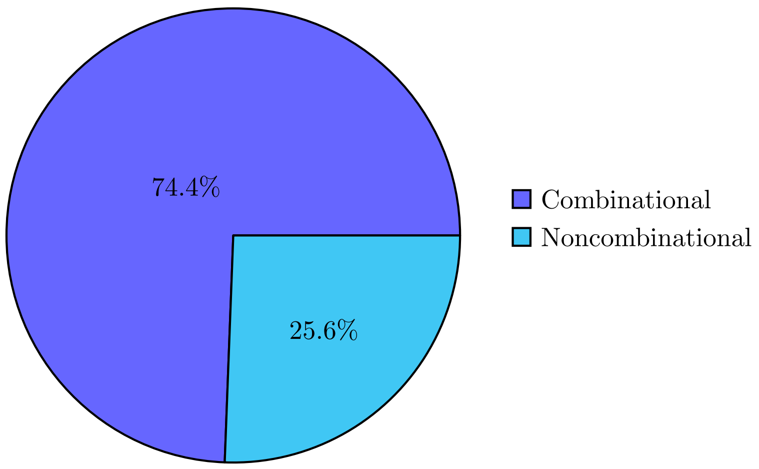
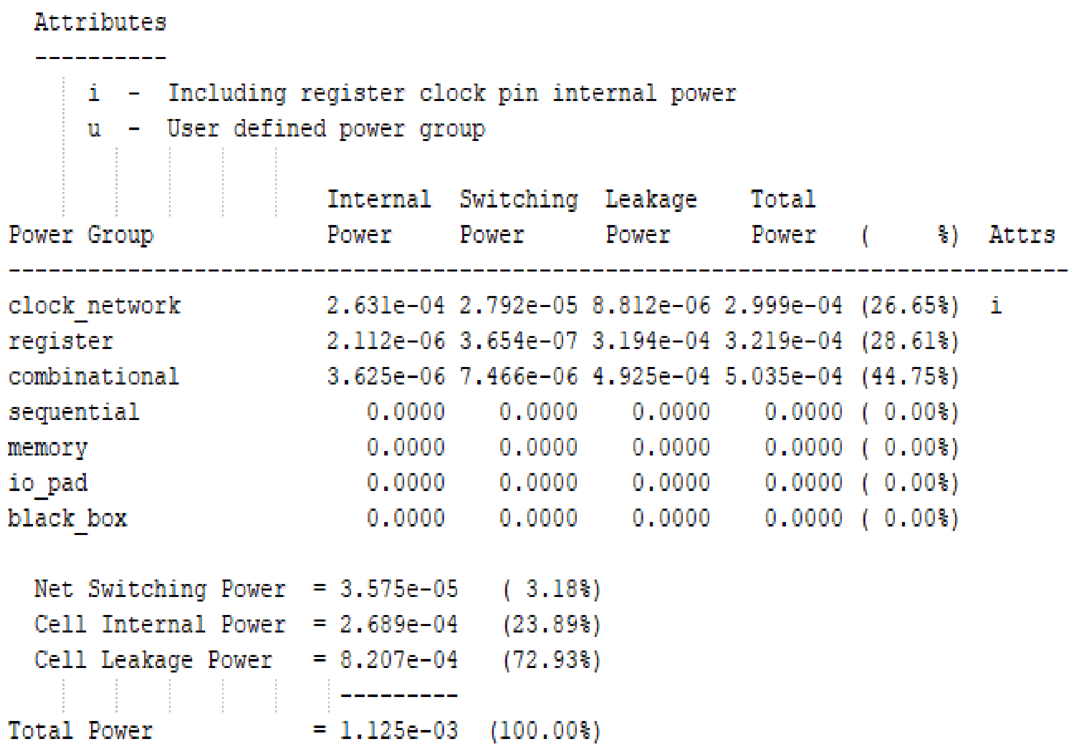


Figure 4.13: Total Area Distribution in second pass synthesis (Topographical Flow)

FIGURE 4.12 shows distribution of cell area across hierarchy blocks, we can notice that both EX stage and Decoding stage are two major blocks compared to other hierarchy blocks. FIGURE 4.13 also shows design area classified into combinational and non-combinational occupied area. We can see that net interconnect area is not included in statistics of FIGURE 4.13, since in topographical synthesis, interconnect parasitics are used in delay calculations rather than WLMs.

Power Results

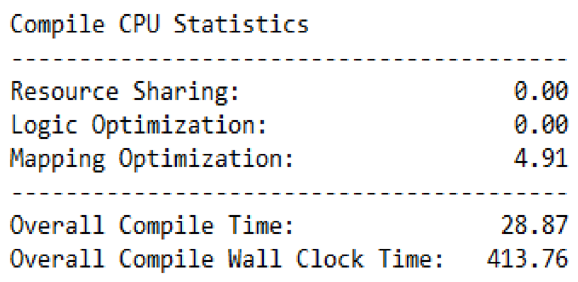


```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power    (      %)  Attrs
--------------------------------------------------------------------------------
clock_network           2.631e-04 2.792e-05 8.812e-06 2.999e-04 (26.65%)  i
register                2.112e-06 3.654e-07 3.194e-04 3.219e-04 (28.61%)
combinational           3.625e-06 7.466e-06 4.925e-04 5.035e-04 (44.75%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 3.575e-05   ( 3.18%)
  Cell Internal Power  = 2.689e-04   (23.89%)
  Cell Leakage Power   = 8.207e-04   (72.93%)
                         ---------
Total Power            = 1.125e-03  (100.00%)
```

Figure 4.14: Total Power Report in second pass synthesis (Topographical Flow)

Total power is measured to be 1.76 mW , major power component is still cell leakage power as shown in FIGURE 4.14.

Synthesis Runtime



```
Compile CPU Statistics
-----------------------------------------
Resource Sharing:                    0.00
Logic Optimization:                  0.00
Mapping Optimization:                4.91
-----------------------------------------
Overall Compile Time:               28.87
Overall Compile Wall Clock Time:   413.76
```

Figure 4.15: Synthesis Runtime Report (Topographical Flow)

Synthesis runtime is measured to be 413 seconds equivalent to about 7 minutes as shown in FIGURE 4.15.

4.1.3 Hierarchical Flow

Results of two-pass topographical flow will be in two categories:

1. First-Pass Synthesis Results

2. Second-Pass Synthesis Results

First-pass Synthesis

Figures 4.16 , 4.17 , 4.18 and 4.18 show that the design has no timing violation in first-pass synthesis.

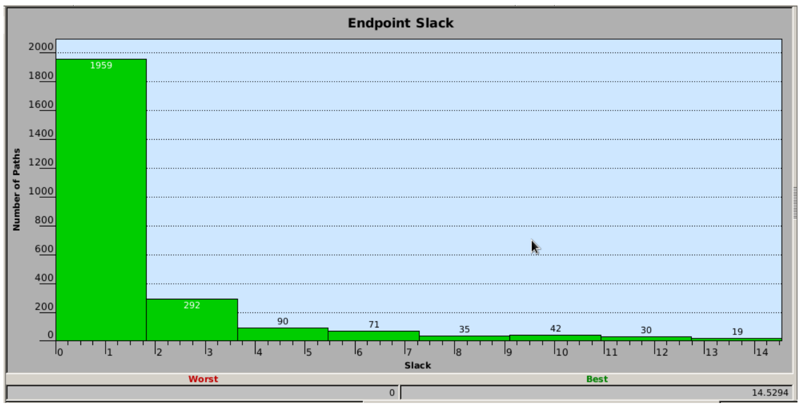


Figure 4.16: Setup timing histogram of first-pass synthesis in Hierarchical Flow

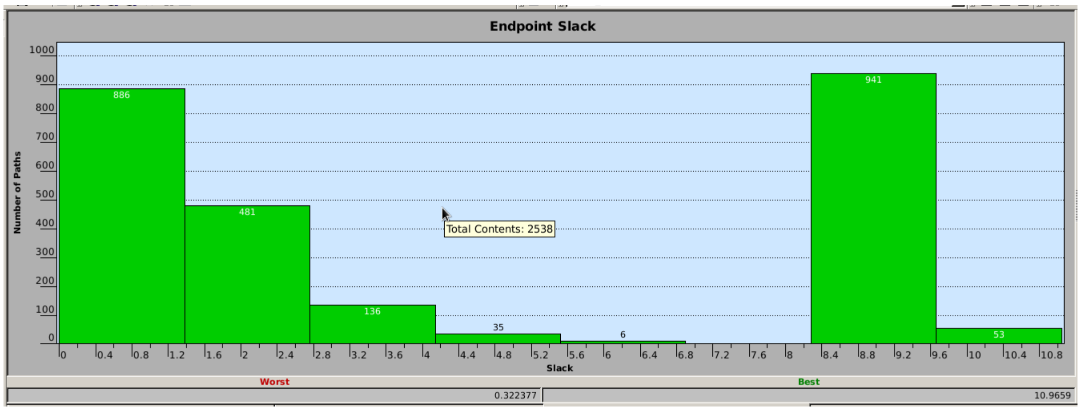
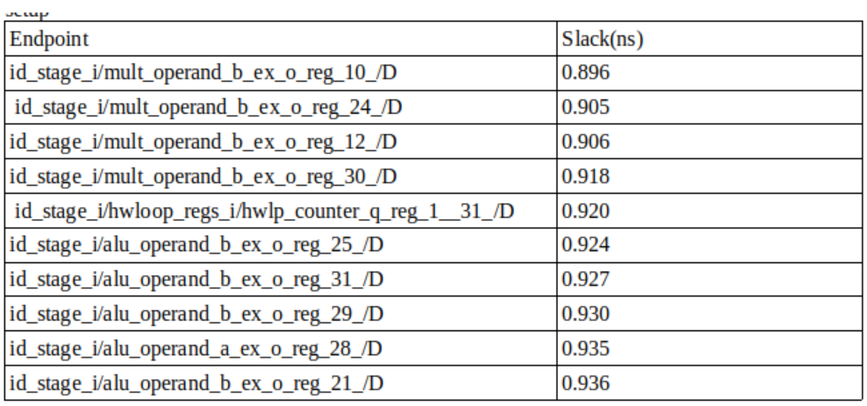
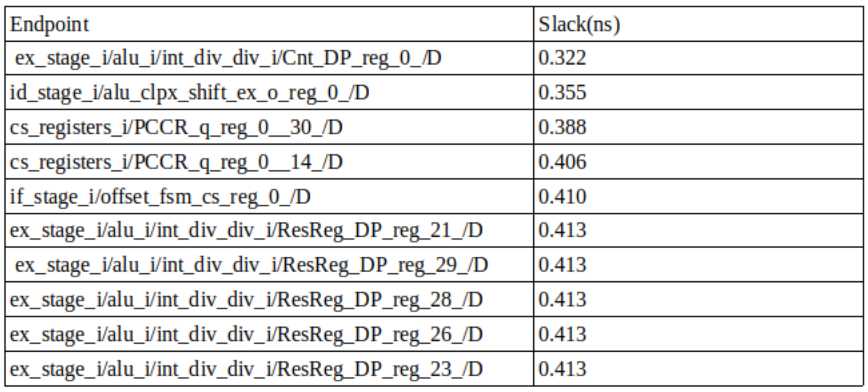


Figure 4.17: Hold timing histogram of first-pass synthesis in Hierarchical Flow



Endpoint	Slack(ns)
id_stage_i/mult_operand_b_ex_o_reg_10_/D	0.896
id_stage_i/mult_operand_b_ex_o_reg_24_/D	0.905
id_stage_i/mult_operand_b_ex_o_reg_12_/D	0.906
id_stage_i/mult_operand_b_ex_o_reg_30_/D	0.918
id_stage_i/hwloop_regs_i/hwlp_counter_q_reg_1__31_/D	0.920
id_stage_i/alu_operand_b_ex_o_reg_25_/D	0.924
id_stage_i/alu_operand_b_ex_o_reg_31_/D	0.927
id_stage_i/alu_operand_b_ex_o_reg_29_/D	0.930
id_stage_i/alu_operand_a_ex_o_reg_28_/D	0.935
id_stage_i/alu_operand_b_ex_o_reg_21_/D	0.936

Figure 4.18: Worst 10 Setup timing paths in first-pass synthesis in Hierarchical Flow



Endpoint	Slack(ns)
ex_stage_i/alu_i/int_div_div_i/Cnt_DP_reg_0_/D	0.322
id_stage_i/alu_clpx_shift_ex_o_reg_0_/D	0.355
cs_registers_i/PCCR_q_reg_0__30_/D	0.388
cs_registers_i/PCCR_q_reg_0__14_/D	0.406
if_stage_i/offset_fsm_cs_reg_0_/D	0.410
ex_stage_i/alu_i/int_div_div_i/ResReg_DP_reg_21_/D	0.413
ex_stage_i/alu_i/int_div_div_i/ResReg_DP_reg_29_/D	0.413
ex_stage_i/alu_i/int_div_div_i/ResReg_DP_reg_28_/D	0.413
ex_stage_i/alu_i/int_div_div_i/ResReg_DP_reg_26_/D	0.413
ex_stage_i/alu_i/int_div_div_i/ResReg_DP_reg_23_/D	0.413

Figure 4.19: Worst 10 hold timing paths in first-pass synthesis in Hierarchical Flow

Area Results

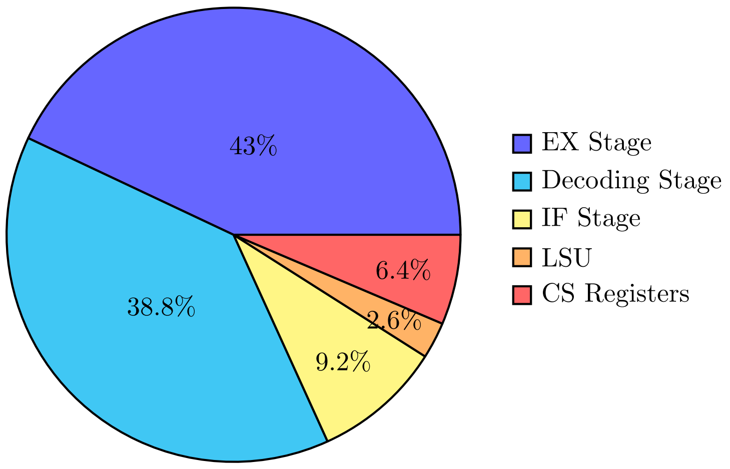


Figure 4.20: Cell Area Distribution across hierarchy in Hierarchical Flow

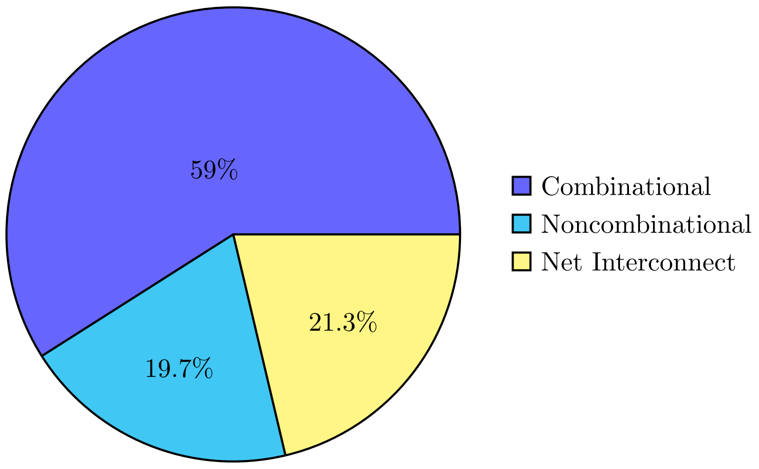
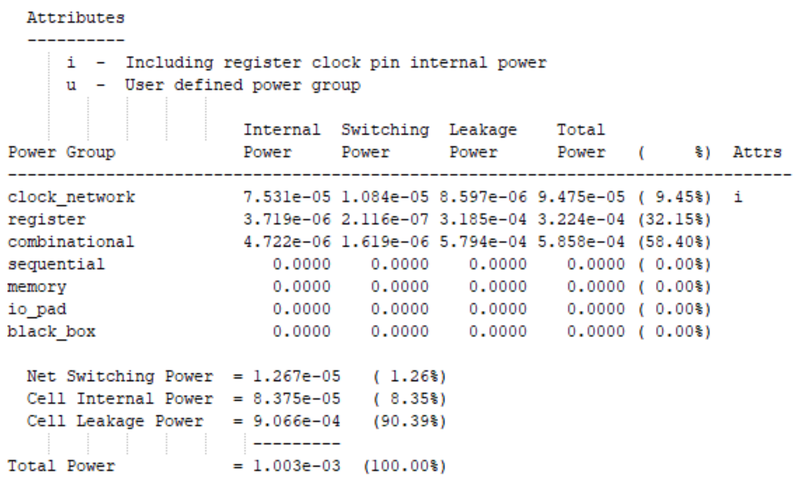


Figure 4.21: Total Area Distribution in Hierarchical Flow

Power Results



```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           7.531e-05 1.084e-05 8.597e-06 9.475e-05 ( 9.45%)  i
register                3.719e-06 2.116e-07 3.185e-04 3.224e-04 (32.15%)
combinational           4.722e-06 1.619e-06 5.794e-04 5.858e-04 (58.40%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 1.267e-05   ( 1.26%)
  Cell Internal Power  = 8.375e-05   ( 8.35%)
  Cell Leakage Power   = 9.066e-04   (90.39%)
                         ---------
Total Power            = 1.003e-03  (100.00%)
```

Figure 4.22: Total Power Report of Hierarchical Flow in first-pass synthesis

Second-pass Synthesis

Figures 4.23 and 4.24 show that the design has no timing violations

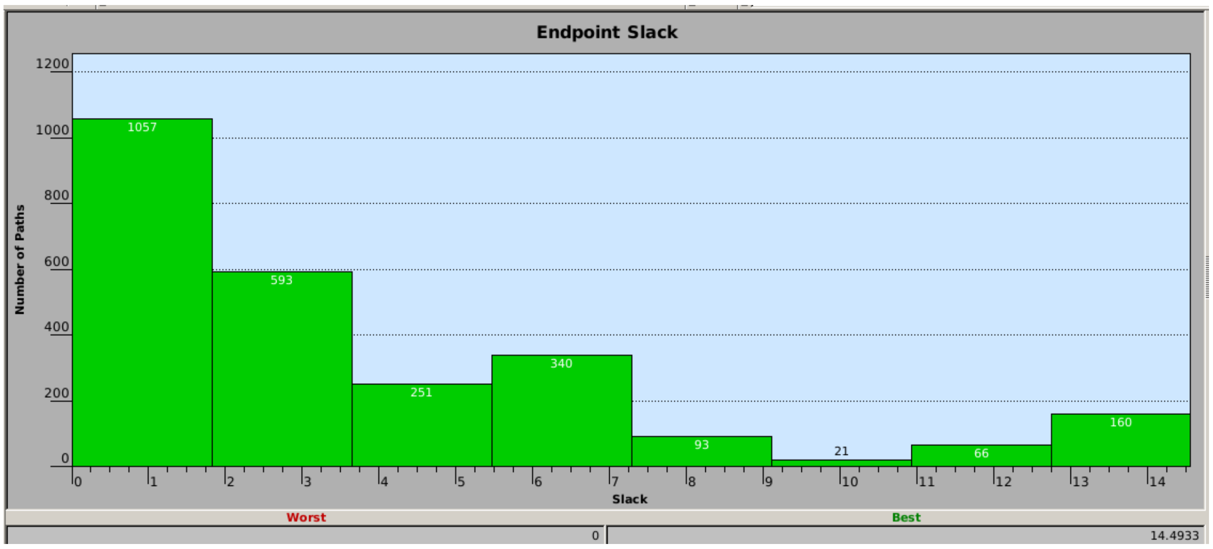


Figure 4.23: Setup timing histogram of second-pass synthesis in Hierarchical Flow

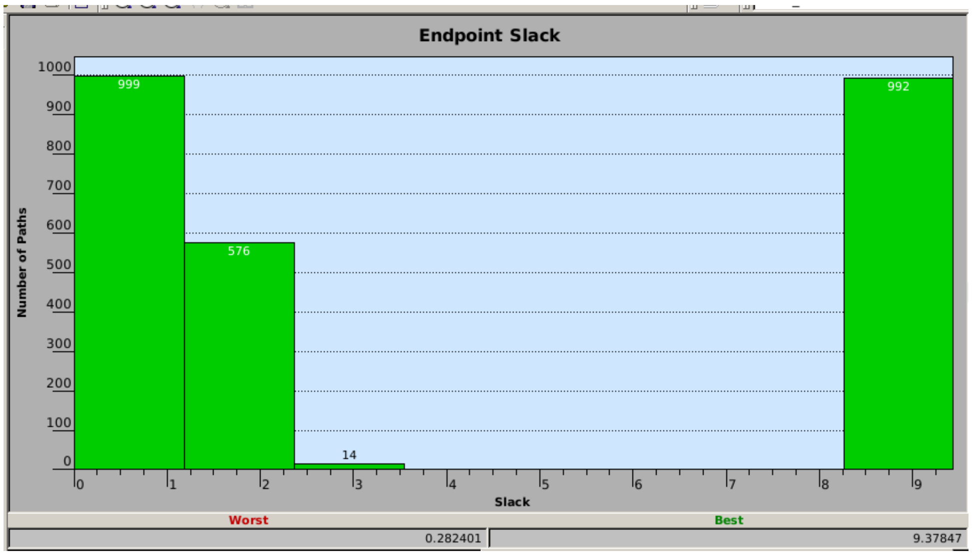
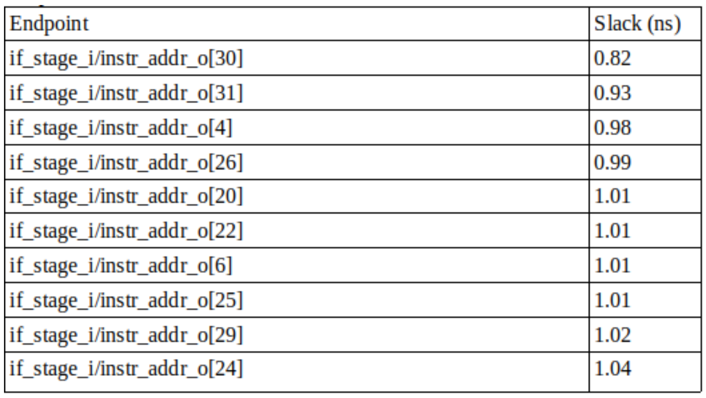
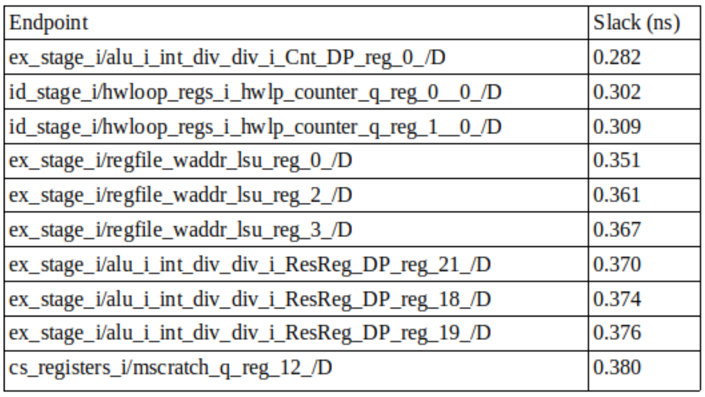


Figure 4.24: Hold timing histogram of second-pass synthesis in Hierarchical Flow



Endpoint	Slack (ns)
if_stage_i/instr_addr_o[30]	0.82
if_stage_i/instr_addr_o[31]	0.93
if_stage_i/instr_addr_o[4]	0.98
if_stage_i/instr_addr_o[26]	0.99
if_stage_i/instr_addr_o[20]	1.01
if_stage_i/instr_addr_o[22]	1.01
if_stage_i/instr_addr_o[6]	1.01
if_stage_i/instr_addr_o[25]	1.01
if_stage_i/instr_addr_o[29]	1.02
if_stage_i/instr_addr_o[24]	1.04

Figure 4.25: Worst 10 setup timing paths in second-pass synthesis in Hierarchical Flow



Endpoint	Slack (ns)
ex_stage_i/alu_i_int_div_div_i_Cnt_DP_reg_0_/D	0.282
id_stage_i/hwloop_regs_i_hwlp_counter_q_reg_0__0_/D	0.302
id_stage_i/hwloop_regs_i_hwlp_counter_q_reg_1__0_/D	0.309
ex_stage_i/regfile_waddr_lsu_reg_0_/D	0.351
ex_stage_i/regfile_waddr_lsu_reg_2_/D	0.361
ex_stage_i/regfile_waddr_lsu_reg_3_/D	0.367
ex_stage_i/alu_i_int_div_div_i_ResReg_DP_reg_21_/D	0.370
ex_stage_i/alu_i_int_div_div_i_ResReg_DP_reg_18_/D	0.374
ex_stage_i/alu_i_int_div_div_i_ResReg_DP_reg_19_/D	0.376
cs_registers_i/mscratch_q_reg_12_/D	0.380

Figure 4.26: Worst 10 hold timing paths in second-pass synthesis in Hierarchical Flow

Area Results

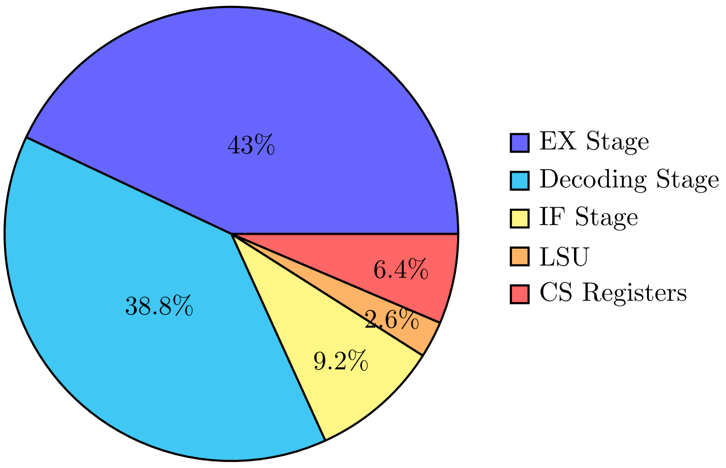


Figure 4.27: Cell Area Distribution across hierarchy in Hierarchical Flow

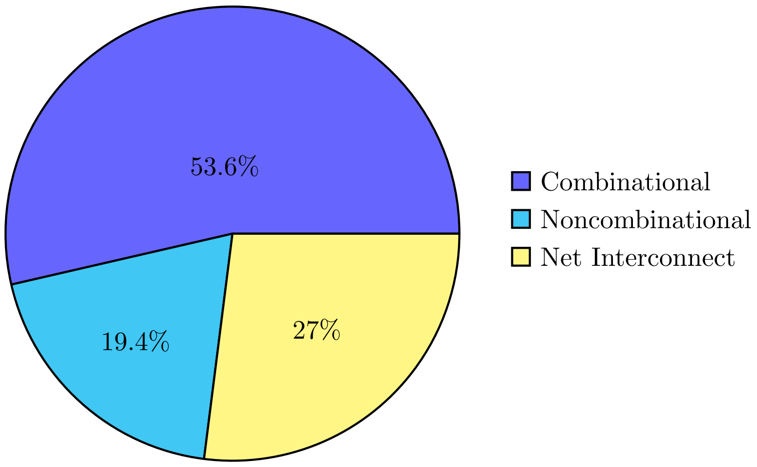
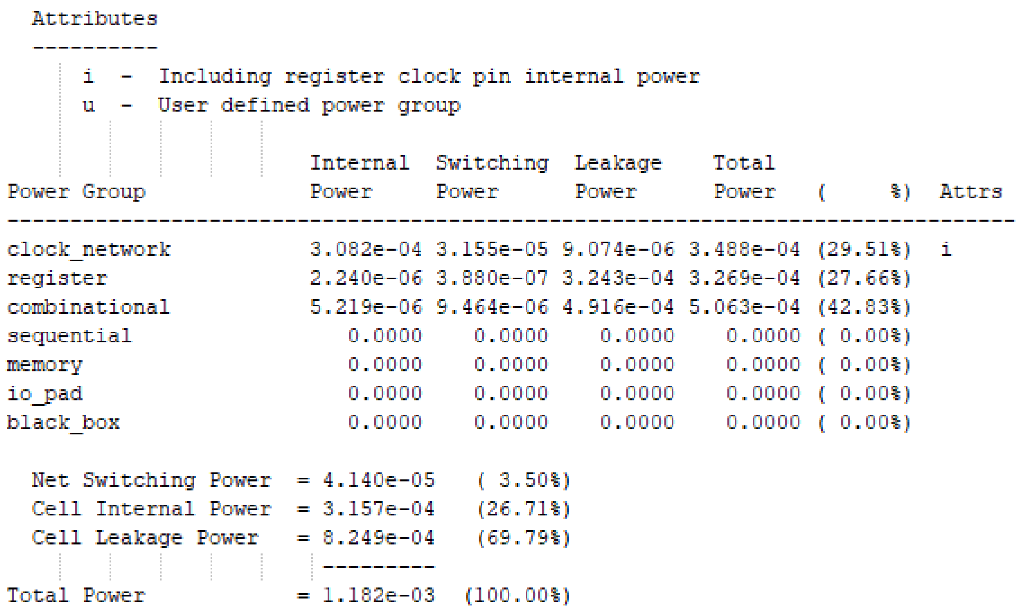


Figure 4.28: Total Area Distribution in Hierarchical Flow

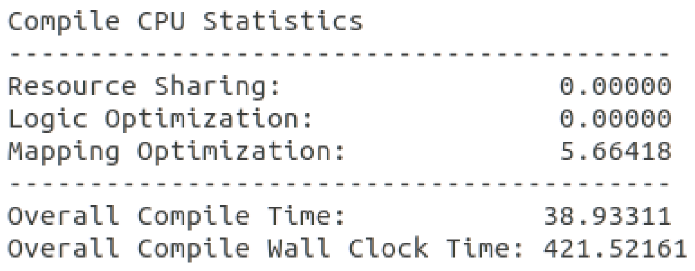
Power Results



```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           3.082e-04 3.155e-05 9.074e-06 3.488e-04 (29.51%)  i
register                2.240e-06 3.880e-07 3.243e-04 3.269e-04 (27.66%)
combinational           5.219e-06 9.464e-06 4.916e-04 5.063e-04 (42.83%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 4.140e-05   ( 3.50%)
  Cell Internal Power  = 3.157e-04   (26.71%)
  Cell Leakage Power   = 8.249e-04   (69.79%)
                         ---------
Total Power            = 1.182e-03  (100.00%)
```

Figure 4.29: Total Power Report in second-pass synthesis in Hierarchical Flow

Synthesis Runtime



```
Compile CPU Statistics
----------------------------------------
Resource Sharing:                0.00000
Logic Optimization:              0.00000
Mapping Optimization:            5.66418
----------------------------------------
Overall Compile Time:           38.93311
Overall Compile Wall Clock Time: 421.52161
```

Figure 4.30: Synthesis Runtime Report in Hierarchical Flow

4.2 PnR results

4.2.1 Flat Flow

Chip Floorplan

Core Utilization	0.601
Number of Rows	316
Core Width(micron)	264.328
Core Height(micron)	528.352
Aspect Ratio	1.999

Table 4.7: Chip Floorplan Table (Flat Flow)

From TABLE 4.7, core area is calculated to be 0.1397 mm^2. Number of rows is the number of rows in which standard cells are arranged, we choose an aspect ratio of 2 so that we can reduce horizontal routing congestion and increase the utilization of the chip.

Timing results

Maximum delay timing results:

Endpoint	**Arrival Time** (ns)	**Req Time** (ns)	**Slack)** (ns)
id_stage_i/alu_operand_b_ex_o_reg_20_/D	21.27	21.45	0.18
U3355/U2944_ex_stage_i_R_214/D	21.28	21.47	0.19
id_stage_i/alu_operand_b_ex_o_reg_30_/D	21.23	21.42	0.19
R_215/D	21.20	21.40	0.20
id_stage_i/mult_dot_op_b_ex_o_reg_20_/D	21.27	21.47	0.20
id_stage_i/mult_dot_op_b_ex_o_reg_30_/D	21.23	21.45	0.22
id_stage_i/alu_operand_b_ex_o_reg_11_/D	21.23	21.45	0.22
id_stage_i/alu_operand_b_ex_o_reg_27_/D	21.18	21.40	0.22
id_stage_i/mult_operand_b_ex_o_reg_11_/D	21.23	21.47	0.24
id_stage_i/mult_dot_op_b_ex_o_reg_11_/D	21.23	21.50	0.26

Table 4.8: WNS of 10 maximum delay timing paths after PnR (Flat Flow)

Table 4.8 illustrates the worst slack of the first 10 timing paths for setup time. There are latch-based timing paths in this design located ended at register file. Therefore, we have more than 1000 paths that stuck at 0 slack due to latches. So, we have to begin tracking our timing results after the last latch-based path so that we can determine the real quality of the timing optimization. In our design, we have gained a worst slack of 0.18 nanosecond at the first path, and a 0.26 nanosecond slack at the tenth path.

Figure 4.31 illustrates the number of timing paths against slack of the maximum delay timing based on their endpoints. As shown in the figure, the number of paths is almost equally distributed over the chart except the first column. This is because of latch-based timing paths which are typically stuck at 0 nanosecond, due to time borrowing phenomenon. Figure 1 also shows the worst and the best slack ever in the design. The worst slack is typically 0 as said before, and the best one is 19.33 nanosecond.

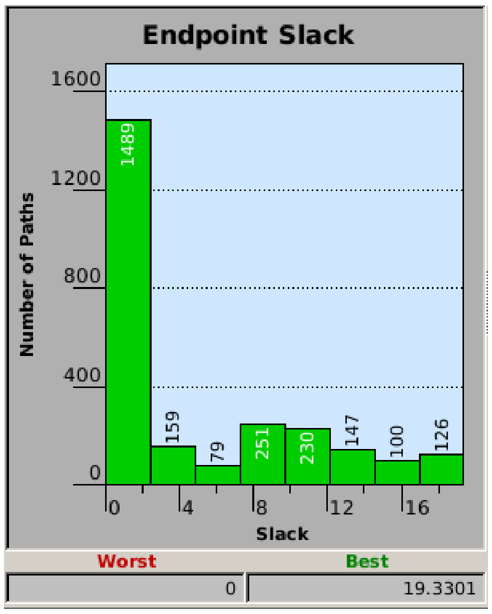


Figure 4.31: Max delay path slack histogram after PnR (Flat Flow)

Minimum delay timing results:

Endpoint	**Arrival Time(ns)**	**Req Time(ns)**	**Slack** (ns)
U3355/U2944_ex_stage_i_regfile_waddr_lsu_reg_2_/D	2.04	1.57	0.47
U3355/U2944_ex_stage_i_regfile_waddr_lsu_reg_3_/D	2.04	1.57	0.48
U3355/U2944_load_store_unit_i_data_sign_ext_q_reg_0_/D	2.04	1.56	0.48
U3355/U2944_ex_stage_i_regfile_waddr_lsu_reg_1_/D	2.06	1.56	0.50
U3355/U2944_ex_stage_i_regfile_waddr_lsu_reg_0_/D	2.06	1.56	0.50
U3355/cs_registers_i_PCCR_q_reg_0__0_/D	2.11	1.60	0.51
U3355/U2944_load_store_unit_i_data_we_q_reg/D	2.07	1.55	0.52
U3355/U2944_ex_stage_i_regfile_waddr_lsu_reg_4_/D	2.07	1.54	0.53
if_stage_i/prefetch_32_prefetch_buffer_i_fifo_i_rdata _Q_reg_3__16_/D	2.20	1.67	0.53
U3355/U2944_ex_stage_i_R_132/D	2.17	1.63	0.53

Table 4.9: WNS of 10 minimum delay timing paths after PnR (Flat Flow)

As shown in Table 4.9, The worst positive slack for hold time requirement is 0.47 nanosecond and the worst slack of the tenth one is 0.53.

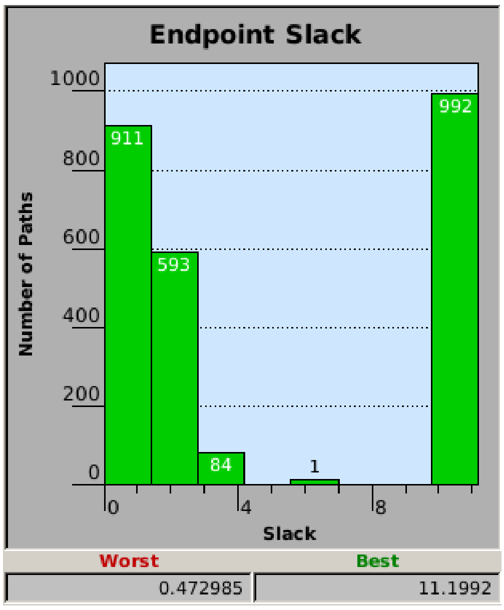


Figure 4.32: Min delay path slack histogram after PnR (Flat Flow)

In the latch-based timing paths, there is no hold time requirements. Therefore, all the timing paths illustrated in Figure 4.32 is pure flipflop-based paths. The figure also shows the worst and the best slack of all timing paths. Thus, the worst slack is 0.473 nanosecond and the best one is 11.199 nanosecond.

Core Area

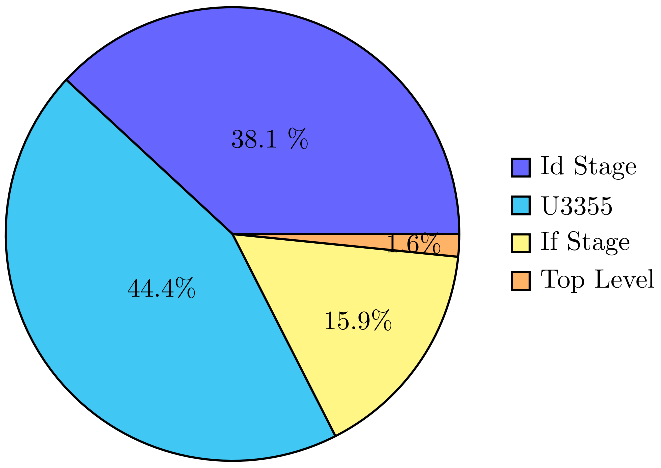


Figure 4.33: Cell area distribution after PnR (Flat Flow)

As shown in Figure 4.33, the dominant percentage of cell area lies in “U3355” unit as it contains all other flattened top-level blocks so that it contributes with about 44.4% of the entire cell area which is approximately “73594.766” microns square. The second dominant cell is the id stage which contains decoding logic and register file, so it is expected to contributed with such a ratio which is 38.1%. IF stage contributes with 15.9 %, since it contains only the prefetch buffering logic. The remaining percentage is dedicated for the rest of the top-level design logic like clock-gating cells.

Total Area

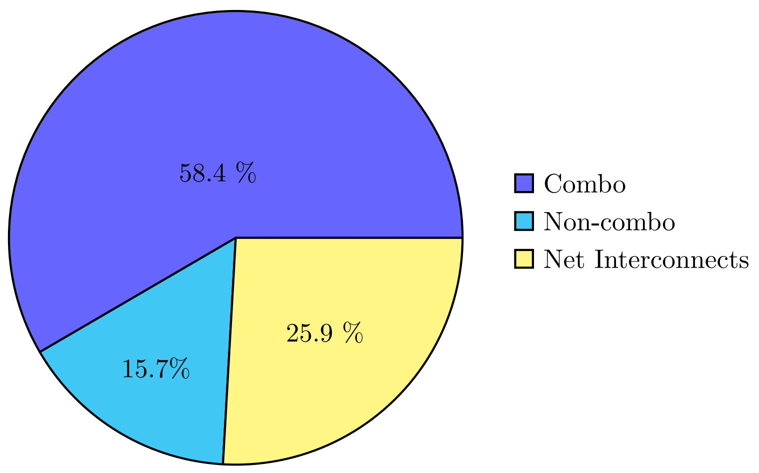


Figure 4.34: Total area distribution after PnR (Flat Flow)

Total area of the design consists of three main components: combinational area, non-combinational area, and net interconnect area as illustrated in Figure 4.34. Combinational logic contributed with 58.4% of the total area, and memory elements contribute with 15.7%, and the routing wires contribute with 25.9%.

Routing Congestion

Phase	Total (%)	Vertical (%)	Horizontal (%)
Floorplan	0.01	0.00	0.01
Placement	0.00	0.00	0.00
CTS	0.00	0.00	0.00
Routing	0.05	0.01	0.05
Chip Finish	0.06	0.01	0.06

Table 4.10: Routing congestion table after PnR (Flat Flow)

Table 4.10 exposes the percentage of routing congestion in both vertical and horizontal directions and their overall percentage with respect to the available global routing cells (GRCs). In floorplan stage, the congestion is 0.01% which indicates that the design is completely routable without obstacles, and after fine placement of the design, the congestion reduced to be 0%. The congestion remains 0% after performing the CTS stage, since the CTS was established in metal-8 and metal-9 layers. These two layers are independent on other routing layers. Therefore, routing in them does not affect the signal routing layers. After routing, the congestion increases to be 0.05% due to actual routing. In the last stage, the chip is completely filled with standard cells and metal fillers which contribute with a small factor on the total congestion.

CTS results

Clock Tree name	clk_i
Number of levels	19
Max Global Skew (ns)	0.930
Max Insertion Delay (ns)	2.463
Min Insertion Delay (ns)	1.533

Table 4.11: CTS table (Flat Flow)

Table 4.11 illustrates the main results of the clock tree network that was achieved in our design. We have achieved a maximum global skew of 0.93 nanosecond which is somehow acceptable. The maximum achieved propagation delay is 2.463 ns which is approximately 12.3% of the clock period.

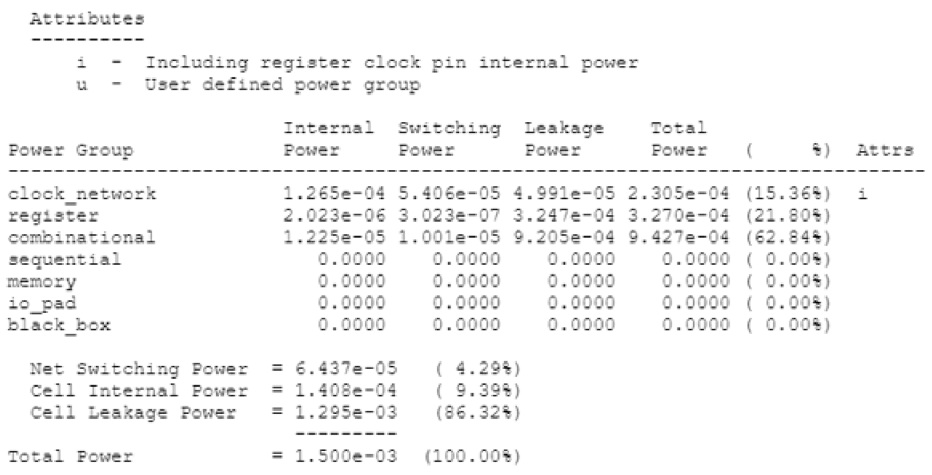
Wire Statistics

	Total	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**
Wire length(um)	796510	25969	263062	222856	157227	83751	5148	2866	23728	16296
Wire length(%)	100	3.26	33.026	27.979	19.739	10.5147	0.6463	0.3598	2.979	2.046

Table 4.12: Wire statistics table (Flat Flow)

Table 4.12 illustrates the wire length statistics over metal layers stack. The power network was purely built in layer 6 and layer 7, thus the power network contributes with a small percentage of the entire wire length. The CTS was constructed on layer 8 and layer 9. The remaining signal routes were inserted mainly on layers 2, 3 and 4 with little contribution of other layers.

Power Consumption



```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           1.265e-04 5.406e-05 4.991e-05 2.305e-04 (15.36%)  i
register                2.023e-06 3.023e-07 3.247e-04 3.270e-04 (21.80%)
combinational           1.225e-05 1.001e-05 9.205e-04 9.427e-04 (62.84%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 6.437e-05   ( 4.29%)
  Cell Internal Power  = 1.408e-04   ( 9.39%)
  Cell Leakage Power   = 1.295e-03   (86.32%)
                         ---------
Total Power            = 1.500e-03  (100.00%)
```

Figure 4.35: Total power report after PnR (Flat Flow)

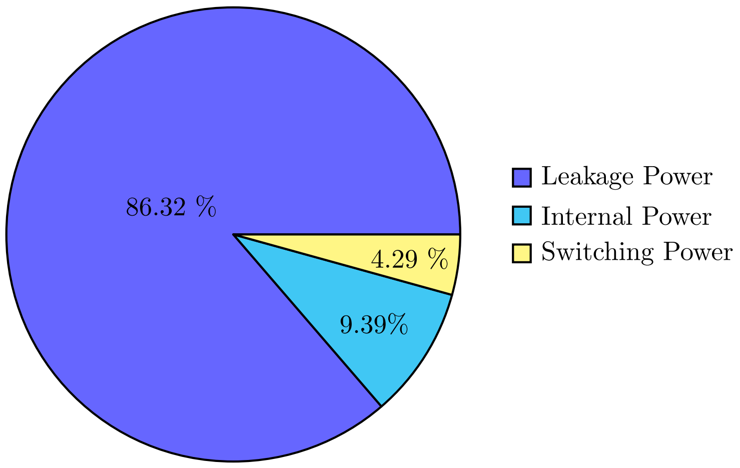


Figure 4.36: Total power distribution after PnR (Flat Flow)

As discussed before, when the technology steps forward, dynamic power consumption decreases and the leakage power increases. Figure 4.36 shows that the leakage power is the dominant component of the total power due to the advanced technology. Figure 4.41 shows a detailed report for power consumption in each part of the design.

4.2.2 Topographical Flow

In the next sections, we describe in more details the physical results of the processor core implementation.

Chip Floorplan

Core Utilization	0.701
Number of Rows	285
Core Width(micron)	238.488
Core Height(micron)	476.52
Aspect Ratio	1.998

Table 4.13: Chip Floorplan Table (Topographical Flow)

From TABLE 4.13, core area is calculated to be 0.113 mm^2. Number of rows is the number of rows in which standard cells are arranged, we chose an aspect ratio of 2 so that we can reduce horizontal routing congestion and increase the utilization of the chip.

Timing Results

Endpoint	**Arrival time**	**Req time**	**Slack**
id_stage_i/alu_operand_b_ex_o_reg_26_/D	25.44	25.39	-0.05
id_stage_i/alu_operand_b_ex_o_reg_24_/D	25.42	25.39	-0.03
id_stage_i/alu_operand_b_ex_o_reg_15_/D	25.27	25.25	-0.02
id_stage_i/alu_operand_b_ex_o_reg_27_/D	25.37	25.36	-0.01
id_stage_i/hwloop_regs_i/hwlp_counter_q_reg_1__25_/D	25.41	25.41	0.00
id_stage_i/alu_operand_b_ex_o_reg_31_/D	25.36	25.36	0.00
id_stage_i/mult_operand_b_ex_o_reg_26_/D	25.44	25.44	0.00
id_stage_i/alu_operand_a_ex_o_reg_0_/D	25.25	25.25	0.00
id_stage_i/alu_operand_b_ex_o_reg_18_/D	25.34	25.34	0.00
id_stage_i/alu_operand_b_ex_o_reg_23_/D	25.37	25.37	0.00
id_stage_i/alu_operand_b_ex_o_reg_29_/D	25.37	25.37	0.00
id_stage_i/alu_operand_a_ex_o_reg_29_/D	25.41	25.41	0.00

Table 4.14: Worst 12 setup timing paths after PnR (Topographical Flow)

TABLE 4.14 shows the most critical 10 paths after Place and Route is finished, we can see that the WNS equals to 50ps which is very small and can be resolved in Post-Layout STA verification as we will see in the next sections. Note that time unit used is ”ns”.

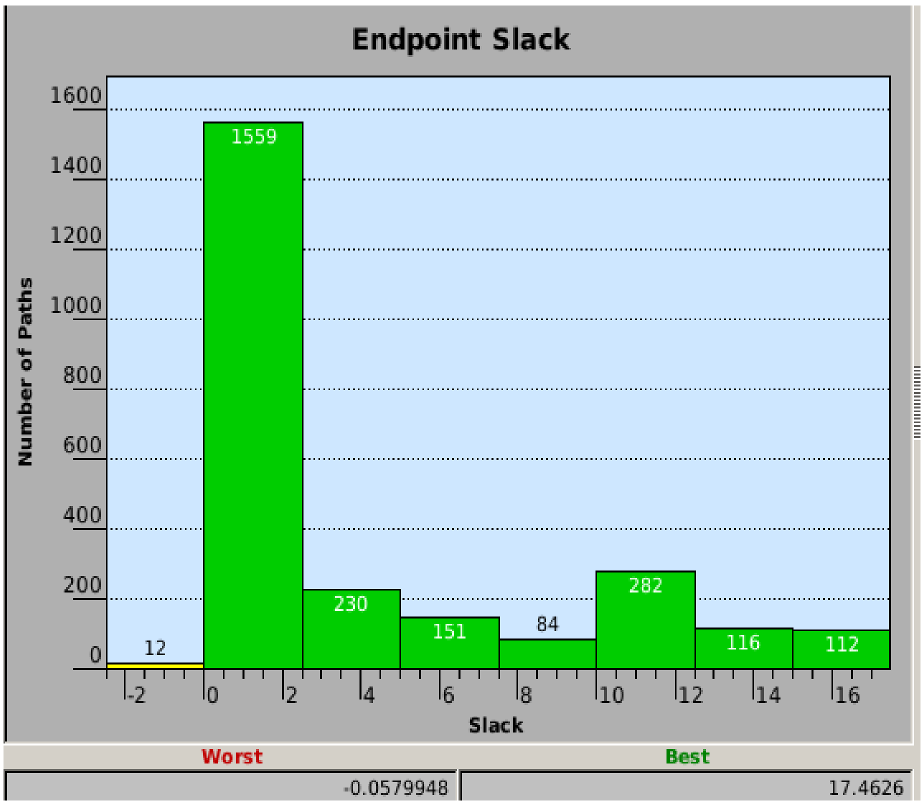


Figure 4.37: Setup timing histogram after PnR (Topographical Flow)

from FIGURE 4.37 we can see that there are 12 timing paths which are below 0 shown in more details in TABLE 4.14. Also we can observe the peak number of paths near to 0 slack which is back to the concept of time borrowing discussed in previous sections, which is related to latch-based timing paths in our design.

Endpoint	Arrival time(ns)	Req time(ns)	Slack(ns)
id_stage_i/hwloop_regs_i/hwlp_counter_q_reg_0__31_/D	3.76	2.99	0.77
id_stage_i/hwloop_regs_i/hwlp_counter_q_reg_0__31_/D	3.77	2.99	0.78
id_stage_i/hwloop_regs_i/hwlp_counter_q_reg_1__31_/D	3.78	3.00	0.79
id_stage_i/hwloop_regs_i/hwlp_counter_q_reg_0__31_/D	3.79	2.99	0.81
if_stage_i/prefetch_32_prefetch_buffer_i/fifo_i/rdata_Q_reg_1__26_/D	4.48	3.42	1.06
if_stage_i/prefetch_32_prefetch_buffer_i/fifo_i/rdata_Q_reg_1__19_/D	4.48	3.42	1.06
if_stage_i/prefetch_32_prefetch_buffer_i/fifo_i/rdata_Q_reg_0__11_/D	4.51	3.43	1.08
id_stage_i/registers_i/riscv_register_file_i/wdata_b_q_reg_22_/D	4.31	3.18	1.13
if_stage_i/prefetch_32_prefetch_buffer_i/fifo_i/rdata_Q_reg_0__11_/D	4.49	3.36	1.13
id_stage_i/int_controller_i/irq_id_q_reg_1_/D	4.19	3.06	1.13
id_stage_i/int_controller_i/irq_id_q_reg_2_/D	4.19	3.06	1.13
id_stage_i/int_controller_i/irq_id_q_reg_3_/D	4.19	3.06	1.13

Table 4.15: Worst 10 hold timing paths after PnR (Topographical Flow)

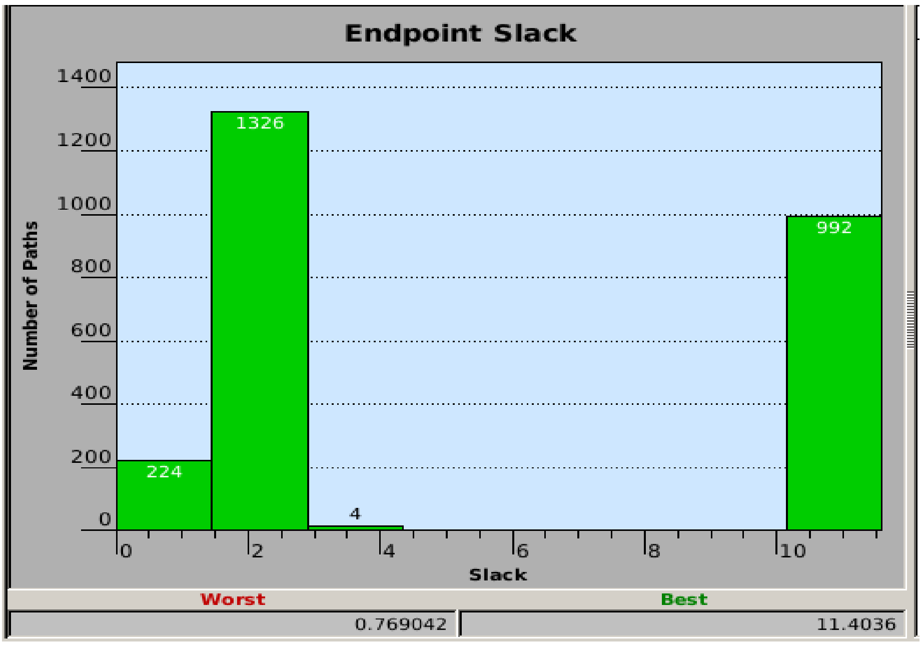


Figure 4.38: Hold timing histogram after PnR (Topographical Flow)

Core Area

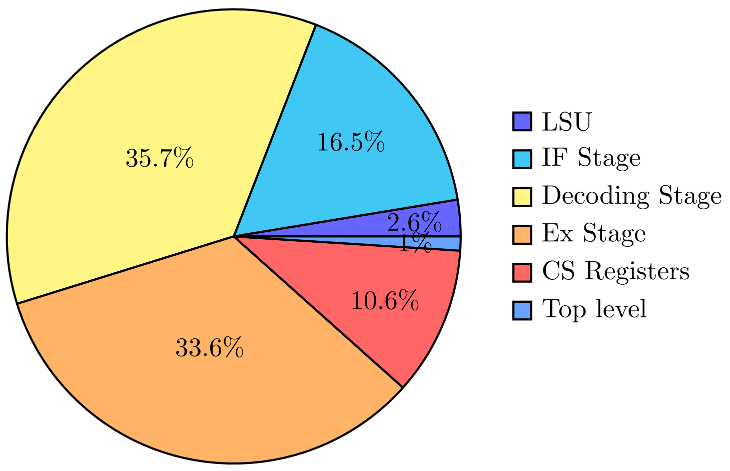


Figure 4.39: Cell Area Distribution across hierarchy after PnR (Topographical Flow)

FIGURE 4.39 shows the cell area percentage for each block out of the total cell area, we found that both pipeline decoding stage and execution stage are two major blocks with respect to cell area and together they represent more than half of the total cell area.

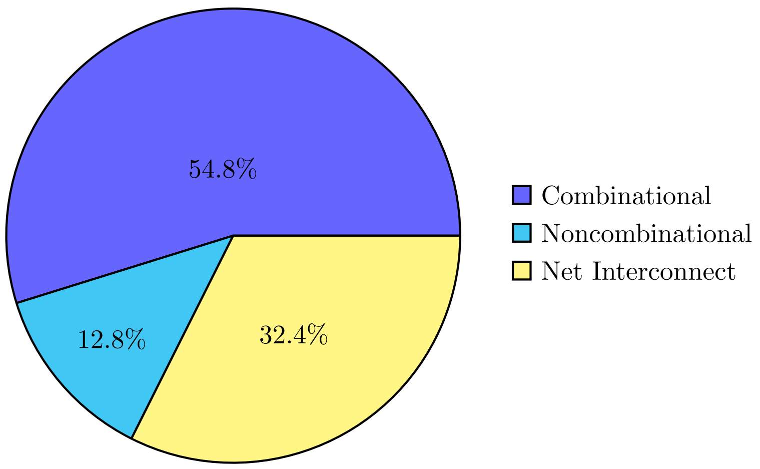


Figure 4.40: Total Area Distribution after PnR (Topographical Flow)

In FIGURE 4.40 , we can see the total area of the chip classified into three main groups. Combinational, Noncombinational and interconnect area. Combinational area occupies more than half of the total area of the chip where interconnect area comes next to it.

Routing Congestion

Phase	**Total (%)**	**Vertical(%)**	**Horizontal(%)**
Floorplan	0.00	0.00	0.00
Placement	0.03	0.00	0.03
CTS	0.52	0.03	0.49
Routing	1.18	0.02	1.16
Chip Finish	1.20	0.02	1.18

Table 4.16: Routing Congestion table after PnR (Topographical Flow)

TABLE 4.16 refers to the routing congestion percentage after each PnR step. It is clear that congestion has a remarkable increase especially after CTS, which is reasonable as new cells are inserted for clock tree synthesis which increases routing congestion, However, congestion is still acceptable over all phases.

CTS Results

Clock Tree Name	**clk_i**
Number of levels	**26**
Max Global Skew(ns)	**0.406**
Max insertion delay(ns)	**2.1**
Min insertion delay(ns)	**1.694**

Table 4.17: CTS table (Topographical Flow)

From TABLE 4.17 we can see that global skew is less than the clock uncertainty that is assumed at synthesis step, which is acceptable amount of clock skew, also we can see that max insertion delay is nearly 10% of the clock period.

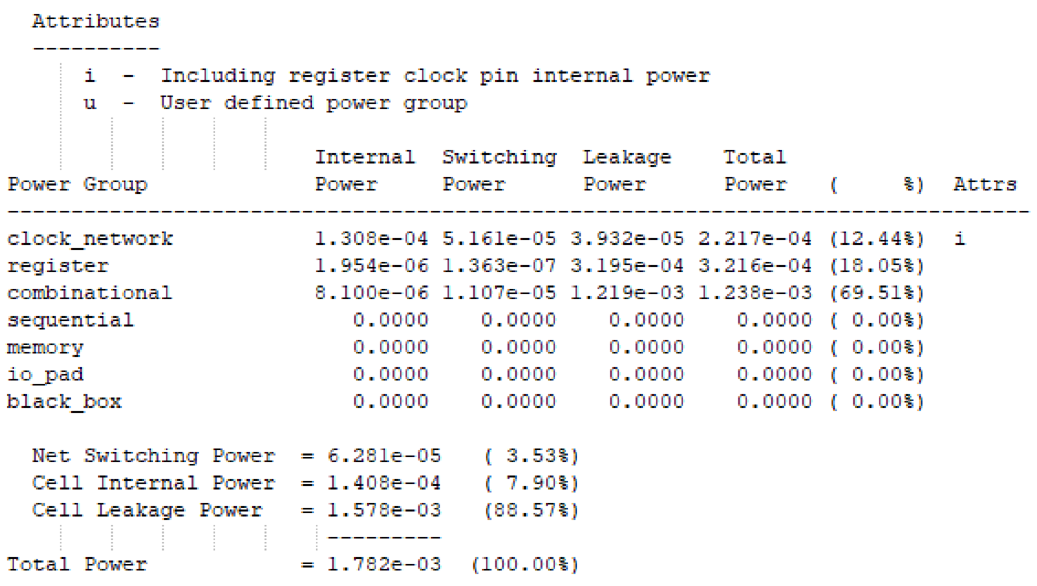
Wire Statistics

	Total	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**
Wire Length(um)	722595	10804	317348	218364	45384	13800	51836	42799	22259
Wire Length(%)	100	1.495	43.917	30.219	6.28	1.909	7.173	5.922	3.08

Table 4.18: Wire Statistics table (Topographical Flow)

TABLE 4.18 shows design routing statistics. We can see that design routing is mostly done using both Metal2 and Metal3 , this is because they were reserved for signals routing. Metal4 and Metal5 were reserved for power straps, Metal6-Metal8 are used for clock network routing and Metal1 is mostly used for power rails and internal connections of standard cells.

Power



```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage   Total
Power Group             Power     Power      Power     Power    (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           1.308e-04 5.161e-05 3.932e-05 2.217e-04 (12.44%)  i
register                1.954e-06 1.363e-07 3.195e-04 3.216e-04 (18.05%)
combinational           8.100e-06 1.107e-05 1.219e-03 1.238e-03 (69.51%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 6.281e-05   ( 3.53%)
  Cell Internal Power  = 1.408e-04   ( 7.90%)
  Cell Leakage Power   = 1.578e-03   (88.57%)
                         ---------
Total Power            = 1.782e-03  (100.00%)
```

Figure 4.41: Total Power report after PnR (Topographical Flow)

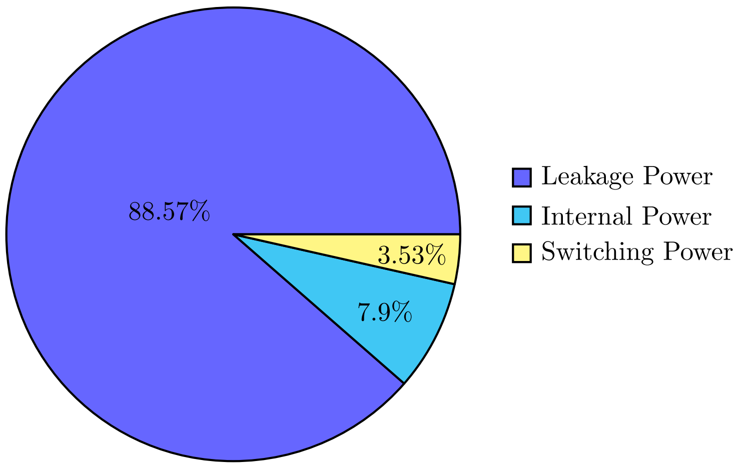
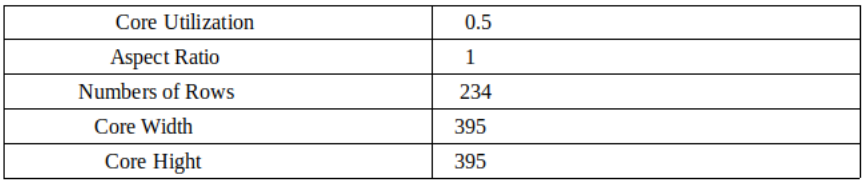


Figure 4.42: Total Power Distribution after PnR (Topographical Flow)

FIGURE 4.41 shows design power statistics. Total power is 1.78 mW calculated at a 10% toggle rate which is the default toggling rate available, we can notice that leakage power specifically dominates the amount of total power. Such leakage power can be reduced with switching from RVT cells to HVT cells as an example but with the penalty of slower cells and larger delays.

4.2.3 Hierarchical Flow

Chip Floorplan



Core Utilization	0.5
Aspect Ratio	1
Numbers of Rows	234
Core Width	395
Core Hight	395

Figure 4.43: Chip Floorplan Table in Hierarchical Flow

Timing Results

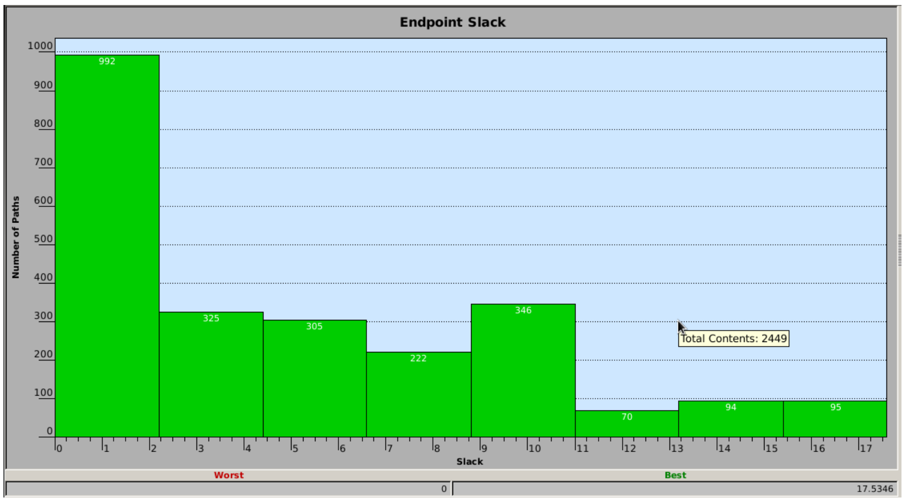


Figure 4.44: Setup timing histogram in Hierarchical Flow

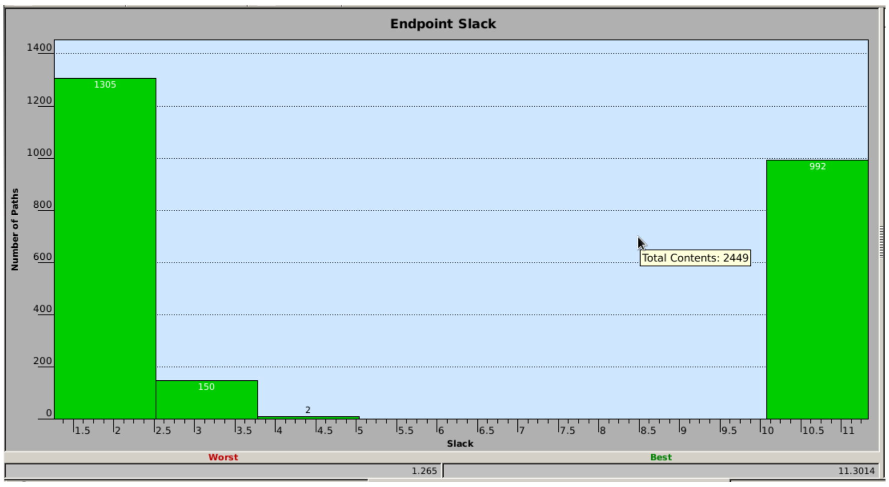
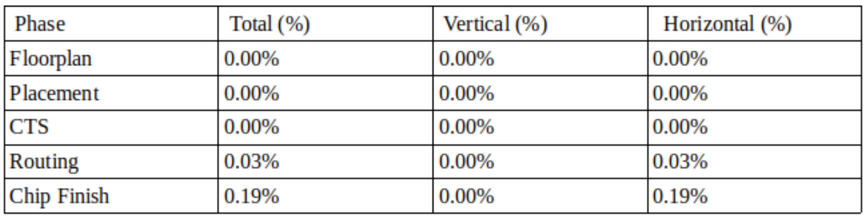


Figure 4.45: Hold timing histogram in Hierarchical Flow

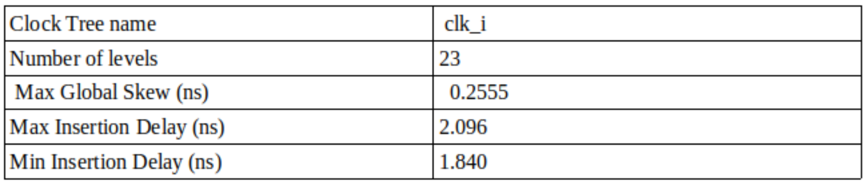
Routing Congestion



Phase	Total (%)	Vertical (%)	Horizontal (%)
Floorplan	0.00%	0.00%	0.00%
Placement	0.00%	0.00%	0.00%
CTS	0.00%	0.00%	0.00%
Routing	0.03%	0.00%	0.03%
Chip Finish	0.19%	0.00%	0.19%

Figure 4.46: Routing Congestion in Hierarchical Flow

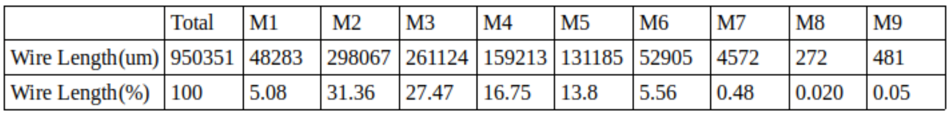
CTS Results



Clock Tree name	clk_i
Number of levels	23
Max Global Skew (ns)	0.2555
Max Insertion Delay (ns)	2.096
Min Insertion Delay (ns)	1.840

Figure 4.47: CTS Results in Hierarchical Flow

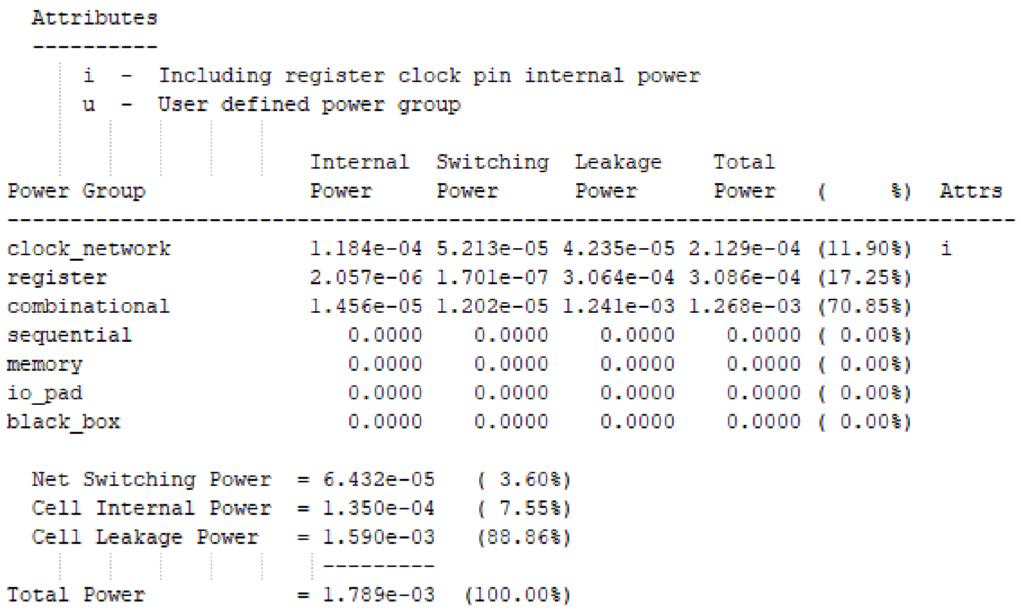
Wire Statistics



	Total	M1	M2	M3	M4	M5	M6	M7	M8	M9
Wire Length(um)	950351	48283	298067	261124	159213	131185	52905	4572	272	481
Wire Length(%)	100	5.08	31.36	27.47	16.75	13.8	5.56	0.48	0.020	0.05

Figure 4.48: Wire Statistics in Hierarchical Flow

Power



```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power    (      %)  Attrs
--------------------------------------------------------------------------------
clock_network           1.184e-04 5.213e-05 4.235e-05 2.129e-04 (11.90%)  i
register                2.057e-06 1.701e-07 3.064e-04 3.086e-04 (17.25%)
combinational           1.456e-05 1.202e-05 1.241e-03 1.268e-03 (70.85%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 6.432e-05   ( 3.60%)
  Cell Internal Power  = 1.350e-04   ( 7.55%)
  Cell Leakage Power   = 1.590e-03   (88.86%)
                         ---------
Total Power            = 1.789e-03  (100.00%)
```

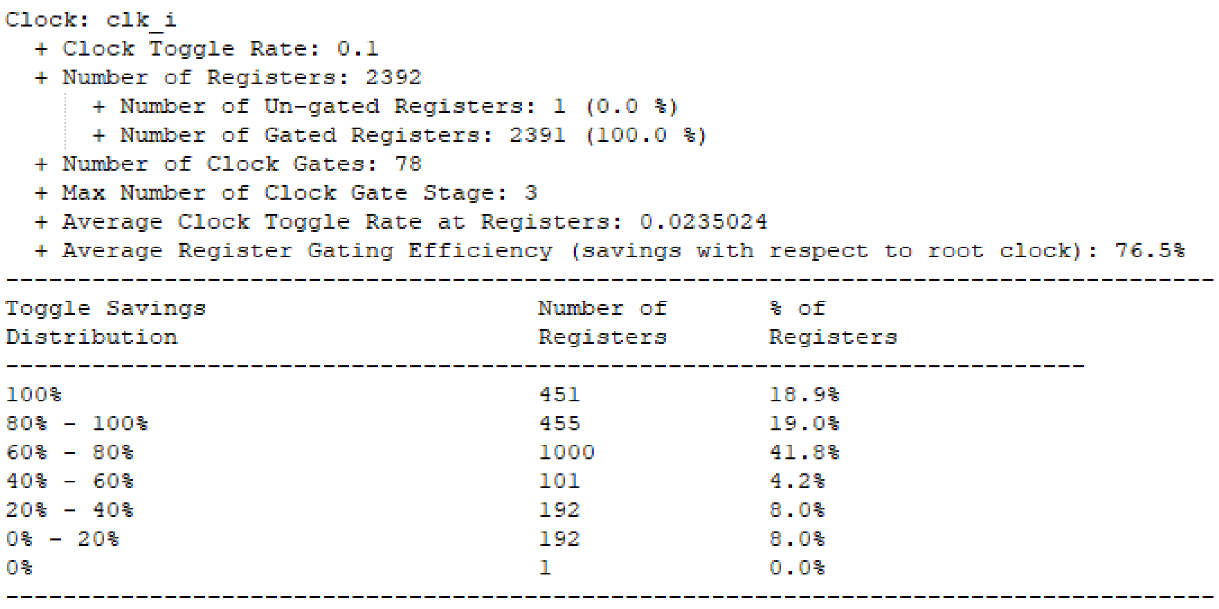
Figure 4.49: Total Power Report in Hierarchical Flow

4.3 STA results

4.3.1 Flat Flow

The design timing was tested using static timing analysis(STA) in primetime. In STA you define the design constraints and then test the design and see if it passes them. These constraints are the same constraints used to build up the design in addition to the parasitics come from layout. We test the design for 22 case and show how to make the design passes without timing violations for these cases.

Clock gating savings



```
Clock: clk_i
  + Clock Toggle Rate: 0.1
  + Number of Registers: 2392
      + Number of Un-gated Registers: 1 (0.0 %)
      + Number of Gated Registers: 2391 (100.0 %)
  + Number of Clock Gates: 78
  + Max Number of Clock Gate Stage: 3
  + Average Clock Toggle Rate at Registers: 0.0235024
  + Average Register Gating Efficiency (savings with respect to root clock): 76.5%
-------------------------------------------------------------------------------------
Toggle Savings                       Number of       % of
Distribution                         Registers       Registers
----------------------------------------------------------------------------
100%                                 451             18.9%
80% - 100%                           455             19.0%
60% - 80%                            1000            41.8%
40% - 60%                            101             4.2%
20% - 40%                            192             8.0%
0% - 20%                             192             8.0%
0%                                   1               0.0%
-------------------------------------------------------------------------------------
```

Figure 4.50: clock gating report (Flat Flow)

Before we go through STA results of the previous mentioned corners, we review the effciency of clock gating in the design. Clock gating savings report shown in Figure 4.50 summarizes the overall effectiveness of clock gating in the design, the toggle savings and gating effciency are computed based on 10% clock toggling rate. Average Register Gating effciency is 76.5% (savings with respect to root clock) , toggle saving reaches 60% to 80% for about half the sequential elements in the design which is good, since 60% to 80% toggles are suppressed by clock gating logic which highly reduces dynamic power in the design. Toggle saving is by de nition the ratio between toggle rate at clock pin of registers to the toggle rate at clock root pin, toggle rate is represented as a percentage. Toggle rate of a sequential element means the rate at which output pin switches relative to the clock input frequency, toggle rate ranges between 0%-200%. Toggle rate of 100% as an example means that frequency of the output pin is half the frequency of the clock input. This convention comes from the assumption that the output changes every rising edge or a falling edge of the clock cycle.

Review of cases

we will show how to make the design passes all 22 case with OCV 1.1 derate of setup check and 0.9 derate of hold check except the worst case ss0p7v125c with 1.05 derate of setup check.

notes:

how to read case's name ?

ss means slow-slow case.

tt means typical-typical case.

ff means fast-fast case.

and the next part of name corresponds to operating voltage and operating temberature.

example:

ss0p7v125c means slow-slow case operating at 0.7voltage and 125°C.

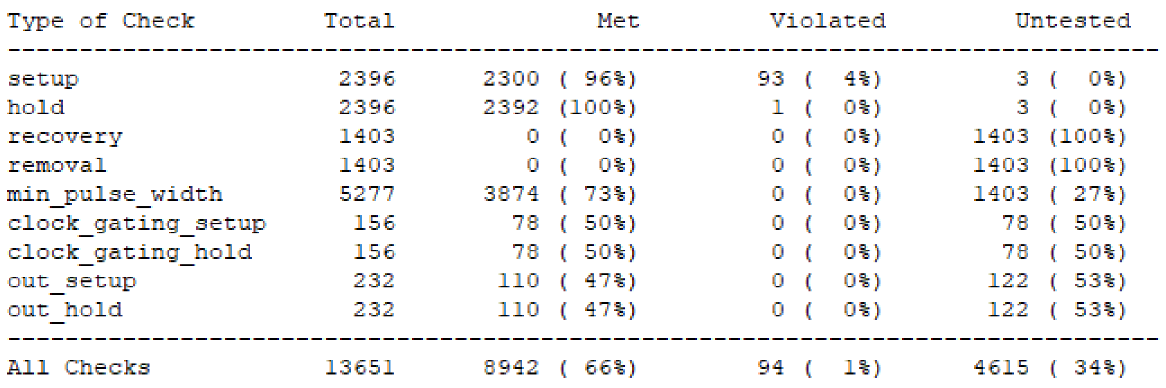
case	number of SETUP violationed pathes	number of HOLD violationed pathes
ss0p7v125c	93	1
ss0p75v125c	0	2
ss0p95v25c	0	3
ss0p95v125c	0	3
ss0p95vn40c	0	3
tt0p78v25c	0	0
tt0p78v125c	0	3
tt0p78vn40c	0	2
tt0p85v25c	0	3
tt0p85v125c	0	3
tt0p85vn40c	0	2
tt1p05v25c	0	46
tt1p05v125c	0	3
tt1p05vn40c	0	3
ff0p85v25c	0	3
ff0p85v125c	0	34
ff0p85vn40c	0	3
ff0p95v25c	0	3
ff0p95v125c	0	3
ff0p95vn40c	0	3
ff1p16v25c	0	3
ff1p16vn40c	0	3

Table 4.19: Table of 22 cases before ECO fixing time (Flat Flow)

As shown in Table 4.19 the corner cases are ss0p7v125c ,tt1p05v25c ,and ff0p85v125c.
From this point we start to solve setup time violations with upsizing technique and solve hold time violations with inserting buffers technique.

More details about corner cases:

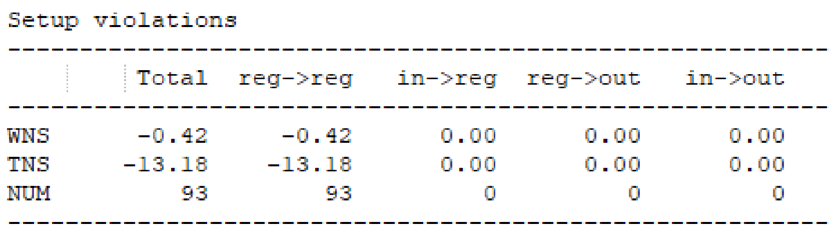
1- slow-slow at 0.7v and 125°C



```
Type of Check          Total             Met          Violated         Untested
--------------------------------------------------------------------------------
setup                   2396      2300 ( 96%)        93 (   4%)        3 (   0%)
hold                    2396      2392 (100%)         1 (   0%)        3 (   0%)
recovery                1403         0 (   0%)        0 (   0%)     1403 (100%)
removal                 1403         0 (   0%)        0 (   0%)     1403 (100%)
min_pulse_width         5277      3874 ( 73%)         0 (   0%)     1403 ( 27%)
clock_gating_setup       156        78 ( 50%)         0 (   0%)       78 ( 50%)
clock_gating_hold        156        78 ( 50%)         0 (   0%)       78 ( 50%)
out_setup                232       110 ( 47%)         0 (   0%)      122 ( 53%)
out_hold                 232       110 ( 47%)         0 (   0%)      122 ( 53%)
--------------------------------------------------------------------------------
All Checks             13651      8942 ( 66%)        94 (   1%)     4615 ( 34%)
```

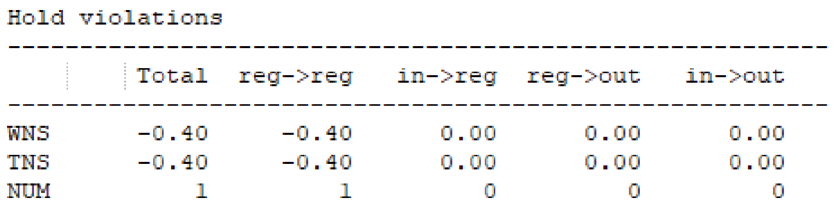
Figure 4.51: Analysis coverage report of Slow-Slow (Flat Flow)

Figure 4.51 shows the initial checks on the design that shows the percentage of met, violated, and untested pathes for each check. For example for setup check there is 2300 path are met that is their percentage of all pathes is 96% and there is 93 violated path that is 4% of all pathes and there is 3 untested path that is their percentage of all pathes almost 0% and so on for all checks. The resons of untested pathes are false pathes such as Reset path and scan enable paths in clock gating these pathes are similar to Reset pathes and also another reson of untested path is that there is no startpoint and endpoint for pathes related to APU block that controls FPU operations in the design. While FPU is removed as mentioned in earlier chapters, APU is also not implemented which makes its related output ports grounded by default.



```
Setup violations
---------------------------------------------------------
            Total  reg->reg   in->reg  reg->out   in->out
---------------------------------------------------------
WNS         -0.42     -0.42      0.00      0.00      0.00
TNS        -13.18    -13.18      0.00      0.00      0.00
NUM            93        93         0         0         0
---------------------------------------------------------
```

Figure 4.52: Global timing report (SETUP) of Slow-Slow (Flat Flow)



```
Hold violations
----------------------------------------------------------
        | Total  reg->reg   in->reg  reg->out   in->out
----------------------------------------------------------
WNS      -0.40     -0.40      0.00      0.00      0.00
TNS      -0.40     -0.40      0.00      0.00      0.00
NUM          1         1         0         0         0
```

Figure 4.53: Global timing report (HOLD) of Slow-Slow (Flat Flow)

Figure 4.52 and Figure 4.53 show a global overview of timing violations before fixing these violations.
we start to solve thes violations by upsizing for setup violations and insertting buffers for hold violations and back again to layout tool and add these changes on the design and then extract new physical results and back to STA tool and repeat again to make the design pass without violations in this case. The following figures slow results after solving violations.

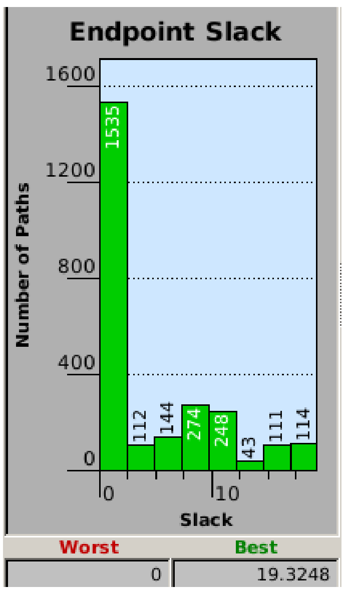


Figure 4.54: Setup slack histogram of Slow-Slow (Flat Flow)

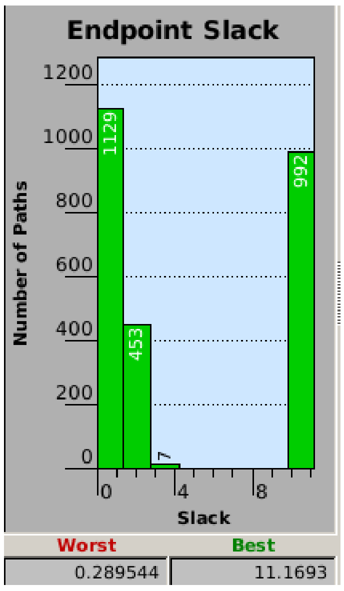


Figure 4.55: Hold slack histogram of Slow-Slow (Flat Flow)

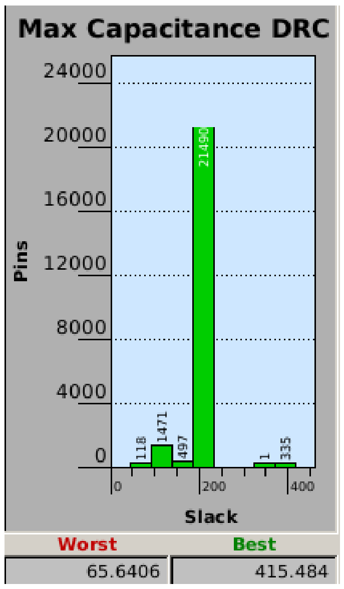


Figure 4.56: Max capacitance histogram of Slow-Slow (Flat Flow)

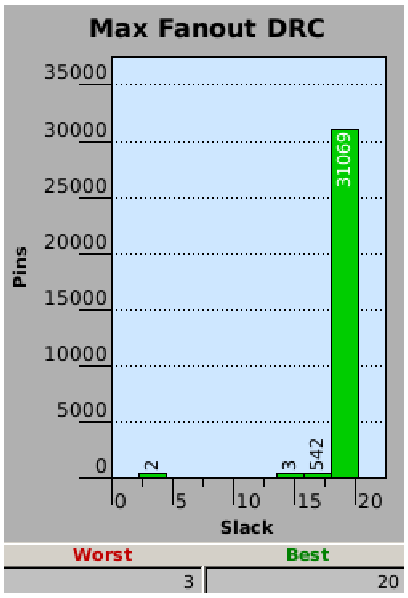


Figure 4.57: Max fanout histogram of Slow-Slow (Flat Flow)

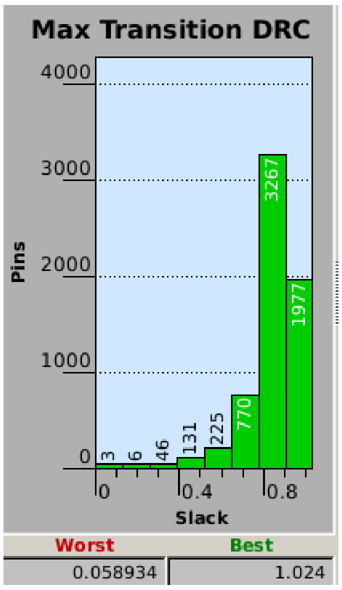
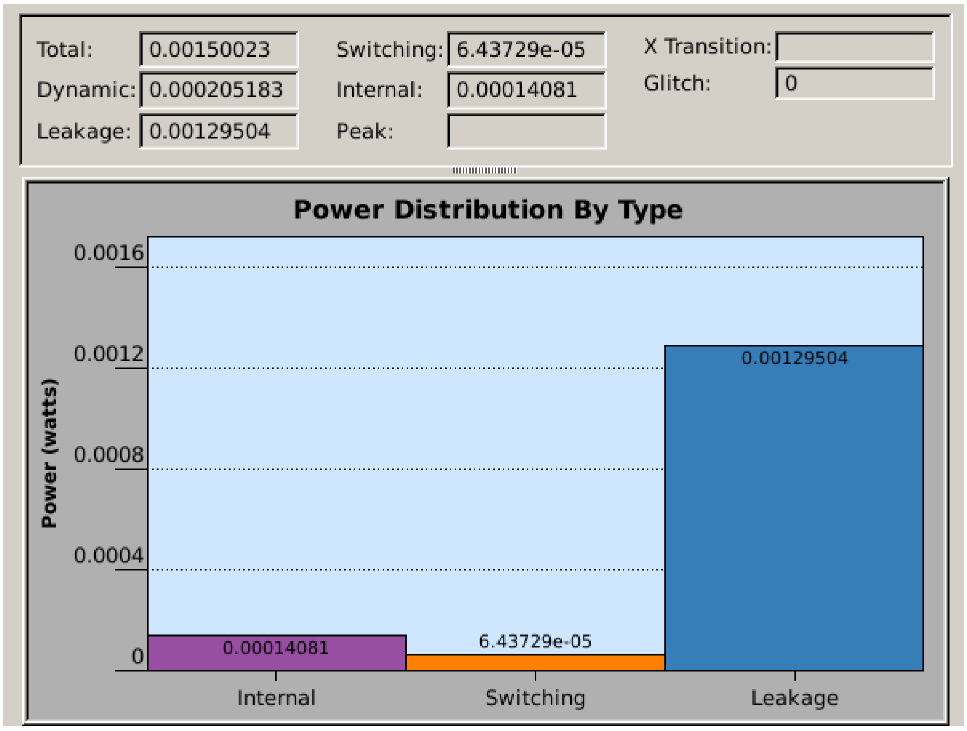


Figure 4.58: Max transation histogram of Slow-Slow (Flat Flow)

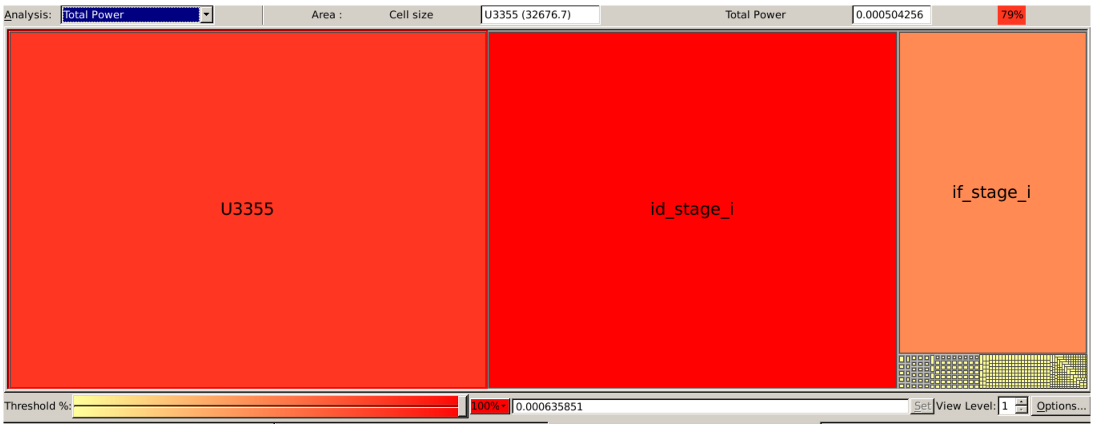
```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           1.265e-04 5.406e-05 4.991e-05 2.305e-04 (15.36%)  i
register                2.023e-06 3.023e-07 3.247e-04 3.270e-04 (21.80%)
combinational           1.225e-05 1.001e-05 9.205e-04 9.427e-04 (62.84%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 6.437e-05   ( 4.29%)
  Cell Internal Power  = 1.408e-04   ( 9.39%)
  Cell Leakage Power   = 1.295e-03   (86.32%)
                         ---------
Total Power            = 1.500e-03  (100.00%)
```

Figure 4.59: power report of Slow-Slow (Flat Flow)



Total:	0.00150023	Switching:	6.43729e-05	X Transition:	
Dynamic:	0.000205183	Internal:	0.00014081	Glitch:	0
Leakage:	0.00129504	Peak:			

Figure 4.60: power histogram of Slow-Slow (Flat Flow)



Analysis: Total Power   Area : Cell size U3355 (32676.7)   Total Power 0.000504256 79%

Threshold %: 100% 0.000635851   View Level: 1

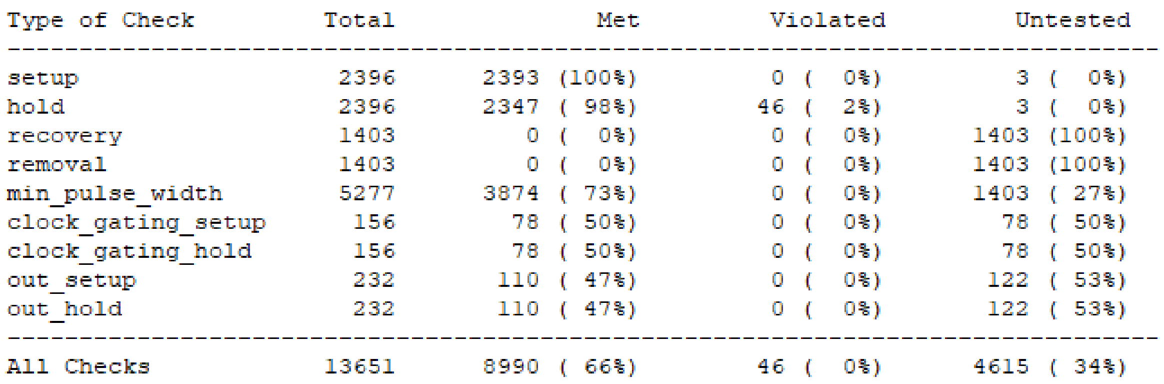
Figure 4.61: power map of the design of Slow-Slow (Flat Flow)

Figure 4.61 shows power distribution across hierarchy. It represents the percentage of power with respect to block that dissipate mostly power with respect to total power.

Block Name	Power relative to threshold(W)	Power relative to threshold(%)
U3355	0.00063581	100
if_stage_i	0.000295003	46
id_stage_i	0.000504256	79

Table 4.20: Power Design Map Statistics of Slow-Slow (Flat Flow)

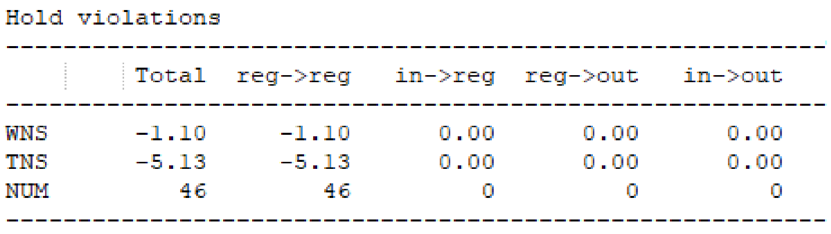
2- typical-typical at 1.05v and 25°C



```
Type of Check          Total             Met         Violated          Untested
--------------------------------------------------------------------------------
setup                   2396     2393 (100%)         0 (   0%)        3 (   0%)
hold                    2396     2347 ( 98%)        46 (   2%)        3 (   0%)
recovery                1403        0 (   0%)        0 (   0%)     1403 (100%)
removal                 1403        0 (   0%)        0 (   0%)     1403 (100%)
min_pulse_width         5277     3874 ( 73%)         0 (   0%)     1403 ( 27%)
clock_gating_setup       156       78 ( 50%)         0 (   0%)       78 ( 50%)
clock_gating_hold        156       78 ( 50%)         0 (   0%)       78 ( 50%)
out_setup                232      110 ( 47%)         0 (   0%)      122 ( 53%)
out_hold                 232      110 ( 47%)         0 (   0%)      122 ( 53%)
--------------------------------------------------------------------------------
All Checks             13651     8990 ( 66%)        46 (   0%)     4615 ( 34%)
```

Figure 4.62: Analysis coverage report of Typ-Typ (Flat Flow)

Figure 4.62 shows the initial checks on the design that shows the percentage of met, violated, and untested pathes for each check. For example for hold check there is 2347 path are met that is their percentage of all pathes is 98% and there is 46 violated path that is 2% of all pathes and there is 3 untested path that is their percentage of all pathes almost 0% and so on for all checks. The resons of untested pathes are false pathes such as Reset path and scan enable paths in clock gating these pathes are similar to Reset pathes and also another reson of untested path is that there is no startpoint and endpoint for pathes related to APU block that controls FPU operations in the design. While FPU is removed as mentioned in earlier chapters, APU is also not implemented which makes its related output ports grounded by default.



```
Hold violations
----------------------------------------------------------
          Total   reg->reg    in->reg   reg->out    in->out
----------------------------------------------------------
WNS       -1.10      -1.10       0.00       0.00       0.00
TNS       -5.13      -5.13       0.00       0.00       0.00
NUM          46         46          0          0          0
----------------------------------------------------------
```

Figure 4.63: Global timing report (HOLD) of Typ-Typ (Flat Flow)

Figure Figure 4.63 shows a global overview of timing violations before fixing these violations. As cleared there is no setup violations only thre is hold time violations that can be solved by insertting buffers.

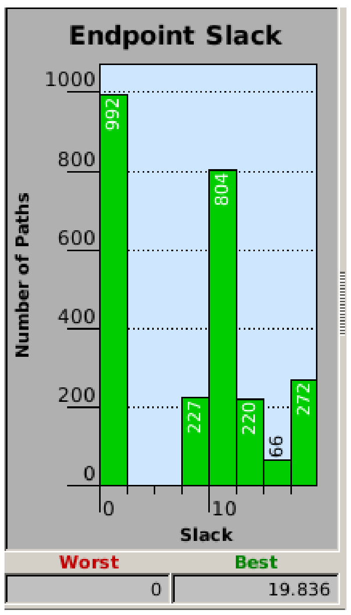


Figure 4.64: Setup slack histogram of Typ-Typ (Flat Flow)

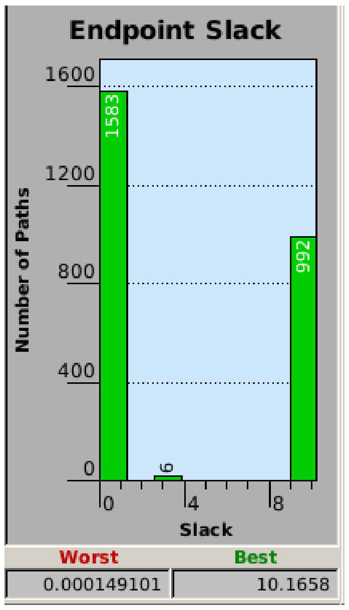


Figure 4.65: Hold slack histogram of Typ-Typ (Flat Flow)

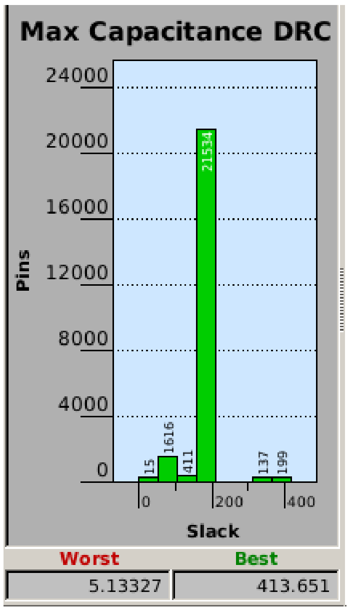


Figure 4.66: Max capacitance histogram of Typ-Typ (Flat Flow)

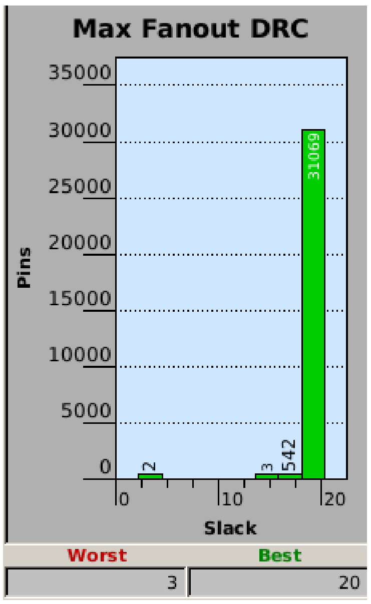


Figure 4.67: Max fanout histogram of Typ-Typ (Flat Flow)

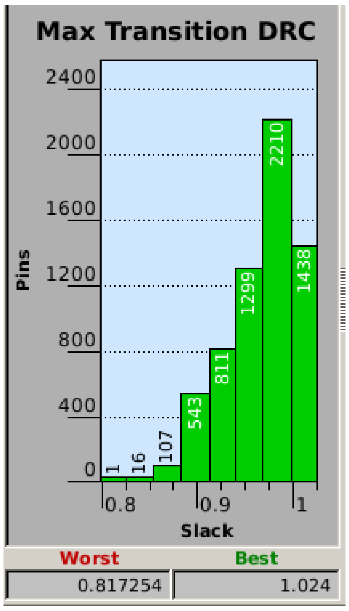
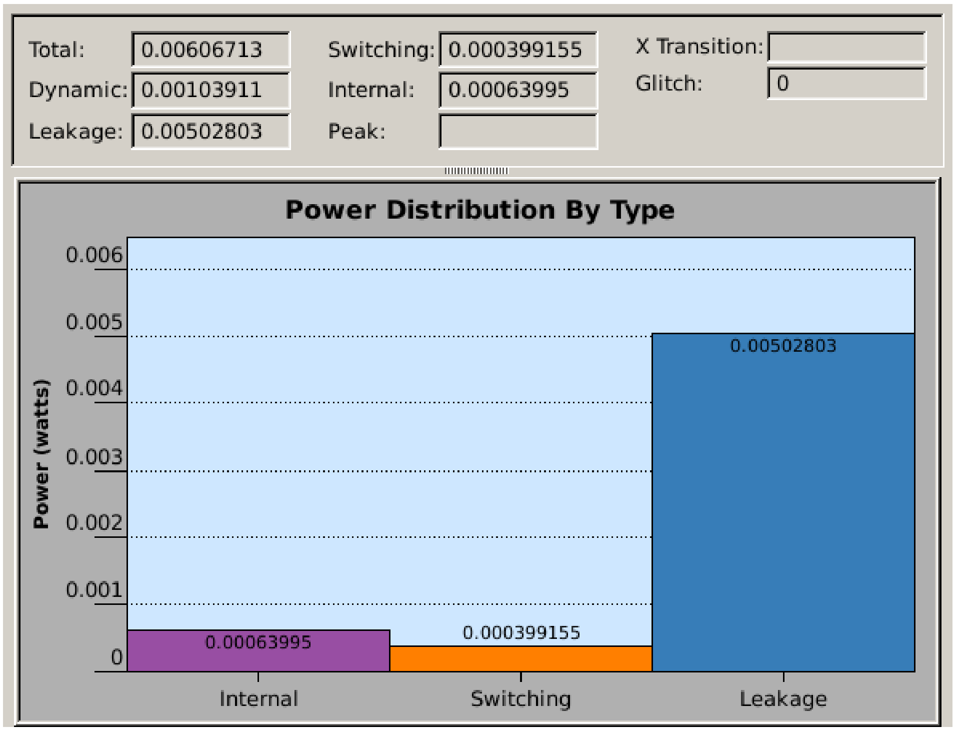


Figure 4.68: Max transation histogram of Typ-Typ (Flat Flow)

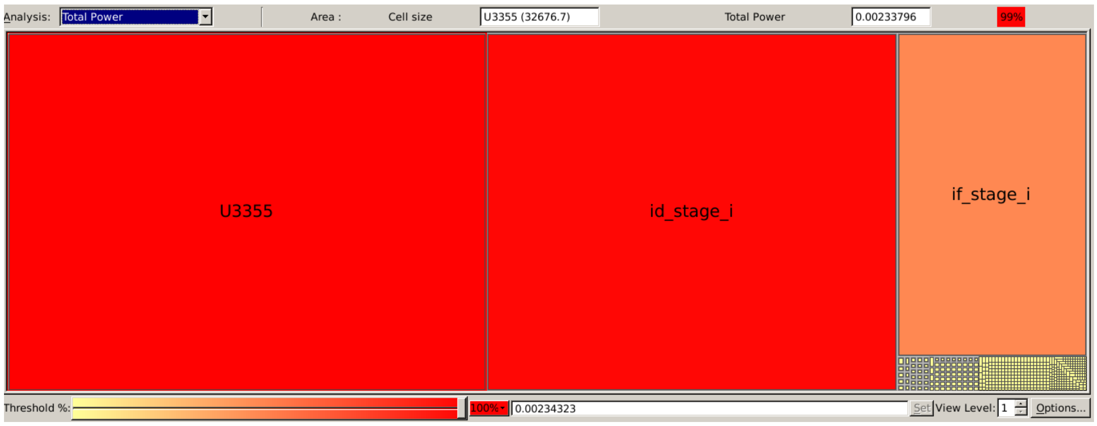
```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power    (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           5.990e-04 3.430e-04 2.000e-04 1.142e-03 (18.82%)  i
register                5.551e-06 1.755e-06 9.684e-04 9.757e-04 (16.08%)
combinational           3.541e-05 5.443e-05 3.860e-03 3.949e-03 (65.10%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 3.992e-04   ( 6.58%)
  Cell Internal Power  = 6.400e-04   (10.55%)
  Cell Leakage Power   = 5.028e-03   (82.87%)
                         ---------
Total Power            = 6.067e-03  (100.00%)
```

Figure 4.69: power report of Typ-Typ (Flat Flow)



Total:	0.00606713	Switching:	0.000399155	X Transition:	
Dynamic:	0.00103911	Internal:	0.00063995	Glitch:	0
Leakage:	0.00502803	Peak:			

Figure 4.70: power histogram of Typ-Typ (Flat Flow)


Analysis: Total Power

Area : Cell size U3355 (32676.7) Total Power 0.00233796 99%

Threshold %: 100% 0.00234323 Set View Level: 1 Options...

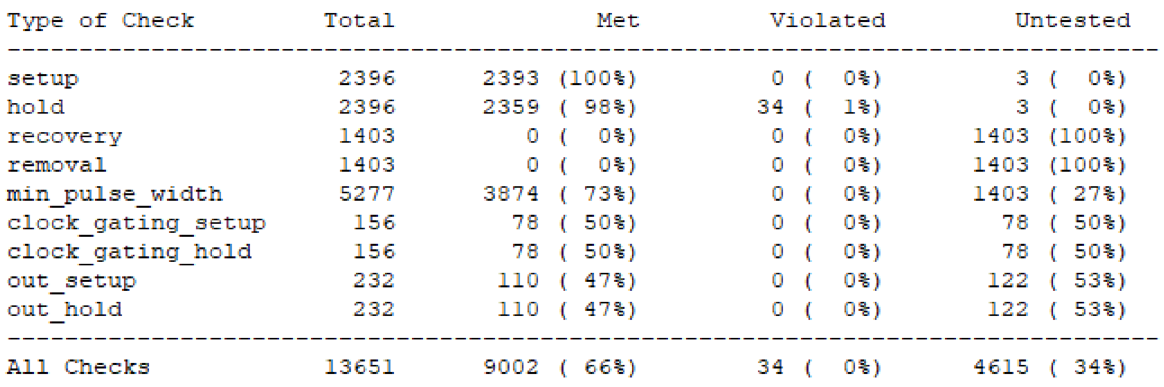
Figure 4.71: power map of the design of Typ-Typ (Flat Flow)

Figure 4.71 shows power distribution across hierarchy. It represents the percentage of power with respect to block that dissipate mostly power with respect to total power.

Block Name	Power relative to threshold(W)	Power relative to threshold(%)
U3355	0.000233796	99
if_stage_i	0.000109478	46
id_stage_i	0.000229038	97

Table 4.21: Power Design Map Statistics of Typ-Typ (Flat Flow)

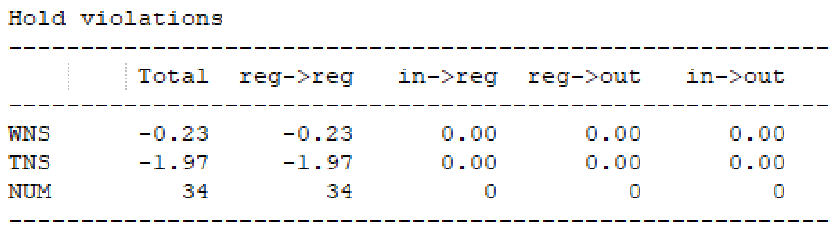
3- fast-fast at 0.85v and 125°C



```
Type of Check          Total            Met         Violated         Untested
------------------------------------------------------------------------------
setup                   2396      2393 (100%)        0 (   0%)        3 (   0%)
hold                    2396      2359 ( 98%)       34 (   1%)        3 (   0%)
recovery                1403         0 (  0%)        0 (   0%)     1403 (100%)
removal                 1403         0 (  0%)        0 (   0%)     1403 (100%)
min_pulse_width         5277      3874 ( 73%)        0 (   0%)     1403 ( 27%)
clock_gating_setup       156        78 ( 50%)        0 (   0%)       78 ( 50%)
clock_gating_hold        156        78 ( 50%)        0 (   0%)       78 ( 50%)
out_setup                232       110 ( 47%)        0 (   0%)      122 ( 53%)
out_hold                 232       110 ( 47%)        0 (   0%)      122 ( 53%)
------------------------------------------------------------------------------
All Checks             13651      9002 ( 66%)       34 (   0%)     4615 ( 34%)
```

Figure 4.72: Analysis coverage report of Fast-Fast (Flat Flow)

Figure 4.72 shows the initial checks on the design that shows the percentage of met, violated, and untested pathes for each check. For example for hold check there is 2359 path are met that is their percentage of all pathes is 98% and there is 34 violated path that is 1% of all pathes and there is 3 untested path that is their percentage of all pathes almost 0% and so on for all checks. The resons of untested pathes are false pathes such as Reset path and scan enable paths in clock gating these pathes are similar to Reset pathes and also another reson of untested path is that there is no startpoint and endpoint for pathes related to APU block that controls FPU operations in the design. While FPU is removed as mentioned in earlier chapters, APU is also not implemented which makes its related output ports grounded by default.



```
Hold violations
---------------------------------------------------------
          Total   reg->reg   in->reg  reg->out   in->out
---------------------------------------------------------
WNS       -0.23      -0.23      0.00      0.00      0.00
TNS       -1.97      -1.97      0.00      0.00      0.00
NUM          34         34         0         0         0
---------------------------------------------------------
```

Figure 4.73: Global timing report (HOLD) of Fast-Fast (Flat Flow)

Figure Figure 4.73 shows a global overview of timing violations before fixing these violations. As cleared there is no setup violations only thre is hold time violations that can be solved by insertting buffers.

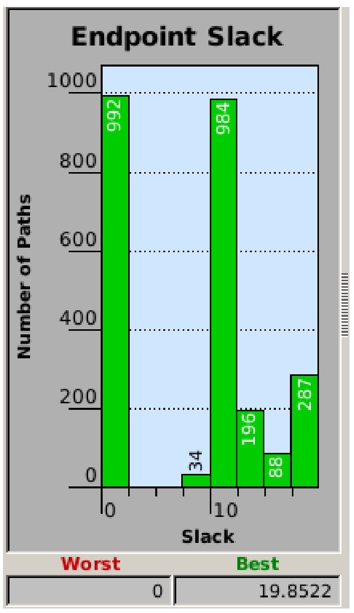


Figure 4.74: Setup slack histogram of Fast-Fast (Flat Flow)

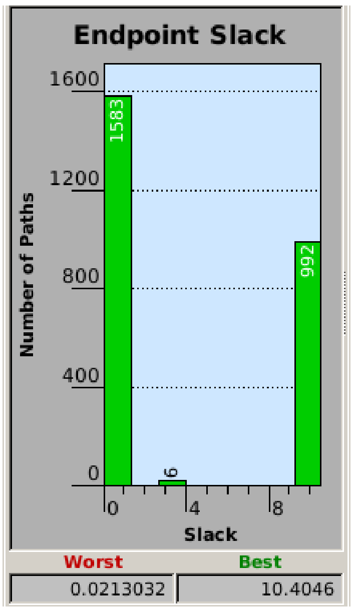


Figure 4.75: Hold slack histogram of Fast-Fast (Flat Flow)

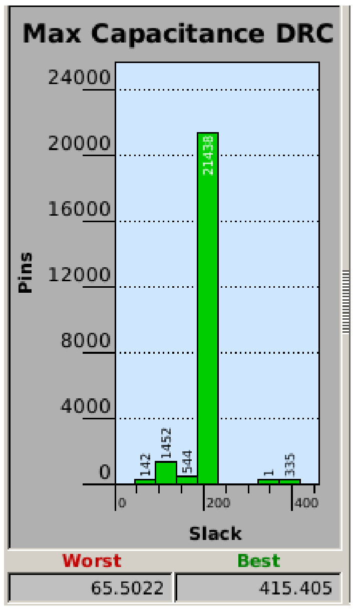


Figure 4.76: Max capacitance histogram of Fast-Fast (Flat Flow)

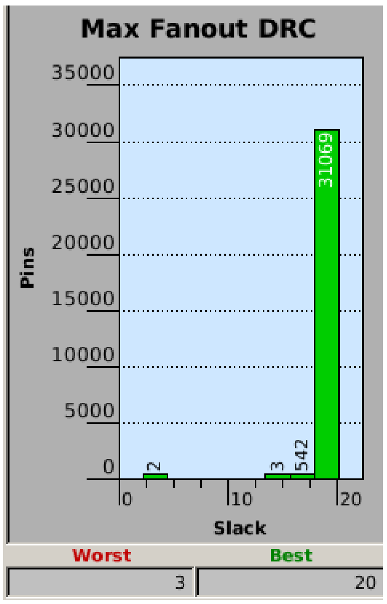


Figure 4.77: Max fanout histogram of Fast-Fast (Flat Flow)

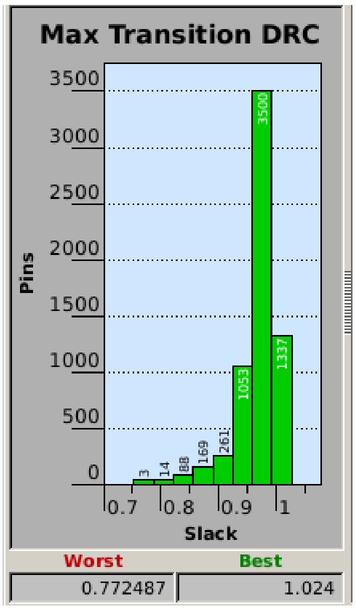


Figure 4.78: Max transation histogram of Fast-Fast (Flat Flow)

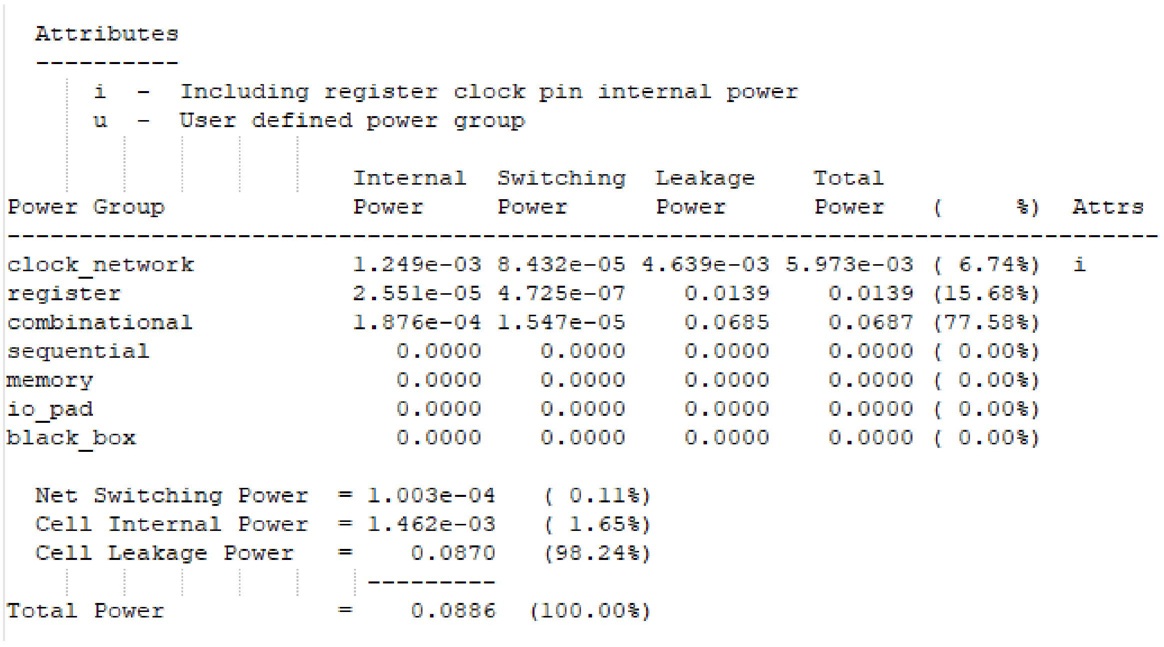
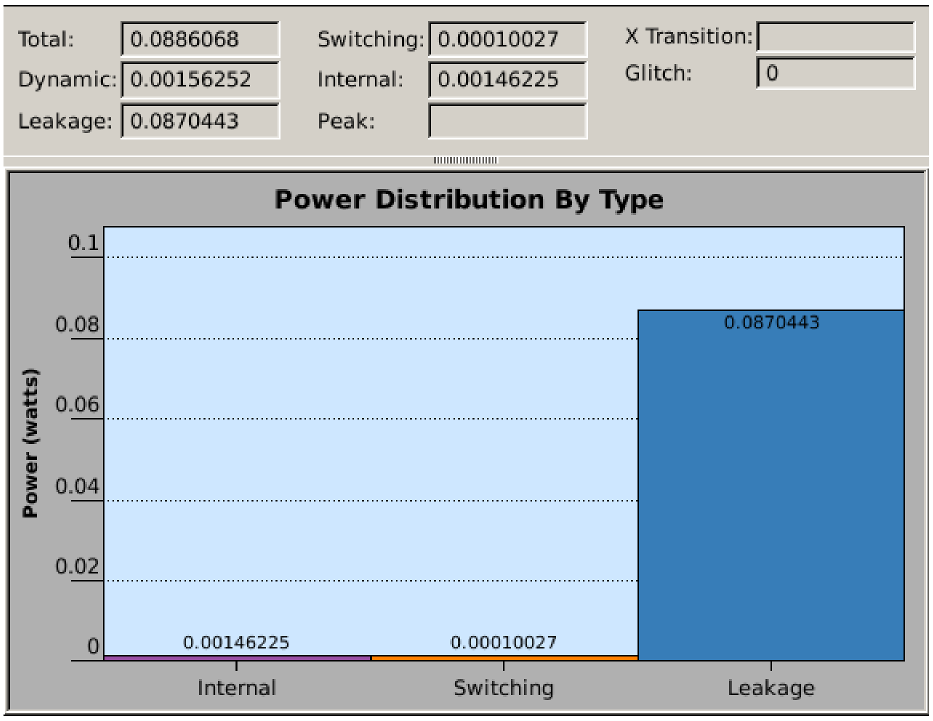

```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           1.249e-03 8.432e-05 4.639e-03 5.973e-03 ( 6.74%)  i
register                2.551e-05 4.725e-07    0.0139    0.0139 (15.68%)
combinational           1.876e-04 1.547e-05    0.0685    0.0687 (77.58%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 1.003e-04   ( 0.11%)
  Cell Internal Power  = 1.462e-03   ( 1.65%)
  Cell Leakage Power   =    0.0870   (98.24%)
                         ---------
Total Power            =    0.0886  (100.00%)
```

Figure 4.79: power report of Fast-Fast (Flat Flow)



Total:	0.0886068	Switching:	0.00010027	X Transition:	
Dynamic:	0.00156252	Internal:	0.00146225	Glitch:	0
Leakage:	0.0870443	Peak:			

Figure 4.80: power histogram of Fast-Fast (Flat Flow)

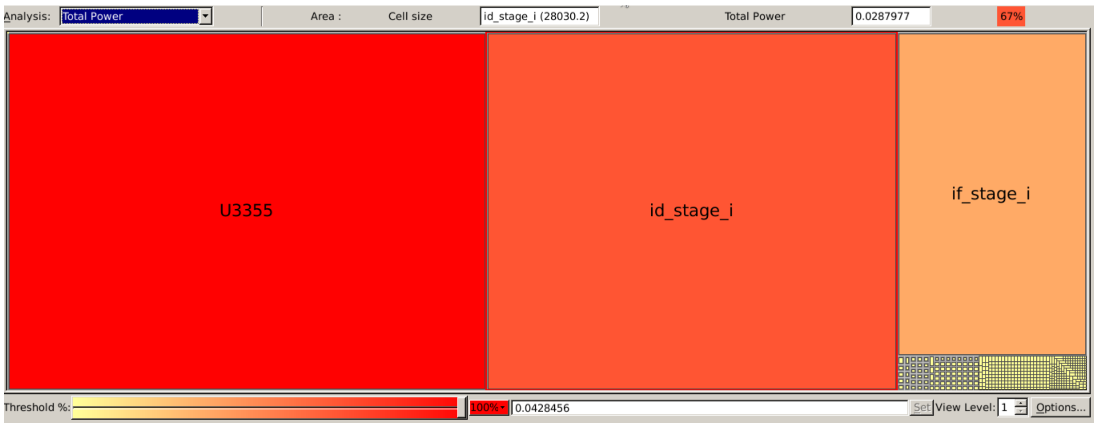


Figure 4.81: power map of the design of Fast-Fast (Flat Flow)

Figure 4.81 shows power distribution across hierarchy. It represents the percentage of power with respect to block that dissipate mostly power with respect to total power.

Block Name	Power relative to threshold(W)	Power relative to threshold(%)
U3355	0.0428456	100
if_stage_i	0.0143898	33
id_stage_i	0.0287977	67

Table 4.22: Power Design Map Statistics of Fast-Fast (Flat Flow)

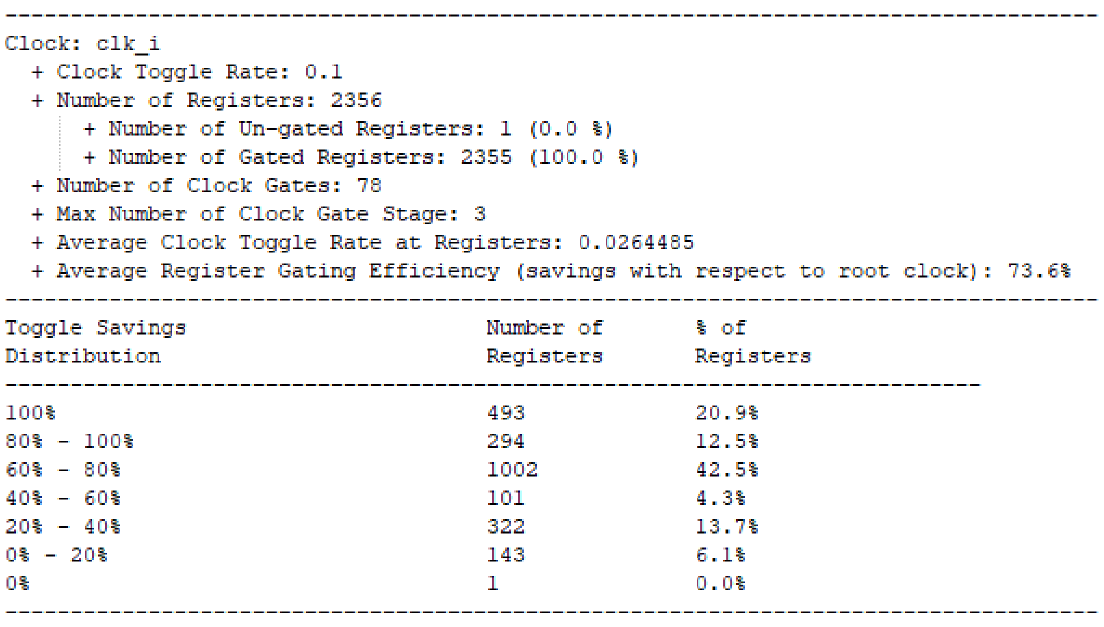
Finally, we succeed to make the design passes without timing violations for all 22 case.

4.3.2 Topographical Flow

In previous chapter we discussed total 22 verified corners out of total 27 corners. In the next sections, we describe in more details the results from verifying design performance using Synopsys PrimeTime STA , especially at three main corners which are:

- Slow-Slow , 0.7 V , 125°C
- Typ-Typ , 0.85 V , 25°C
- Fast-Fast, 1.16 V, -40°C

Clock gating savings



```
--------------------------------------------------------------------------------
Clock: clk_i
  + Clock Toggle Rate: 0.1
  + Number of Registers: 2356
      + Number of Un-gated Registers: 1 (0.0 %)
      + Number of Gated Registers: 2355 (100.0 %)
  + Number of Clock Gates: 78
  + Max Number of Clock Gate Stage: 3
  + Average Clock Toggle Rate at Registers: 0.0264485
  + Average Register Gating Efficiency (savings with respect to root clock): 73.6%
--------------------------------------------------------------------------------
Toggle Savings                        Number of       % of
Distribution                          Registers       Registers
-------------------------------------------------------------------------
100%                                  493             20.9%
80% - 100%                            294             12.5%
60% - 80%                             1002            42.5%
40% - 60%                             101             4.3%
20% - 40%                             322             13.7%
0% - 20%                              143             6.1%
0%                                    1               0.0%
--------------------------------------------------------------------------------
```

Figure 4.82: Clock Gating Savings Report (Topographical Flow)

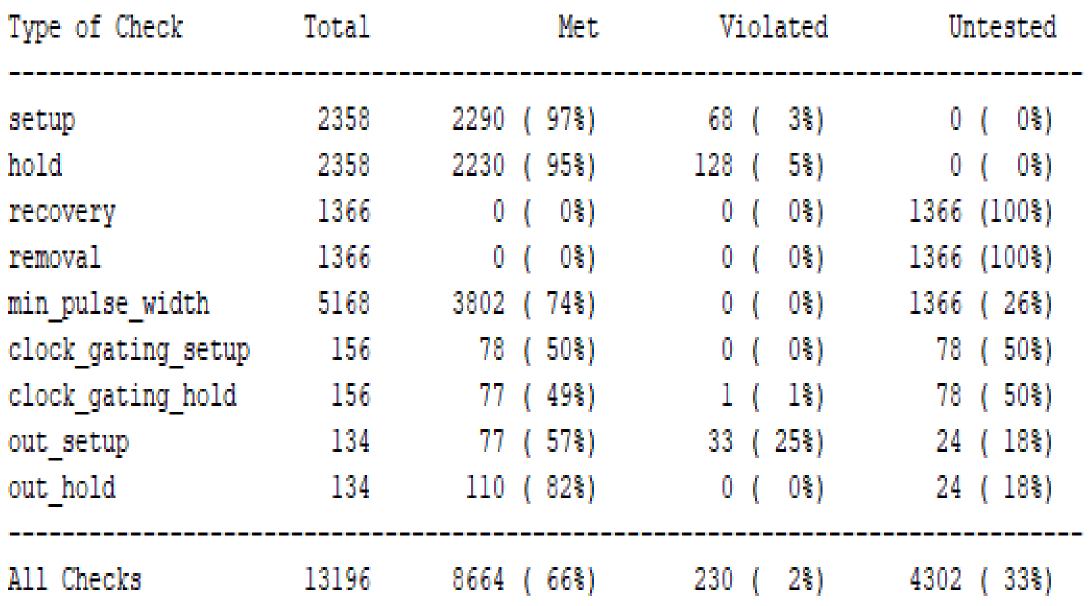
Before we go through STA results of the previous mentioned corners, we review the efficiency of clock gating in the design. Clock gating savings report shown in FIGURE 4.82 summarizes the overall effectiveness of clock gating in the design, the toggle savings and gating efficiency are computed based on 10% clock toggling rate. Average Register Gating efficiency is 73.6% , toggle saving reaches 60% to 80% for about half the sequential elements in the design which is good, since 60% to 80% toggles are suppressed by clock gating logic which highly reduces dynamic power in the design. Toggle saving is by definition the ratio between toggle rate at clock pin of registers to the toggle rate at clock root pin, toggle rate is represented as a percentage. Toggle rate of a sequential element means the rate at which output pin switches relative to the clock input frequency, toggle rate ranges between 0%-200%. Toggle rate of 100% as an example means that frequency of the output pin is half the frequency of the clock input. This convention comes from the assumption that the output changes every rising edge or a falling edge of the clock cycle.

TABLE 4.23 shows the initial number of violated paths over 22 verified corners, we can notice that setup violations only appears at ss0p7v125c corner case. Solving such violations by ECO fixing is performed on both worst case which is ss0p7v125c (Slow-Slow / 0.7 V / 125°C) and best case which is ff1p16vn40c (Fast-Fast / 1.16 V / -40°C), solving on these cases in its turn solved all other violations in the remaining corners.

case	number of SETUP violationed pathes	number of HOLD violationed pathes
ss0p7v125c	101	129
ss0p75v125c	0	56
ss0p95v25c	0	0
ss0p95v125c	0	0
ss0p95vn40c	0	0
tt0p78v25c	0	0
tt0p78v125c	0	0
tt0p78vn40c	0	0
tt0p85v25c	0	0
tt0p85v125c	0	0
tt0p85vn40c	0	0
tt1p05v25c	0	3
tt1p05v125c	0	0
tt1p05vn40c	0	0
ff0p85v25c	0	0
ff0p85v125c	0	0
ff0p85vn40c	0	0
ff0p95v25c	0	4
ff0p95v125c	0	2
ff0p95vn40c	0	6
ff1p16v25c	0	66
ff1p16vn40c	0	119

Table 4.23: Number of violated paths in verified cases before ECO Fixing (Topographical Flow)

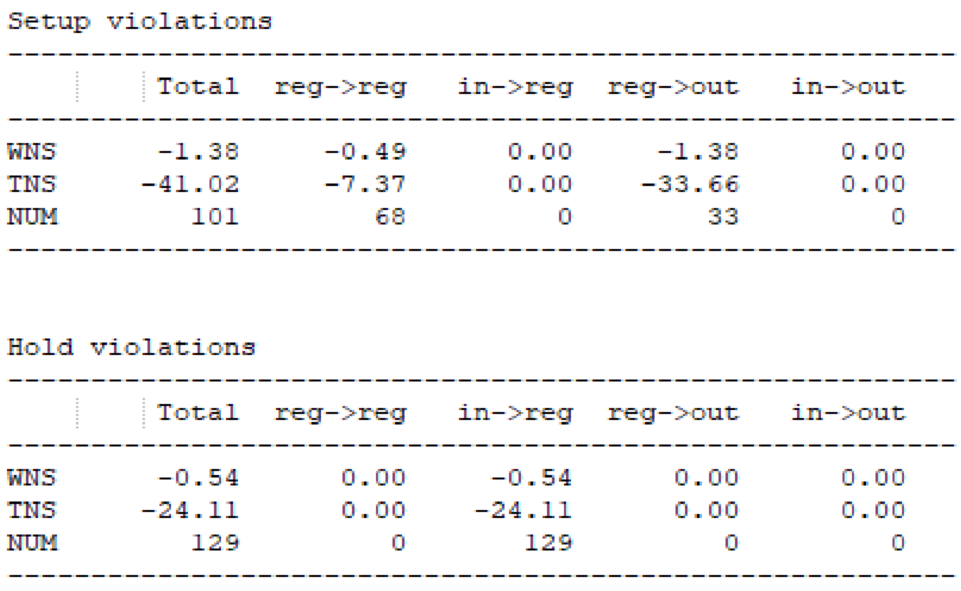
Slow-Slow Results



```
Type of Check          Total          Met        Violated        Untested
--------------------------------------------------------------------------
setup                   2358     2290 ( 97%)       68 (  3%)        0 (  0%)
hold                    2358     2230 ( 95%)      128 (  5%)        0 (  0%)
recovery                1366        0 (  0%)        0 (  0%)     1366 (100%)
removal                 1366        0 (  0%)        0 (  0%)     1366 (100%)
min_pulse_width         5168     3802 ( 74%)        0 (  0%)     1366 ( 26%)
clock_gating_setup       156       78 ( 50%)        0 (  0%)       78 ( 50%)
clock_gating_hold        156       77 ( 49%)        1 (  1%)       78 ( 50%)
out_setup                134       77 ( 57%)       33 ( 25%)       24 ( 18%)
out_hold                 134      110 ( 82%)        0 (  0%)       24 ( 18%)
--------------------------------------------------------------------------
All Checks             13196     8664 ( 66%)      230 (  2%)     4302 ( 33%)
```

Figure 4.83: Analysis Coverage Report of ss0p7v125c (Topographical Flow)

FIGURE 4.83 shows the initial analysis of the design performance at Slow-Slow corner. We can notice 68 setup violations, 128 hold violations, 1 hold violation at clock gating enable pin and 33 setup violations at specifically "REGOUT" paths. Untested paths are intentionally ignored paths from STA in our case. 1366 paths that are untested for recovery, removal and Min pulse width tests which are Reset paths, Reset paths are considered as false paths since they are being static most of the operation time. Also we can see 78 untested paths at clock gating setup/hold which are scan enable paths, scan enable paths are considered as false paths similar to Reset paths. We can also see there is 28 paths that are not checked in REGOUT path group, these paths are related to APU block that controls FPU operations in the design. While FPU is removed as mentioned in earlier chapters, APU is also not implemented which makes its related output ports grounded by default.



```
Setup violations
------------------------------------------------------------
              Total   reg->reg   in->reg  reg->out   in->out
------------------------------------------------------------
WNS           -1.38      -0.49      0.00     -1.38      0.00
TNS          -41.02      -7.37      0.00    -33.66      0.00
NUM             101         68         0        33         0
------------------------------------------------------------


Hold violations
------------------------------------------------------------
              Total   reg->reg   in->reg  reg->out   in->out
------------------------------------------------------------
WNS           -0.54       0.00     -0.54      0.00      0.00
TNS          -24.11       0.00    -24.11      0.00      0.00
NUM             129          0       129         0         0
------------------------------------------------------------
```

Figure 4.84: Global Timing Report of Slow-Slow (Topographical Flow)

FIGURE 4.84 shows a global overview on design timing status. After ECO rounds to solve these violations, we discuss new design status at Slow-Slow corner in the figures below.

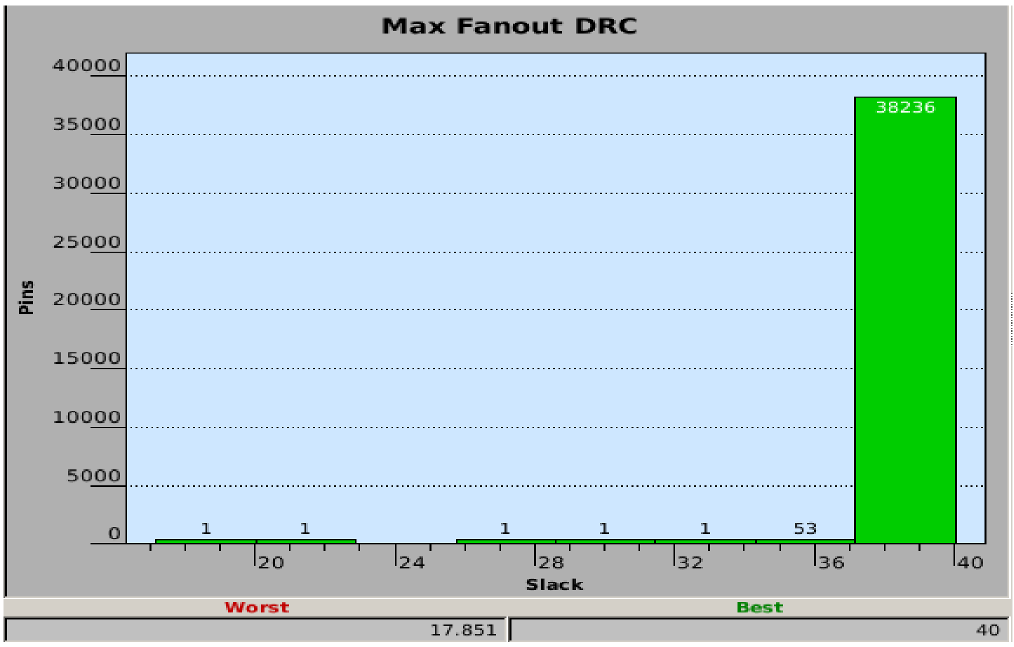


Figure 4.85: Max Fanout Histogram of Slow-Slow (Topographical Flow)

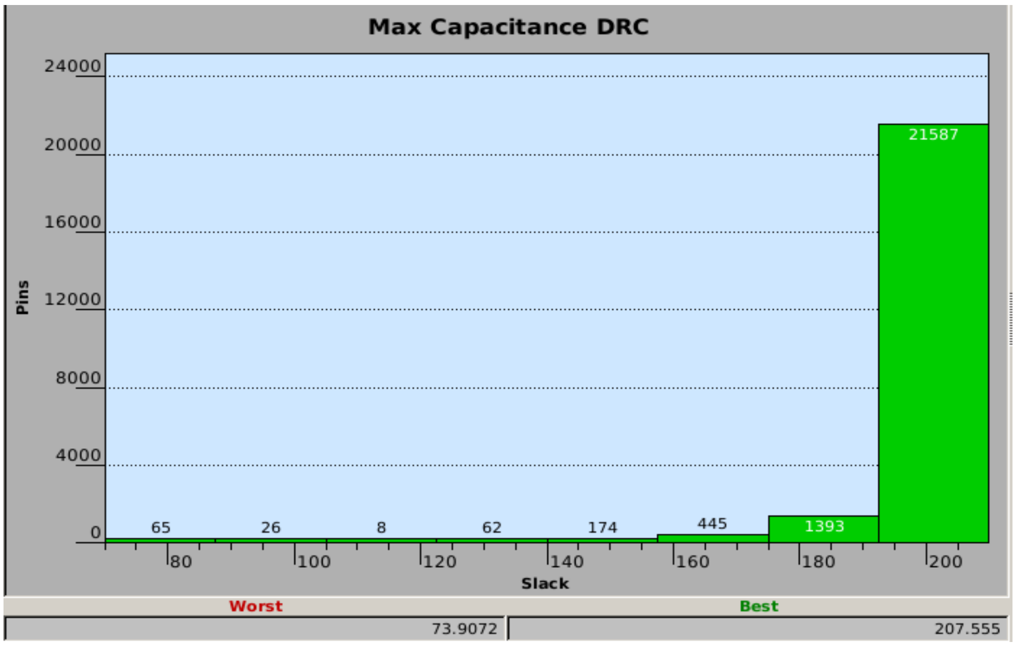


Figure 4.86: Max Capacitance Histogram of Slow-Slow (Topographical Flow)

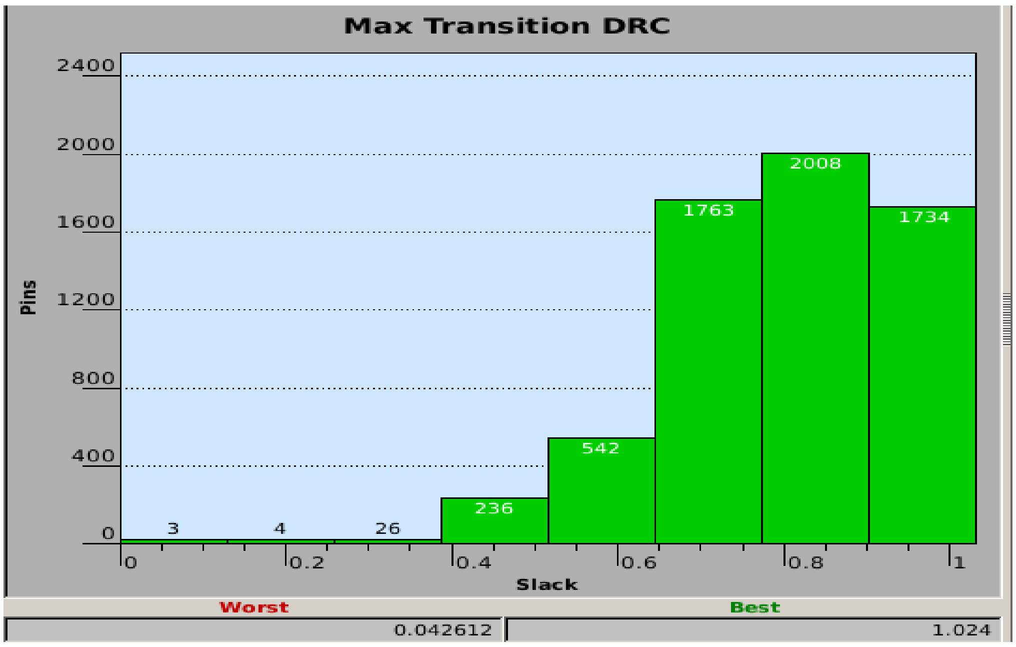


Figure 4.87: Max Transition Histogram of Slow-Slow (Topographical Flow)

As shown in previous three histograms, our design has no DRC violations at Slow-Slow corner after ECO. We also preview our design new timing status in the following two figures that show setup and hold timing histograms.

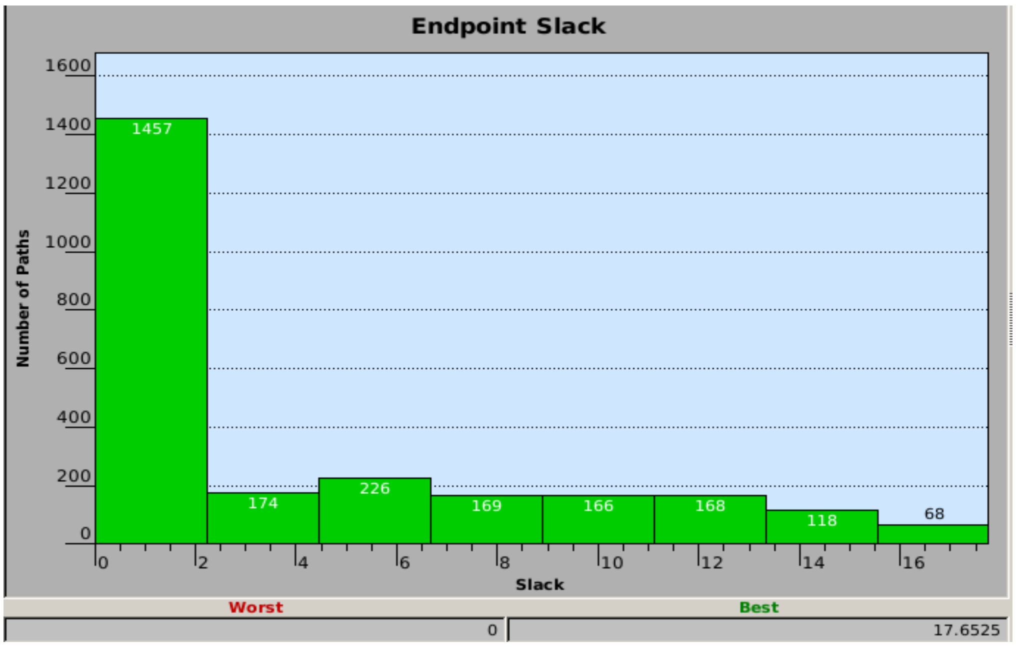


Figure 4.88: Setup Timing Histogram of Slow-Slow (Topographical Flow)

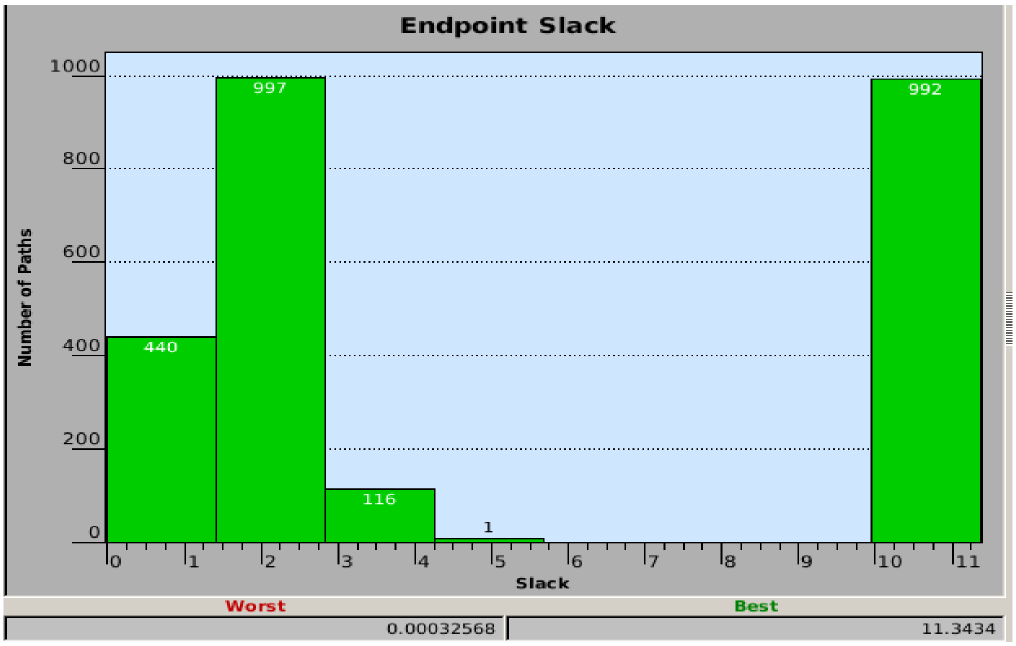
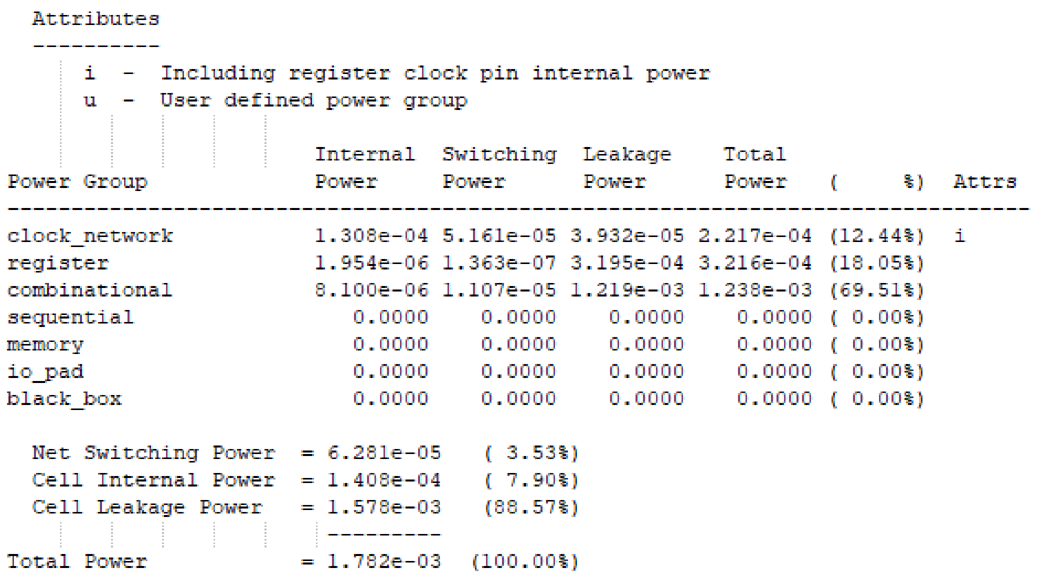
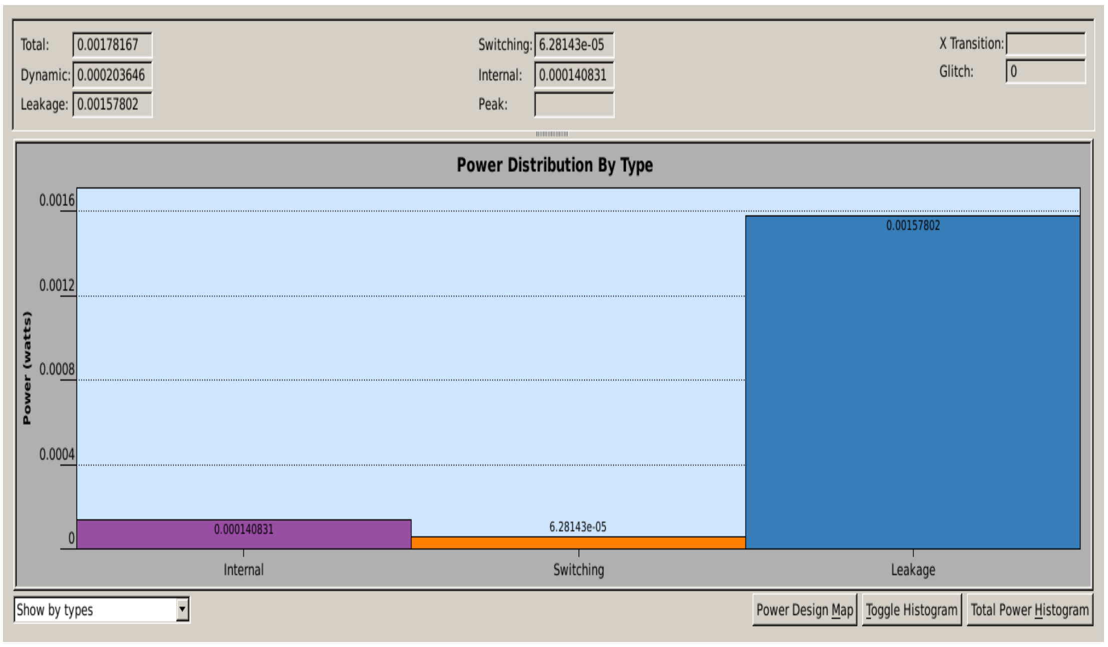


Figure 4.89: Hold Timing Histogram of Slow-Slow (Topographical Flow)



```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power    (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           1.308e-04 5.161e-05 3.932e-05 2.217e-04 (12.44%)  i
register                1.954e-06 1.363e-07 3.195e-04 3.216e-04 (18.05%)
combinational           8.100e-06 1.107e-05 1.219e-03 1.238e-03 (69.51%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 6.281e-05   ( 3.53%)
  Cell Internal Power  = 1.408e-04   ( 7.90%)
  Cell Leakage Power   = 1.578e-03   (88.57%)
                         ---------
Total Power            = 1.782e-03  (100.00%)
```

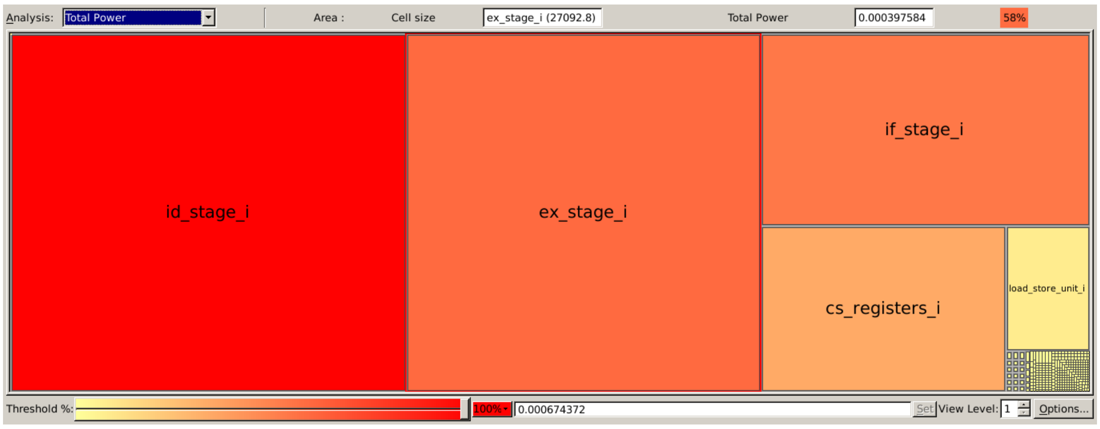
Figure 4.90: Total Power Report at Slow-Slow Corner (Topographical Flow)



```
Total:    0.00178167      Switching: 6.28143e-05      X Transition:
Dynamic:  0.000203646     Internal:  0.000140831      Glitch: 0
Leakage:  0.00157802      Peak:
```

Figure 4.91: Total Power Distribution at Slow-Slow Corner (Topographical Flow)

As shown earlier in Place And Route section, FIGURE 4.90 shows design power statistics. This is exactly the same report of PnR report, as Slow-Slow corner is the corner used to synthesize and implement the design. Power distribution by type is also shown in FIGURE 4.91.



Analysis: Total Power | Area : | Cell size | ex_stage_i (27092.8) | Total Power | 0.000397584 | 58%

Threshold %: | 100% | 0.000674372 | Set | View Level: 1 | Options...

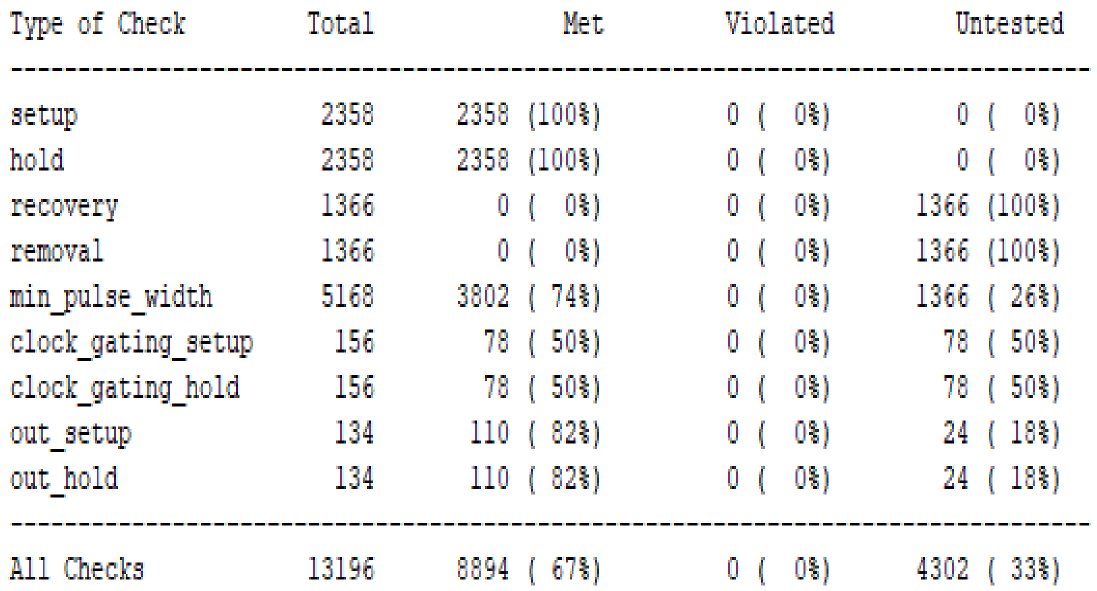
Figure 4.92: Power Design Map at Slow-Slow Corner (Topographical Flow)

FIGURE 4.92 shows power distribution across hierarchy. Power threshold in the bottom of the figure means the power associated with the block that contributes mostly to the total power which is ID stage as shown, other blocks' power is calculated as a percentage of this threshold. We can see that distribution is also by block's occupied area.

Block Name	**Power relative to threshold(W)**	**Power relative to threshold(%)**
ID Stage	0.000674	100
EX Stage	0.000397	58
IF Stage	0.00036	53
CS Registers	0.000226	33
LSU	0.0000051	7

Table 4.24: Power Design Map Statistics Table of Slow-Slow (Topographical Flow)

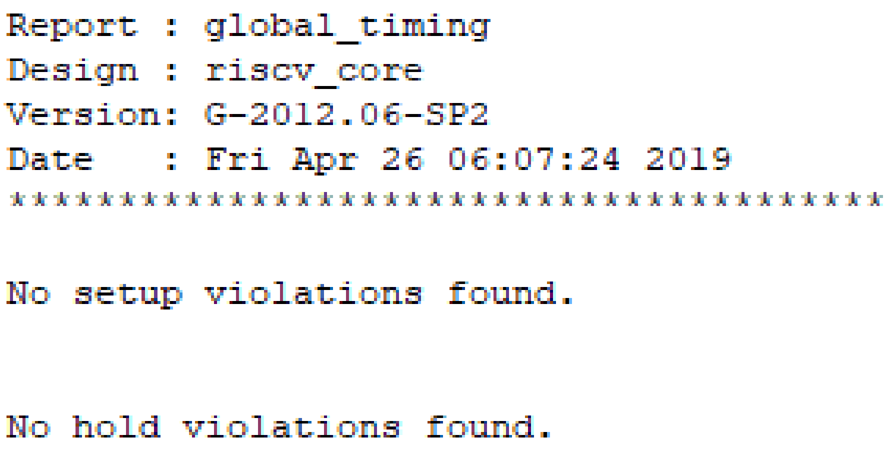
Typical-Typical Results



```
Type of Check        Total            Met        Violated        Untested
---------------------------------------------------------------------------
setup                 2358     2358 (100%)        0 (   0%)        0 (   0%)
hold                  2358     2358 (100%)        0 (   0%)        0 (   0%)
recovery              1366        0 (   0%)       0 (   0%)     1366 (100%)
removal               1366        0 (   0%)       0 (   0%)     1366 (100%)
min_pulse_width       5168     3802 ( 74%)        0 (   0%)     1366 ( 26%)
clock_gating_setup     156       78 ( 50%)        0 (   0%)       78 ( 50%)
clock_gating_hold      156       78 ( 50%)        0 (   0%)       78 ( 50%)
out_setup              134      110 ( 82%)        0 (   0%)       24 ( 18%)
out_hold               134      110 ( 82%)        0 (   0%)       24 ( 18%)
---------------------------------------------------------------------------
All Checks           13196     8894 ( 67%)        0 (   0%)     4302 ( 33%)
```

Figure 4.93: Analysis Coverage Report of Typ-Typ Corner (Topographical Flow)

FIGURE 4.93 shows no timing violations initially at tt0p85v25c corner case, this is expected with typical-typical conditions as violations mainly are concentrated at both worst (slow-slow) and best (fast-fast) corners.



```
Report : global_timing
Design : riscv_core
Version: G-2012.06-SP2
Date   : Fri Apr 26 06:07:24 2019
****************************************

No setup violations found.


No hold violations found.
```

Figure 4.94: Global Timing Report of Typ-Typ Corner (Topographical Flow)

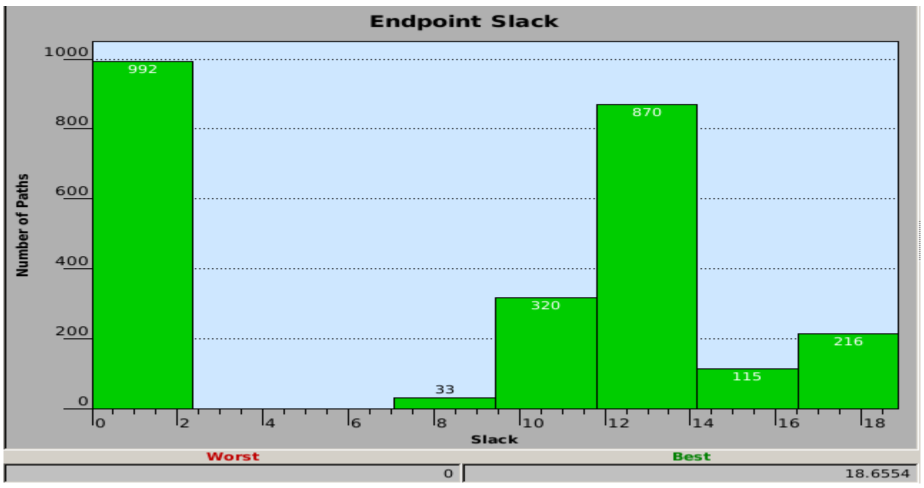


Figure 4.95: Setup Timing Histogram of Typ-Typ Corner (Topographical Flow)

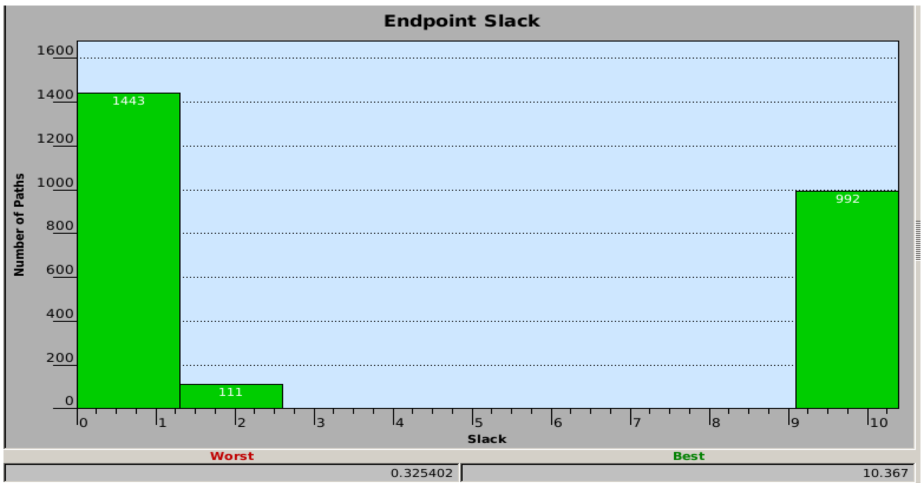


Figure 4.96: Hold Timing Histogram of Typ-Typ Corner (Topographical Flow)

Timing histograms for setup/hold at typical-typical corner cases are shown in FIGURES 4.95 and 4.96. In FIGURE 4.95 we can clearly observe latch-based timing paths with 0 setup slack at the left most of the histogram due to time-borrowing. Physical results related to typical-typical corner case are shown in FIGURES 4.97, 4.98, and 4.99. Design in Typ-Typ corner has no DRC violations.

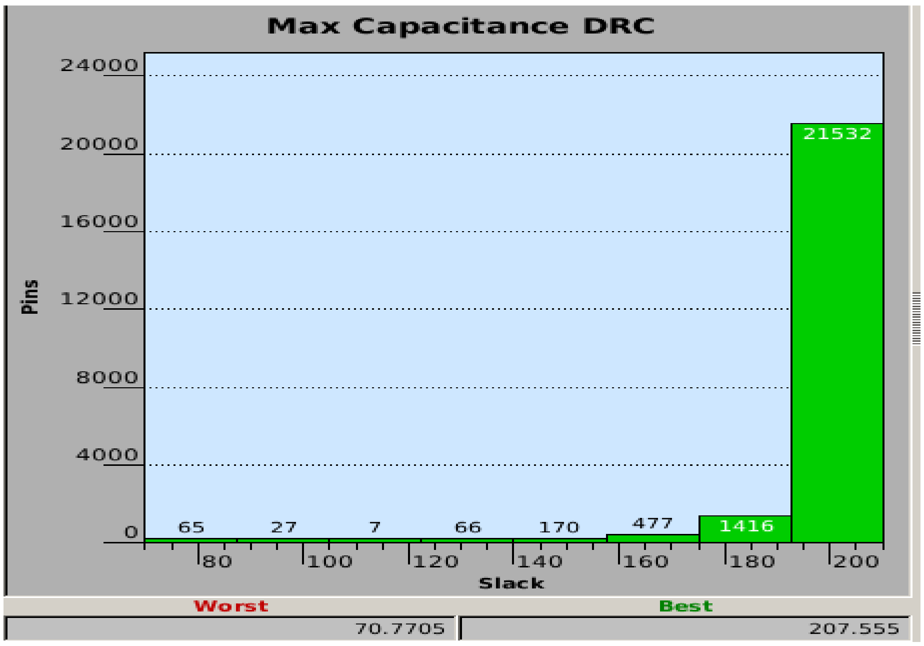


Figure 4.97: Max Capacitance Histogram of Typ-Typ Corner (Topographical Flow)

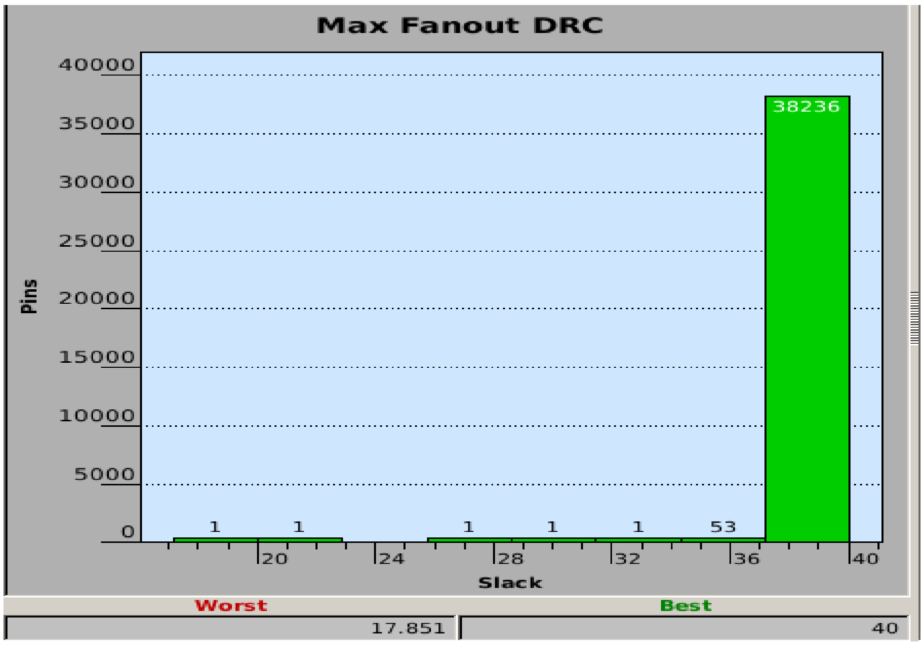


Figure 4.98: Max Fanout Histogram of Typ-Typ Corner (Topographical Flow)

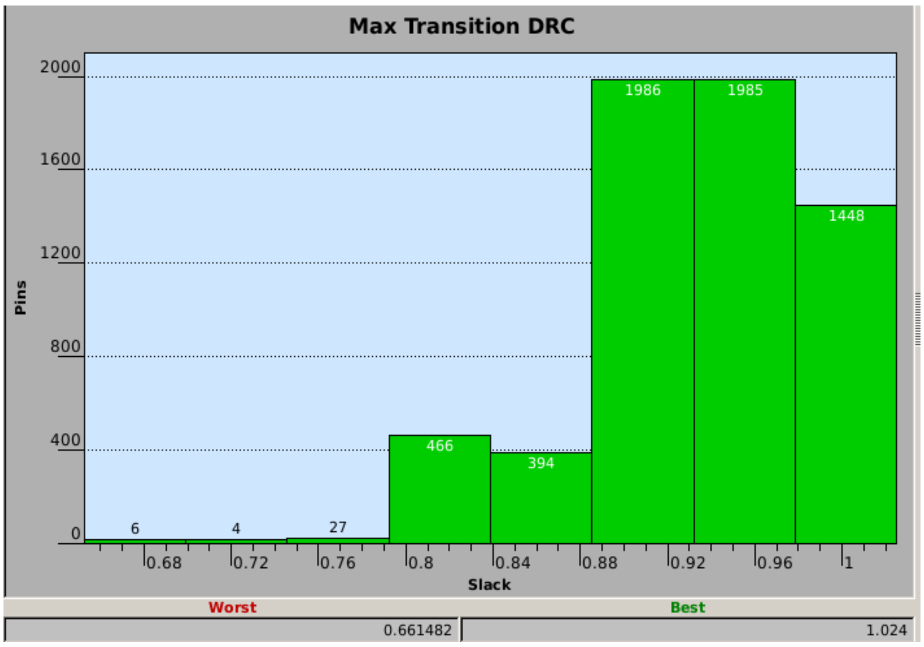
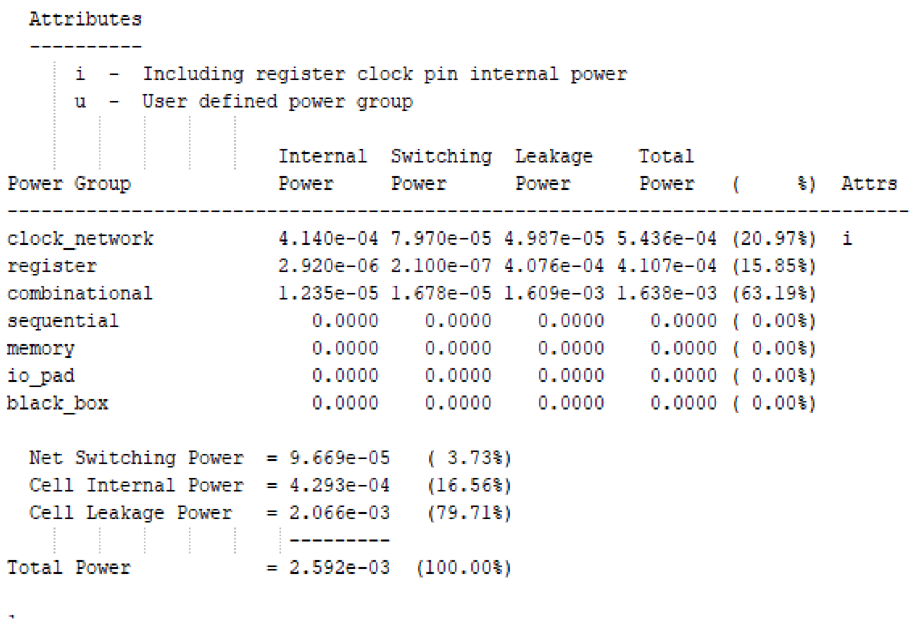


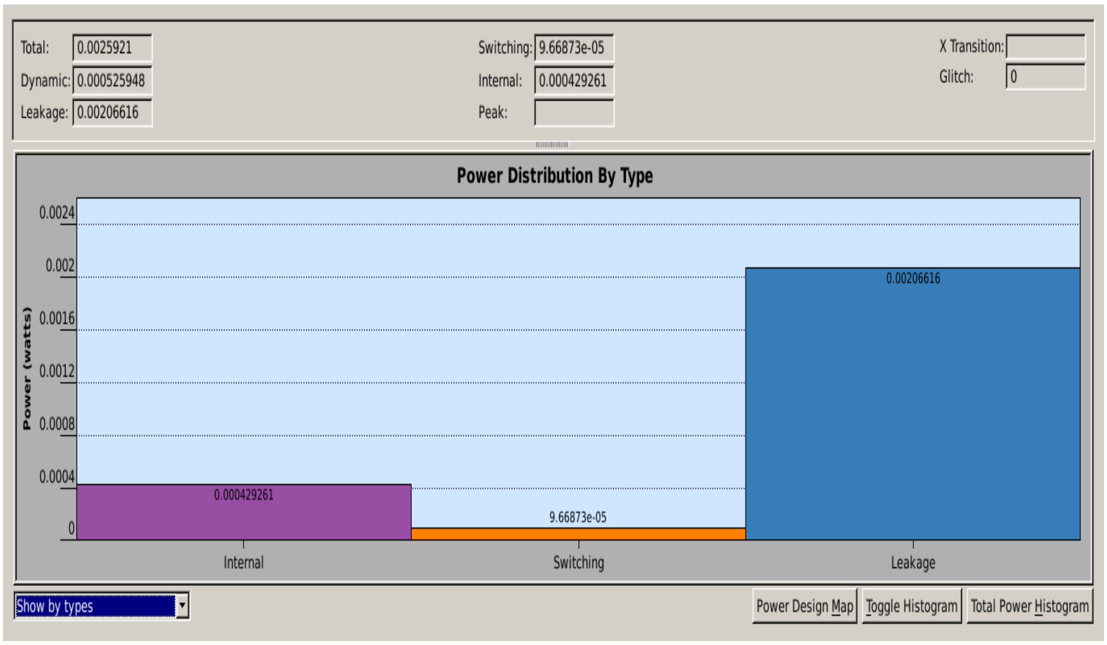
Figure 4.99: Max Transition Histogram of Typ-Typ Corner (Topographical Flow)



```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage   Total
Power Group             Power     Power      Power     Power    (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           4.140e-04 7.970e-05 4.987e-05 5.436e-04 (20.97%)  i
register                2.920e-06 2.100e-07 4.076e-04 4.107e-04 (15.85%)
combinational           1.235e-05 1.678e-05 1.609e-03 1.638e-03 (63.19%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 9.669e-05   ( 3.73%)
  Cell Internal Power  = 4.293e-04   (16.56%)
  Cell Leakage Power   = 2.066e-03   (79.71%)
                         ---------
Total Power            = 2.592e-03  (100.00%)
```

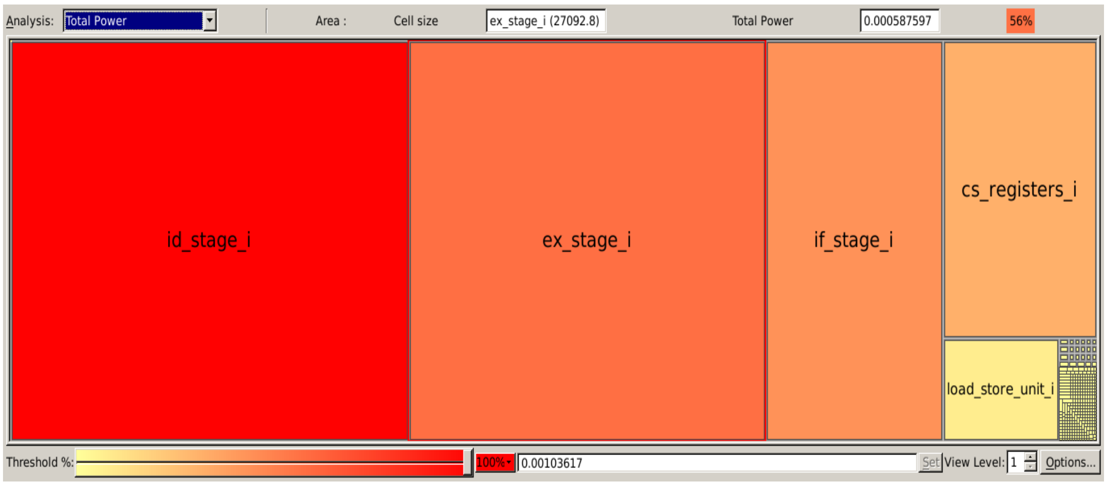
Figure 4.100: Total Power Report on Typ-Typ Corner (Topographical Flow)

FIGURE 4.100 shows design power statistics at Typ-Typ corner. Total power increased in typical conditions relative to total power calculated at Slow-Slow corner.



Total: 0.0025921
Dynamic: 0.000525948
Leakage: 0.00206616
Switching: 9.66873e-05
Internal: 0.000429261
Peak:
X Transition:
Glitch: 0

Figure 4.101: Total Power Distribution by type at Typ-Typ Corner (Topographical Flow)



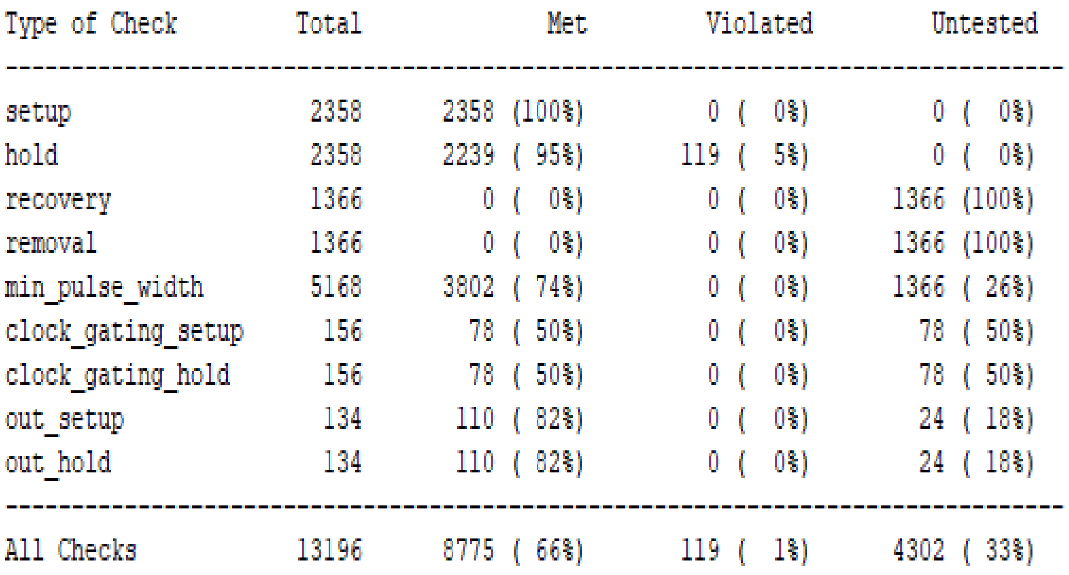
Analysis: Total Power
Area :
Cell size
ex_stage_i (27092.8)
Total Power 0.000587597
56%
Threshold %: 100% 0.00103617
View Level: 1

Figure 4.102: Power Design Map at Typ-Typ Corner (Topographical Flow)

Block Name	Power relative to threshold(W)	Power relative to threshold(%)
ID Stage	0.001036	100
EX Stage	0.000587	56
IF Stage	0.0004455	42
CS Registers	0.0003259	31
LSU	0.0000077	7

Table 4.25: Power Design Map Statistics Table of Typ-Typ Corner (Topographical Flow)

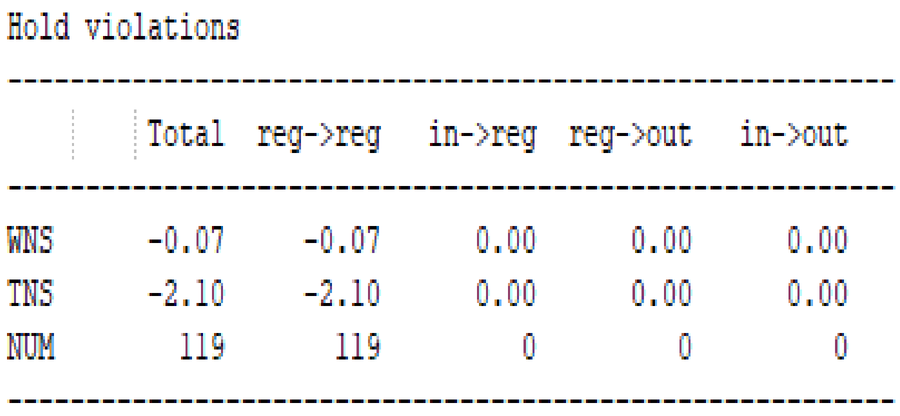
Fast-Fast Results



```
Type of Check          Total           Met        Violated        Untested
--------------------------------------------------------------------------------
setup                   2358      2358 (100%)       0 (  0%)        0 (  0%)
hold                    2358      2239 ( 95%)     119 (  5%)        0 (  0%)
recovery                1366         0 (  0%)       0 (  0%)     1366 (100%)
removal                 1366         0 (  0%)       0 (  0%)     1366 (100%)
min_pulse_width         5168      3802 ( 74%)       0 (  0%)     1366 ( 26%)
clock_gating_setup       156        78 ( 50%)       0 (  0%)       78 ( 50%)
clock_gating_hold        156        78 ( 50%)       0 (  0%)       78 ( 50%)
out_setup                134       110 ( 82%)       0 (  0%)       24 ( 18%)
out_hold                 134       110 ( 82%)       0 (  0%)       24 ( 18%)
--------------------------------------------------------------------------------
All Checks             13196      8775 ( 66%)     119 (  1%)     4302 ( 33%)
```

Figure 4.103: Analysis Coverage Report of Fast-Fast Corner

FIGURE 4.103 shows initial timing analysis of the design at ff1p16vn40c (Fast-Fast / 1.16 V / -40°C), we can see about 5% of hold timing paths are violated and no setup timing violations .



```
Hold violations
------------------------------------------------------------
          Total   reg->reg   in->reg   reg->out   in->out
------------------------------------------------------------
WNS       -0.07     -0.07      0.00      0.00      0.00
TNS       -2.10     -2.10      0.00      0.00      0.00
NUM         119       119         0         0         0
------------------------------------------------------------
```

Figure 4.104: Global Timing Report of Fast-Fast Corner (Topographical Flow)

Violated hold timing paths are found to be specifically reg2reg timing paths as shown in FIGURE 4.104. FIGUREs 4.105 and 4.106 show design timing status after ECO fixing at Fast-Fast corner.

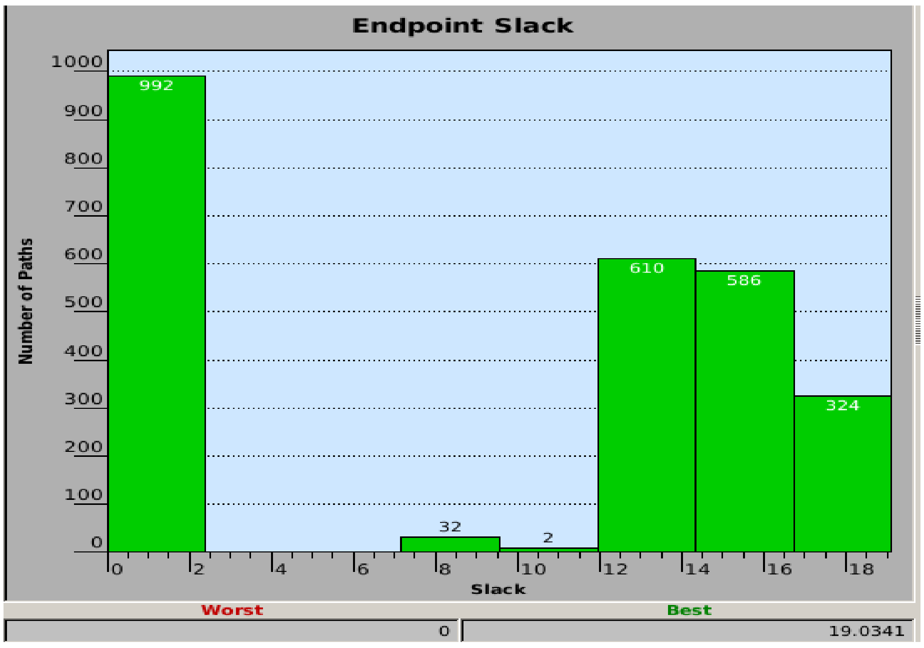


Figure 4.105: Setup Timing Histogram of Fast-Fast Corner (Topographical Flow)

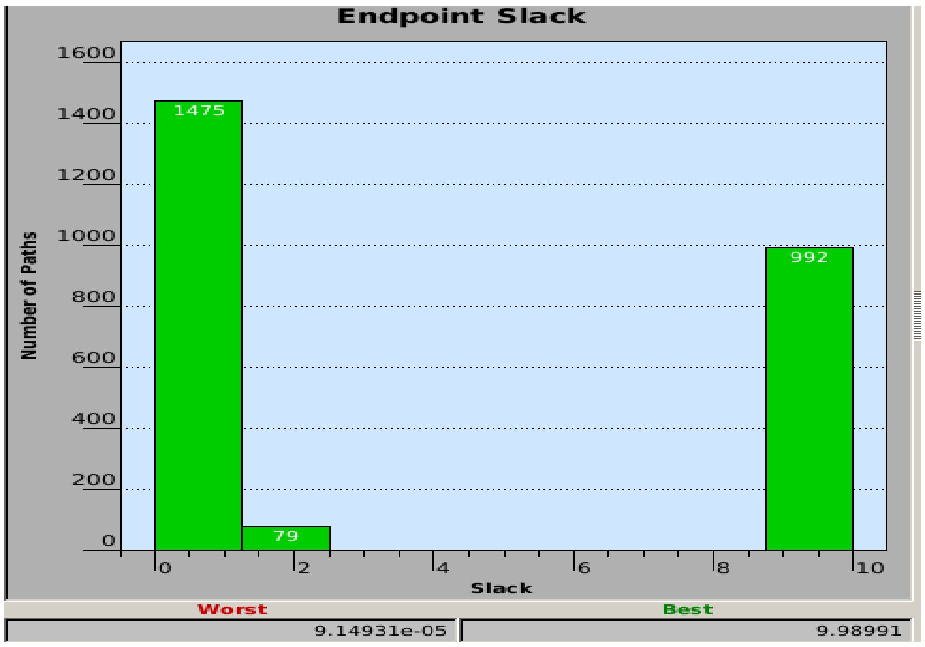


Figure 4.106: Hold Timing Histogram of Fast-Fast Corner (Topographical Flow)

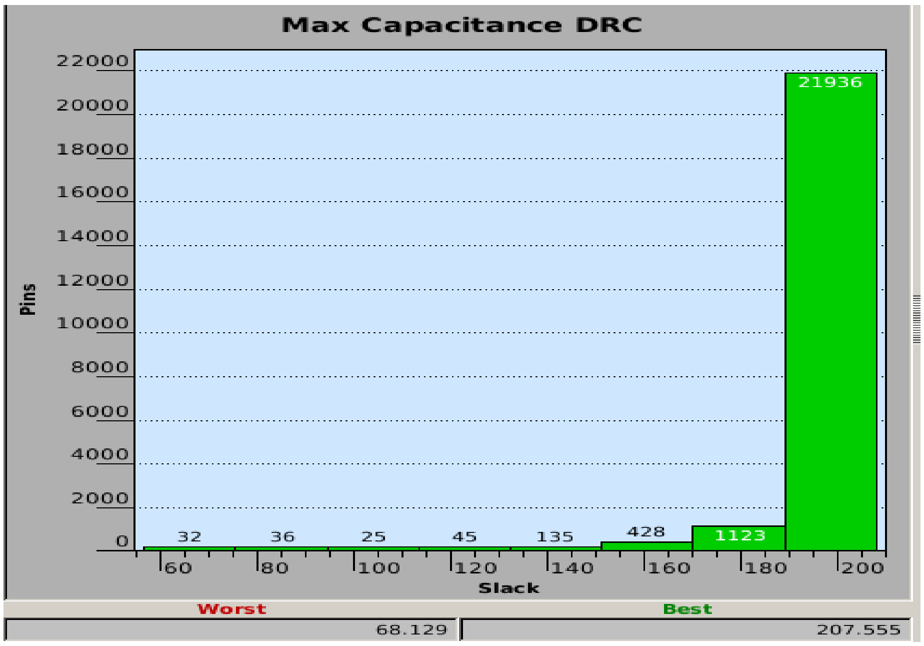


Figure 4.107: Max Capacitance Histogram of Fast-Fast Corner (Topographical Flow)

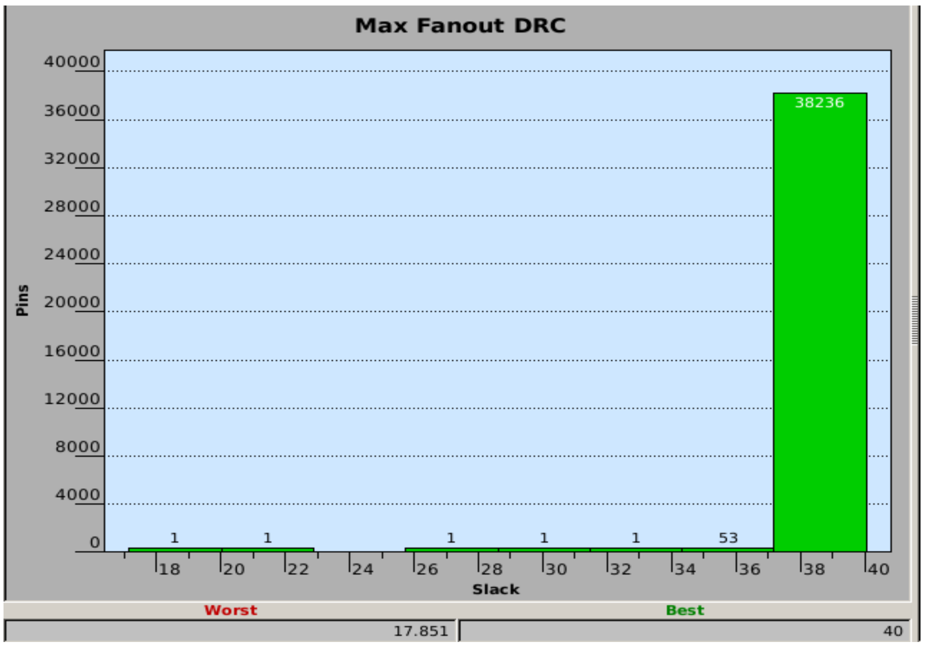


Figure 4.108: Max Fanout Histogram of Fast-Fast Corner (Topographical Flow)

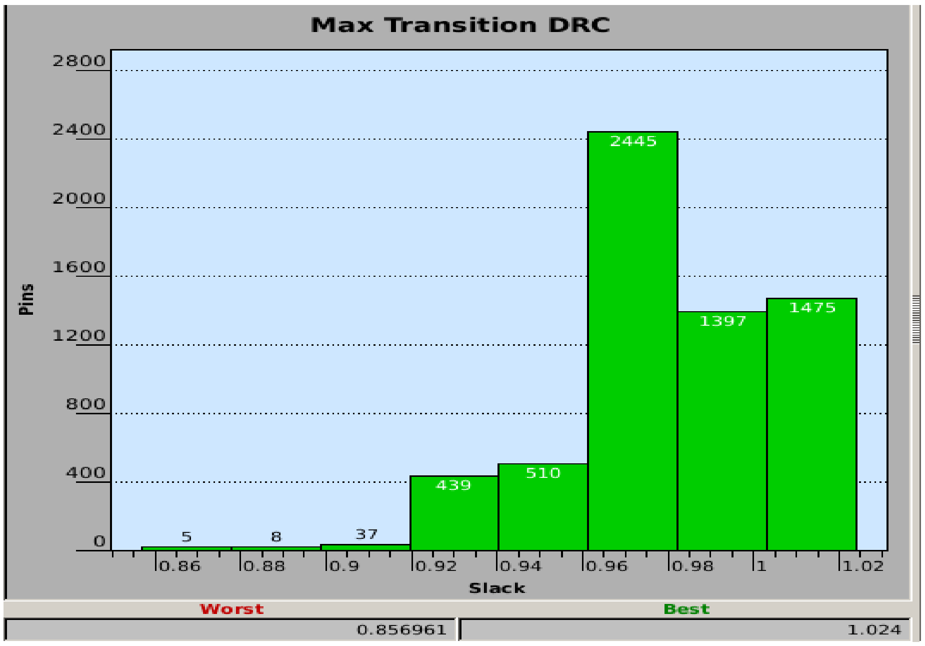
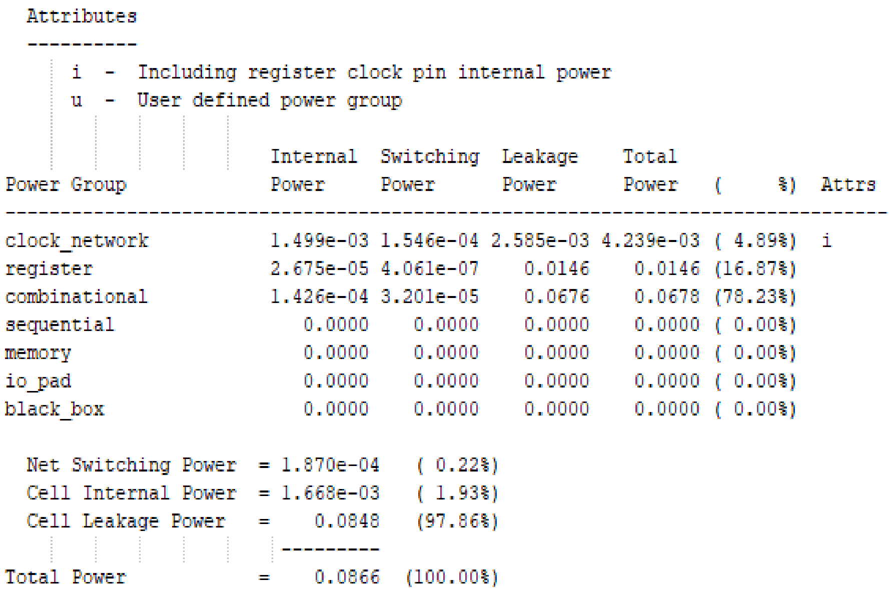


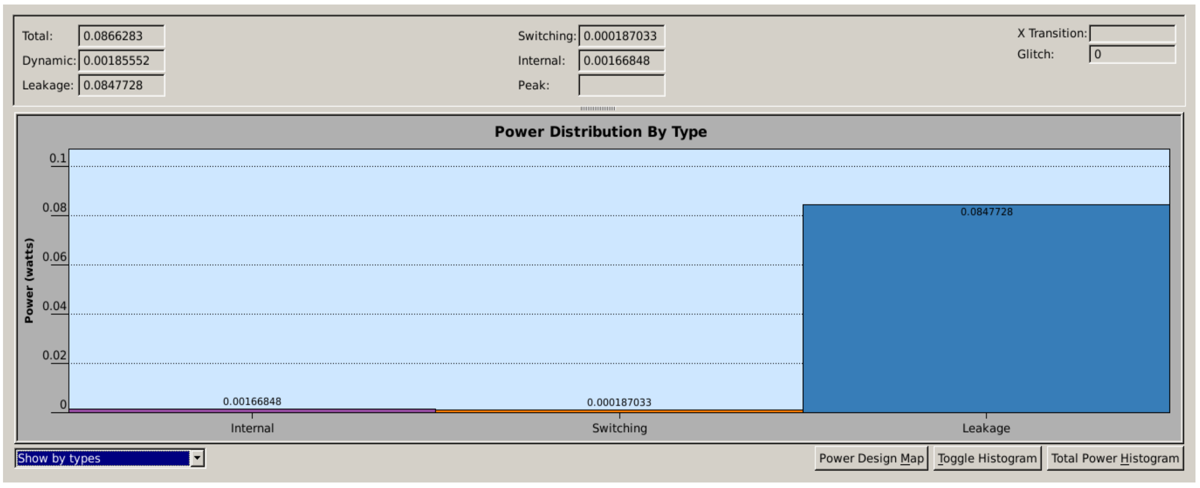
Figure 4.109: Max Transition Histogram of Fast-Fast Corner (Topographical Flow)

FIGUREs 4.107, 4.108, and 4.109 show that design is physically clean of any DRC violations at Fast-Fast corner.



```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           1.499e-03 1.546e-04 2.585e-03 4.239e-03 ( 4.89%)  i
register                2.675e-05 4.061e-07    0.0146    0.0146 (16.87%)
combinational           1.426e-04 3.201e-05    0.0676    0.0678 (78.23%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 1.870e-04   ( 0.22%)
  Cell Internal Power  = 1.668e-03   ( 1.93%)
  Cell Leakage Power   =    0.0848   (97.86%)
                         ---------
Total Power            =    0.0866  (100.00%)
```

Figure 4.110: Total Power Report at Fast-Fast Corner (Topographical Flow)



Total: 0.0866283
Dynamic: 0.00185552
Leakage: 0.0847728
Switching: 0.000187033
Internal: 0.00166848
Peak:
X Transition:
Glitch: 0

Figure 4.111: Total Power Distribution by type at Fast-Fast Corner (Topographical Flow)

FIGURE 4.110 shows design power statistics at Fast-Fast corner with total power of 0.0866W, it shows a large total power increase due to extensive increase in leakage power at Fast-Fast corner. This is considered as the highest total power calculated over all 22 verified corners.

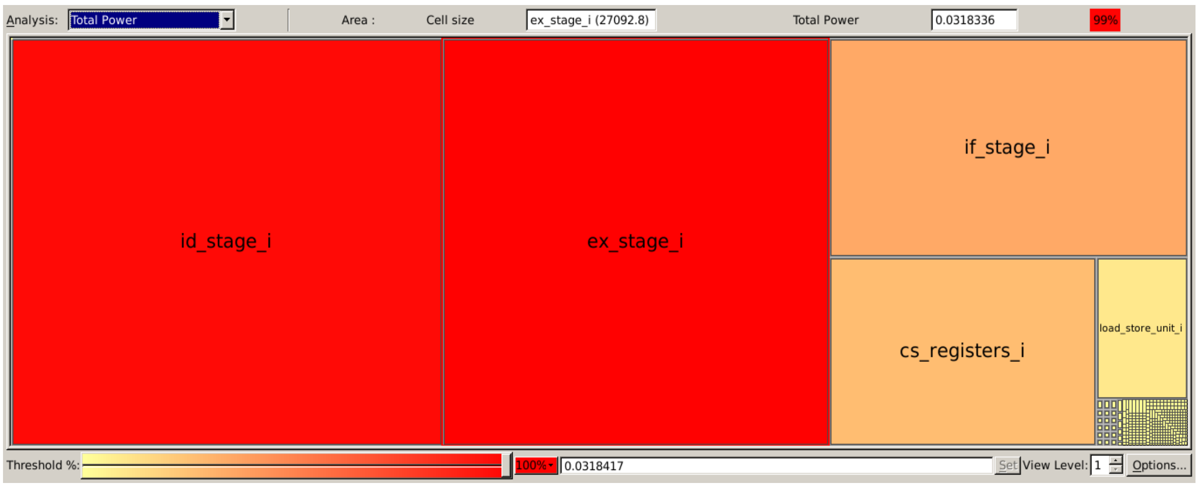


Figure 4.112: Power Design Map at Fast-Fast Corner (Topographical Flow)

Block Name	Power relative to threshold(W)	Power relative to threshold(%)
ID Stage	0.0308	100
EX Stage	0.0318	99
IF Stage	0.0108	34
CS Registers	0.00838	26
LSU	0.0029	9

Table 4.26: Power Design Map Statistics Table of Fast-Fast Corner (Topographical Flow)

4.3.3 Hierarchical Flow

Clock gating savings

figure4.113 , show the clock gating savings which we review the efficiency of clock gating in the design. Clock gating savings report shown in figure 4.113 summarizes the overall effectiveness of clock gating in the design, the toggle savings and gating efficiency are computed based on 10% clock toggling rate. Average Register Gating efficiency is 71.6% , toggle saving reaches 60% to 80% for nearly half the sequential elements in the design which is good, since 60% to 80% toggles are suppressed by clock gating logic which highly reduces dynamic power in the design. Toggle saving is by definition the ratio between toggle rate at clock pin of registers to the toggle rate at clock root pin, toggle rate is represented as a percentage. Toggle rate of a sequential element means the rate at which output pin switches relative to the clock input frequency, toggle rate ranges between 0%-200%. Toggle rate of 100% as an example means that frequency of the output pin is half the frequency of the clock input. This convention comes from the assumption that the output changes every rising edge or a falling edge of the clock cycle.

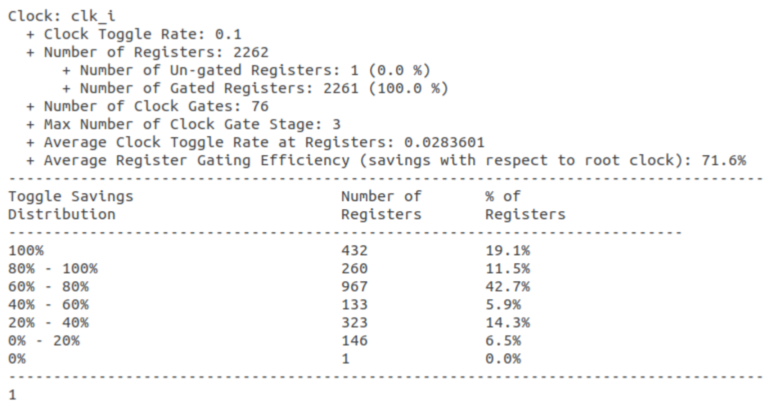

```
Clock: clk_i
  + Clock Toggle Rate: 0.1
  + Number of Registers: 2262
      + Number of Un-gated Registers: 1 (0.0 %)
      + Number of Gated Registers: 2261 (100.0 %)
  + Number of Clock Gates: 76
  + Max Number of Clock Gate Stage: 3
  + Average Clock Toggle Rate at Registers: 0.0283601
  + Average Register Gating Efficiency (savings with respect to root clock): 71.6%
-----------------------------------------------------------------------------------
Toggle Savings                       Number of       % of
Distribution                         Registers       Registers
--------------------------------------------------------------------------
100%                                 432             19.1%
80% - 100%                           260             11.5%
60% - 80%                            967             42.7%
40% - 60%                            133             5.9%
20% - 40%                            323             14.3%
0% - 20%                             146             6.5%
0%                                   1               0.0%
-----------------------------------------------------------------------------------
1
```

Figure 4.113: Clock gating savings Report in Hierarchical Flow

We have tested our design at 22 cases out of 27 and has met the timing requirement and there is no hold or setup violations . We will show one case from each category of corner cases. Worst case discussed is ss0p7v125c, best case is ff0p85v125c and typical case is tt0p78v25c .

Worst Case Results

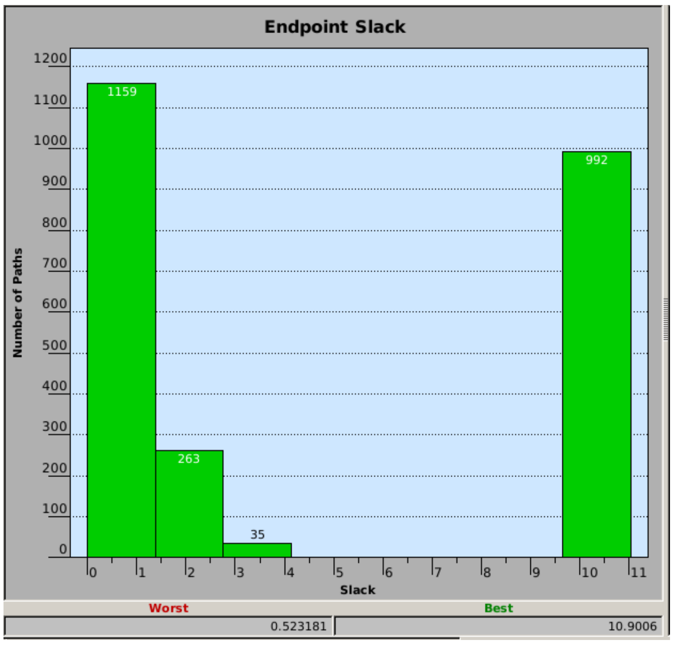


Figure 4.114: Hold timing histogram at ss0p7v125c case in Hierarchical Flow

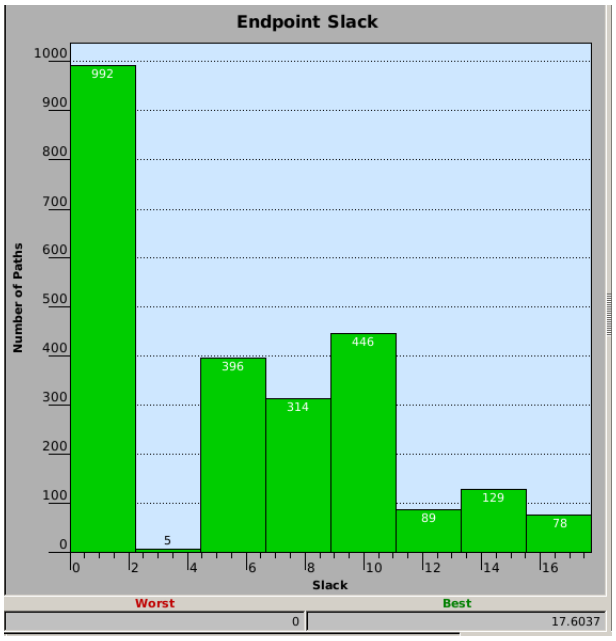


Figure 4.115: Setup timing histogram at ss0p7v125c case in Hierarchical Flow

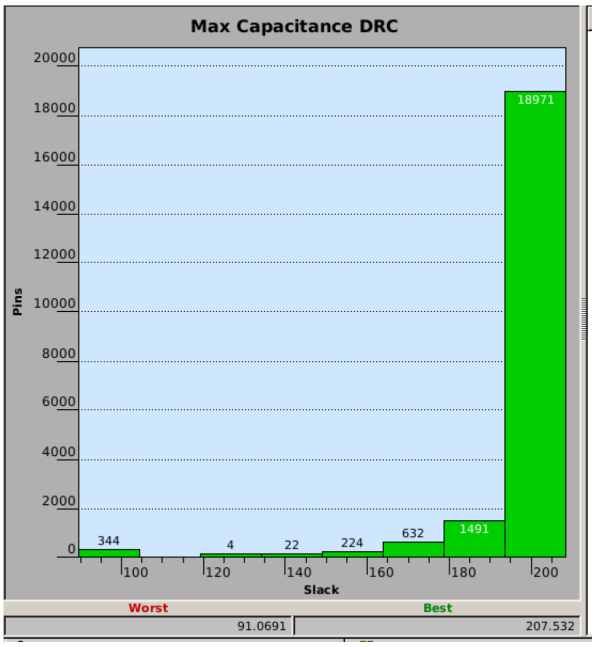


Figure 4.116: Max Capacitance Histogram at ss0p7v125c case in Hierarchical Flow

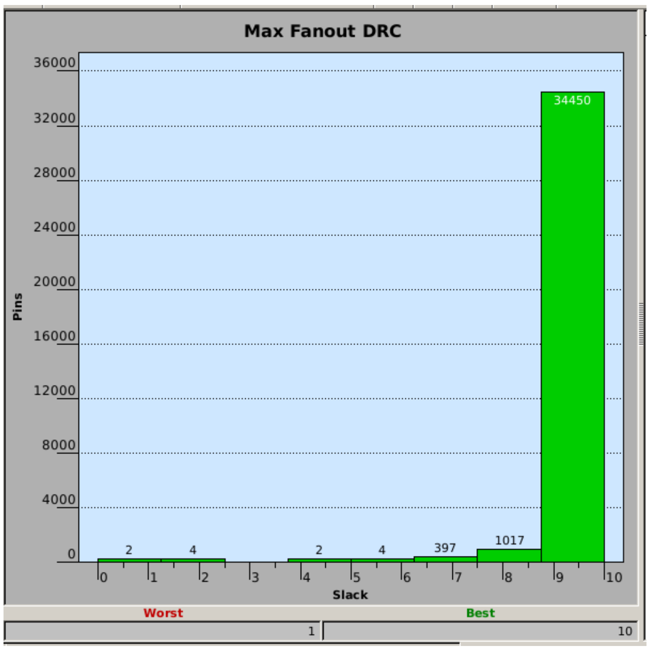


Figure 4.117: Max Fanout Histogram at ss0p7v125c case in Hierarchical Flow

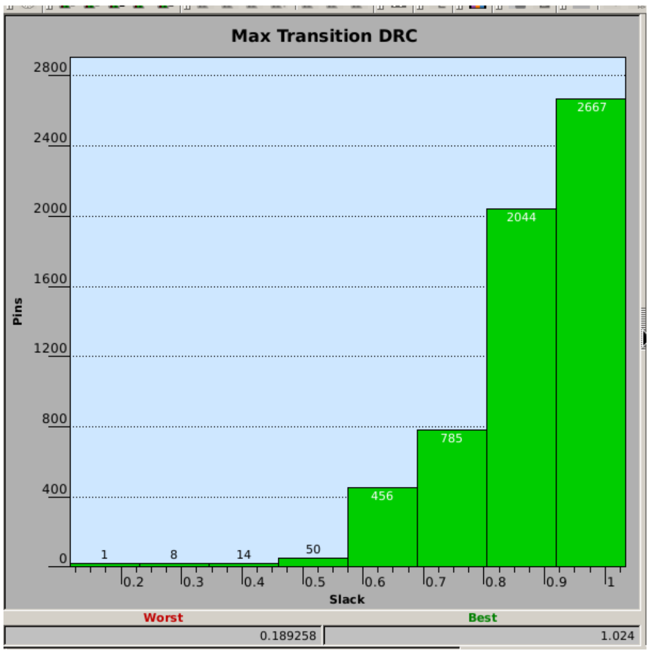


Figure 4.118: Max Transition Histogram at ss0p7v125c case in Hierarchical Flow

figure 4.119 show the power distribution across the blocks . it shows that the id stage is the maximum block that consume total power. Power threshold in the bottom of the figure means the power associated with the block that contributes mostly to the total power which is id stage as shown, other blocks'power is calculated as a percentage of this threshold. We can see that distribution is also by block's occupied area.the power corresponding to each block as shown in table 4.27.

Block Name	Power relative to threshold(W)	Power relative to threshold(%)
id_stage	0.000751705	98
ex_stage	0.000439964	57
if_stage_i	0.000411939	54
cs_registers	0.000269615	35
LSU	6.26743e-05	8

Table 4.27: Power Design Map Statistics Table at ss0p7v125c in Hierarchical Flow

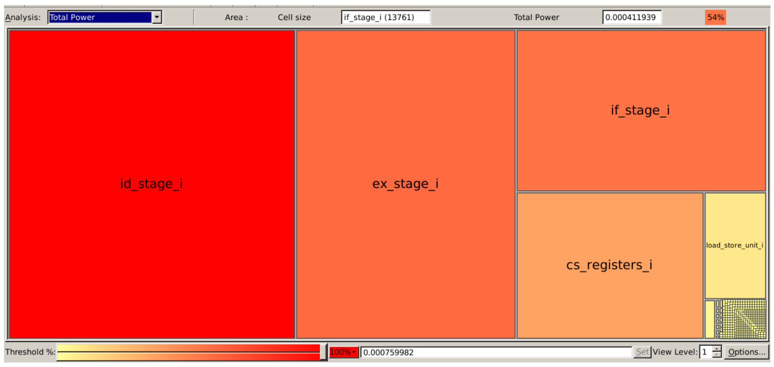
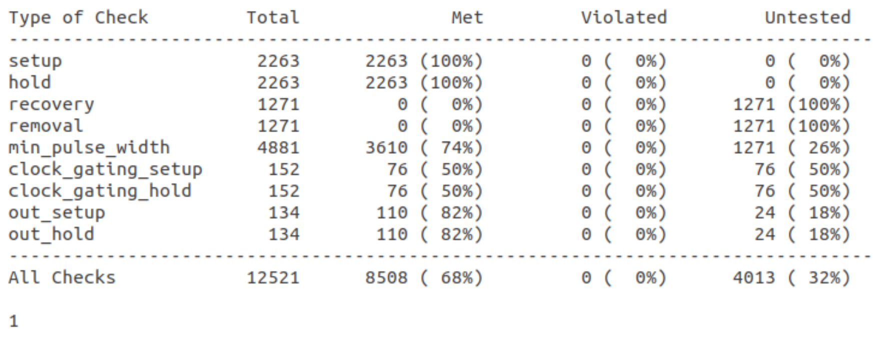


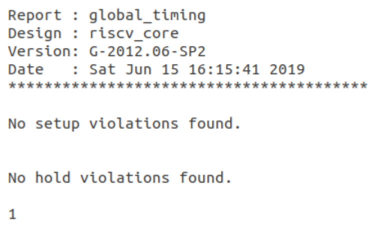
Figure 4.119: Power Design Map at Slow-Slow Corner in Hierarchical Flow

figure 4.120 , show the timing analysis coverage about the design and there are no violations.



```
Type of Check           Total              Met         Violated         Untested
--------------------------------------------------------------------------------
setup                    2263       2263 (100%)         0 (   0%)         0 (   0%)
hold                     2263       2263 (100%)         0 (   0%)         0 (   0%)
recovery                 1271          0 (   0%)         0 (   0%)      1271 (100%)
removal                  1271          0 (   0%)         0 (   0%)      1271 (100%)
min_pulse_width          4881       3610 ( 74%)         0 (   0%)      1271 ( 26%)
clock_gating_setup        152         76 ( 50%)         0 (   0%)        76 ( 50%)
clock_gating_hold         152         76 ( 50%)         0 (   0%)        76 ( 50%)
out_setup                 134        110 ( 82%)         0 (   0%)        24 ( 18%)
out_hold                  134        110 ( 82%)         0 (   0%)        24 ( 18%)
--------------------------------------------------------------------------------
All Checks              12521       8508 ( 68%)         0 (   0%)      4013 ( 32%)

1
```

Figure 4.120: Analysis Coverage Report at ss0p7v125c in Hierarchical Flow



```
Report : global_timing
Design : riscv_core
Version: G-2012.06-SP2
Date   : Sat Jun 15 16:15:41 2019
****************************************

No setup violations found.


No hold violations found.

1
```

Figure 4.121: Global Timing Report at ss0p7v125c in Hierarchical Flowl

Best case Results

in figures 4.122 and 4.123 we show the histogram of both hold and setup time slack and there are no violations as illustrated.

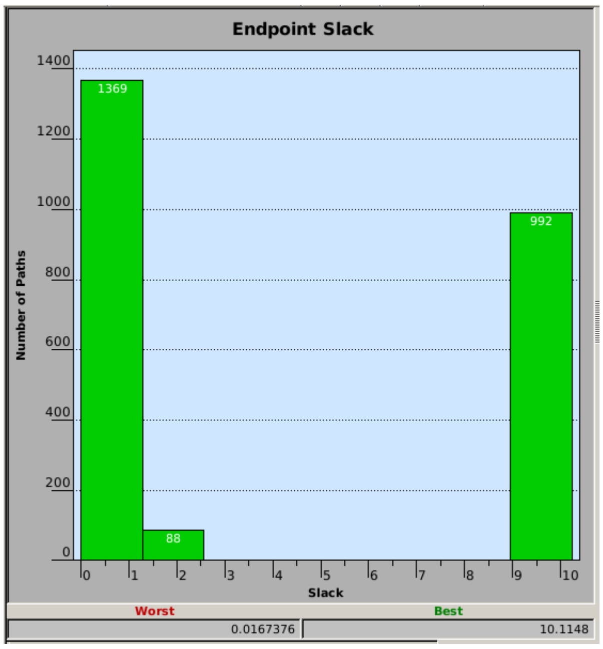


Figure 4.122: Hold timing histogram for ff0p85v125c in Hierarchical Flow

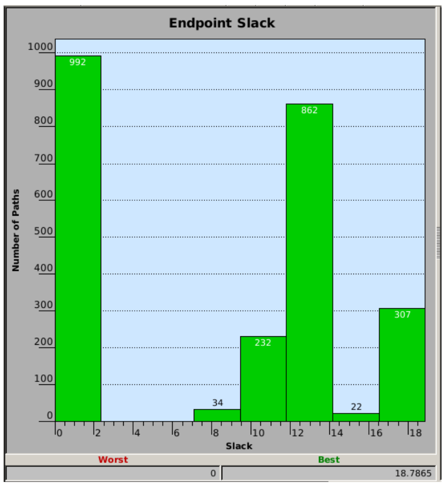


Figure 4.123: Setup timing histogram for ff0p85v125c in Hierarchical Flow

figure 4.127 show the power distribution across the blocks . it shows that the id stage is the maximum block that consume power. Power threshold in the bottom of the figure means the power associated with the block that contributes mostly to the total power which is id stage as shown, other blocks'power is calculated as a percentage of this threshold. We can see that distribution is

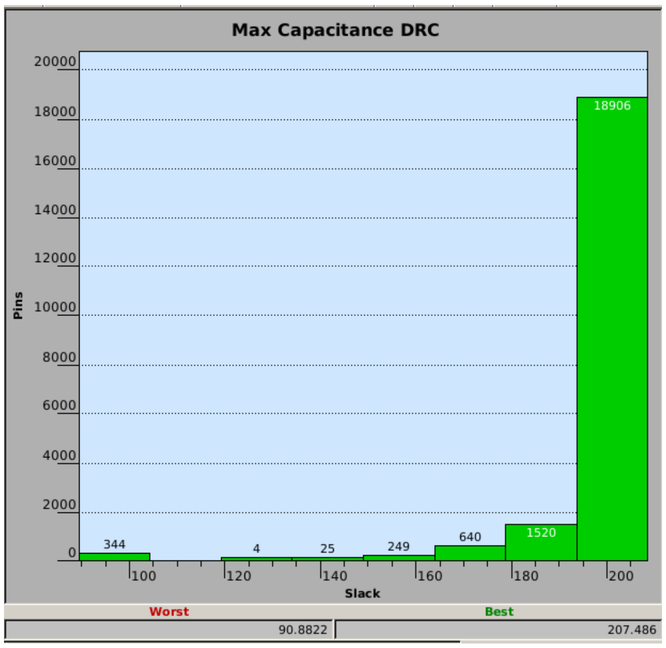


Figure 4.124: Max Capacitance Histogram at ff0p85v125c case in Hierarchical Flow

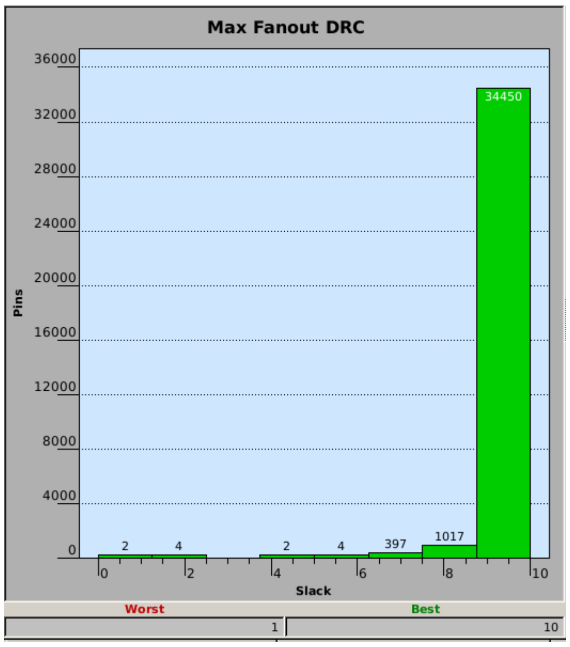


Figure 4.125: Max Fanout Histogram at ff0p85v125c case in Hierarchical Flow

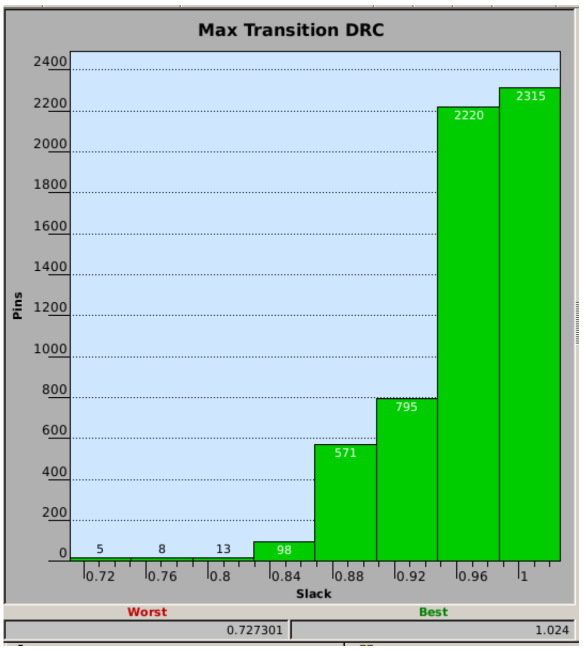
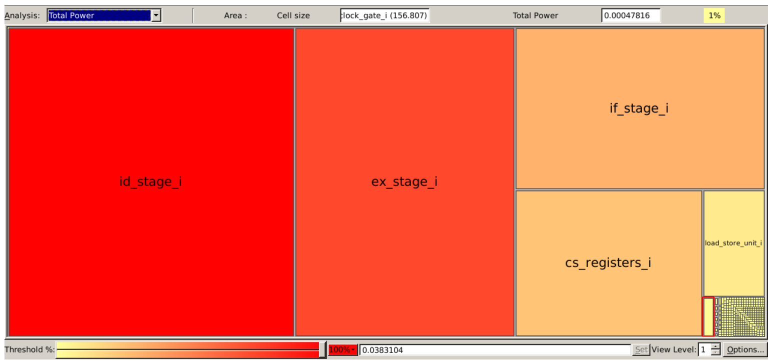


Figure 4.126: Max Transition Histogram at ff0p85v125c case in Hierarchical Flow

also by block's occupied area. the power associated with each block is shown in table 4.28


Analysis: Total Power
Area : Cell size clock_gate_i (156.807)
Total Power 0.00047816 1%

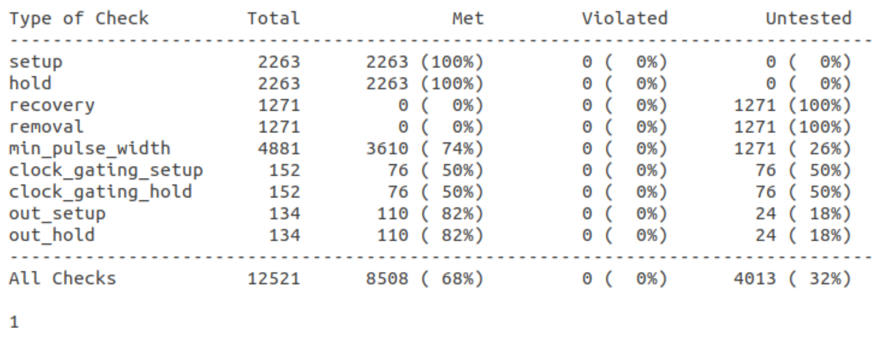
Threshold %: 100% 0.0383104 Set View Level: 1 Options...

Figure 4.127: Power Design Map at Fast-Fast Corner in Hierarchical Flow

Block Name	Power relative to threshold(W)	Power relative to threshold(%)
id_stage	0.0383104	100
ex_stage	0.027634	72
if_stage_i	0.0116358	30
cs_registers	0.00898035	23
LSU	0.00316329	8

Table 4.28: Power Design Map Statistics Table of Fast-Fast corner in Hierarchical Flow

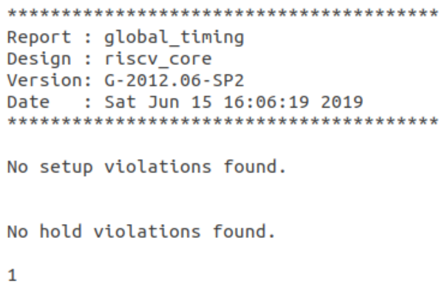
figure 4.128 , show the timing analysis coverage about the design and there is no violation.



```
Type of Check         Total          Met         Violated        Untested
--------------------------------------------------------------------------------
setup                  2263      2263 (100%)        0 (  0%)        0 (  0%)
hold                   2263      2263 (100%)        0 (  0%)        0 (  0%)
recovery               1271         0 (  0%)        0 (  0%)     1271 (100%)
removal                1271         0 (  0%)        0 (  0%)     1271 (100%)
min_pulse_width        4881      3610 ( 74%)        0 (  0%)     1271 ( 26%)
clock_gating_setup      152        76 ( 50%)        0 (  0%)       76 ( 50%)
clock_gating_hold       152        76 ( 50%)        0 (  0%)       76 ( 50%)
out_setup               134       110 ( 82%)        0 (  0%)       24 ( 18%)
out_hold                134       110 ( 82%)        0 (  0%)       24 ( 18%)
--------------------------------------------------------------------------------
All Checks            12521      8508 ( 68%)        0 (  0%)     4013 ( 32%)

1
```

Figure 4.128: Analysis Coverage Report at ff0p85v125c case in Hierarchical Flow

figure 4.129 show the global timing report and there are no violations .



```
****************************************
Report : global_timing
Design : riscv_core
Version: G-2012.06-SP2
Date   : Sat Jun 15 16:06:19 2019
****************************************

No setup violations found.


No hold violations found.

1
```

Figure 4.129: Global timing Report at ff0p85v125c case in Hierarchical Flow

Typical case Results

in figures 4.130 and 4.131 we show the histogram of both hold and setup time slack

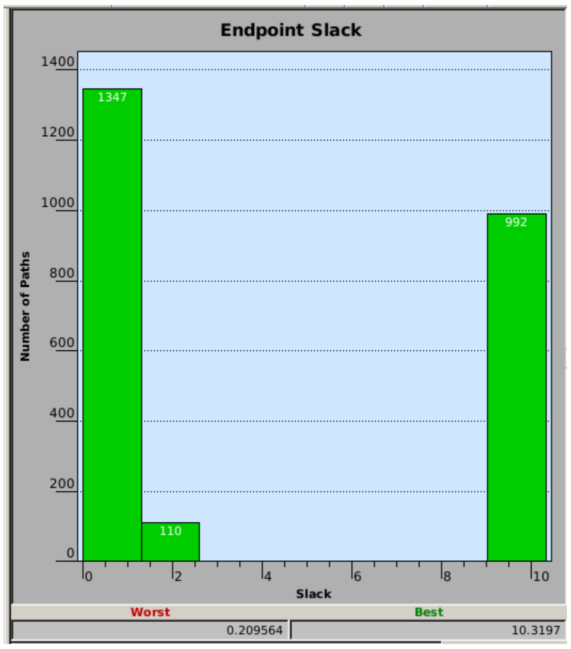


Figure 4.130: Hold timing histogram at tt0p78v25c case in Hierarchical Flow

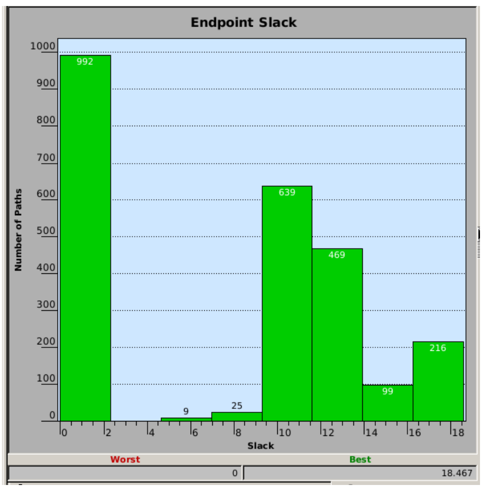


Figure 4.131: Setup timing histogram at tt0p78v25c case in Hierarchical Flow

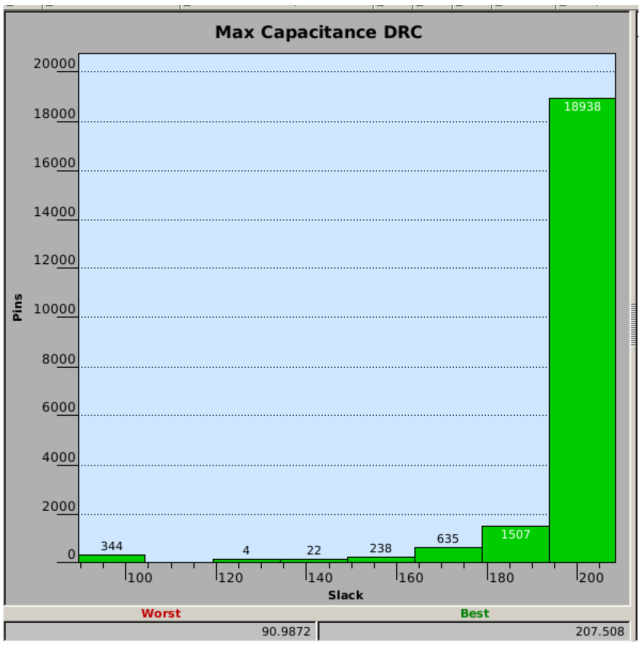


Figure 4.132: Max Capacitance histogram at tt0p78v25c case in Hierarchical Flow

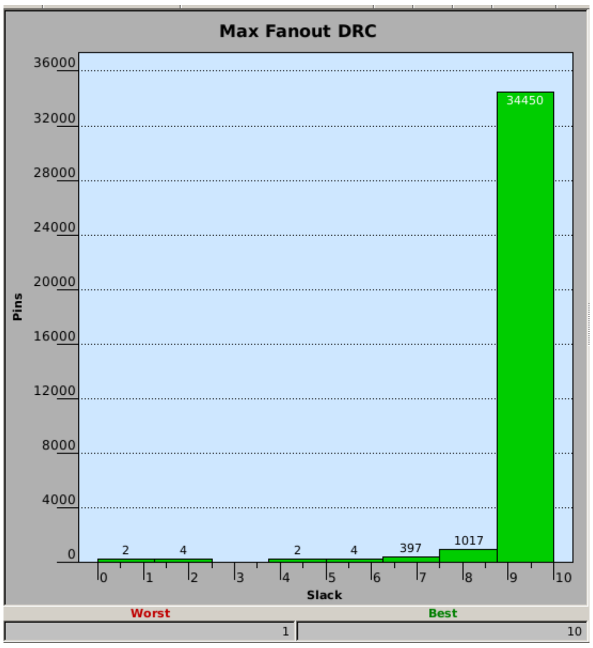


Figure 4.133: Max Fanout histogram at tt0p78v25c case in Hierarchical Flow

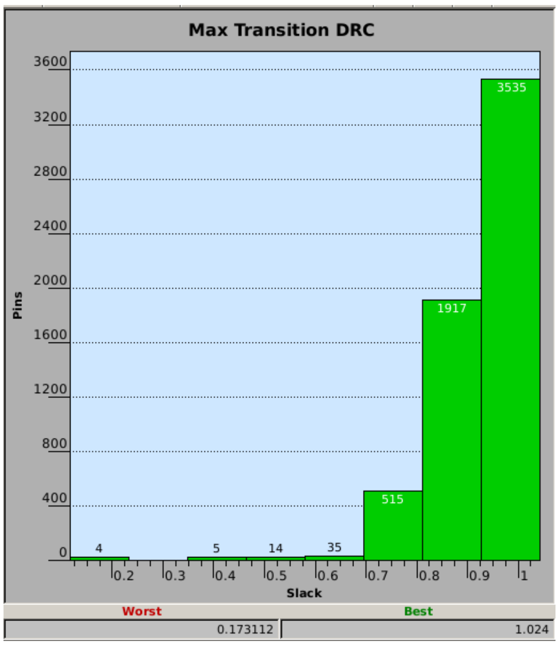


Figure 4.134: Max Transition histogram at tt0p78v25c case in Hierarchical Flow

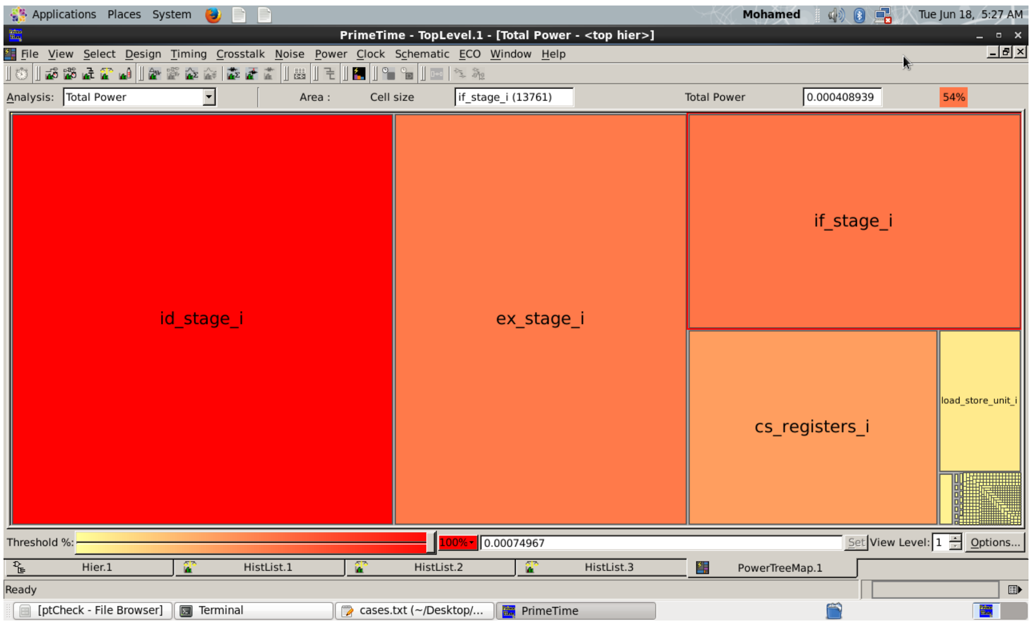
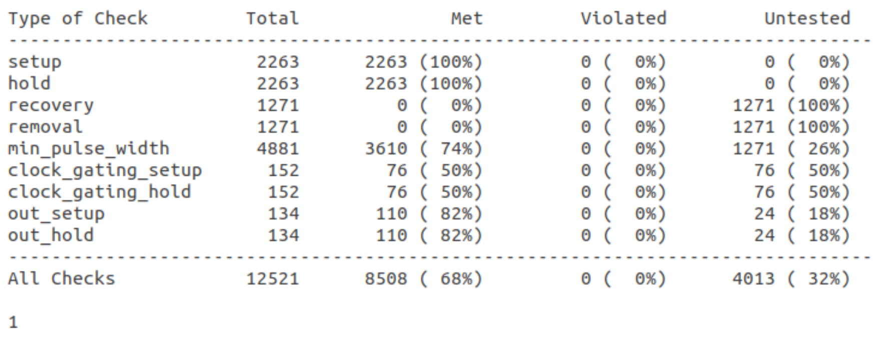


Figure 4.135: Power Design Map at Typical case in Hierarchical Flow

figure 4.136 , show the timing analysis coverage about the design and there is no voilation.

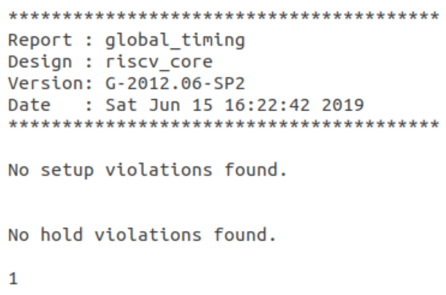
Block Name	Power relative to threshold(W)	Power relative to threshold(%)
id_stage	0.00074967	100
ex_stage	0.000394525	52
if_stage_i	0.000408939	54
cs_registers	0.000287508	38
LSU	6.41922e-05	8

Table 4.29: Power Design Map Statistics table of typical-typical case in Hierarchical Flow



```
Type of Check         Total             Met        Violated        Untested
---------------------------------------------------------------------------
setup                  2263      2263 (100%)        0 (  0%)        0 (  0%)
hold                   2263      2263 (100%)        0 (  0%)        0 (  0%)
recovery               1271         0 (  0%)        0 (  0%)     1271 (100%)
removal                1271         0 (  0%)        0 (  0%)     1271 (100%)
min_pulse_width        4881      3610 ( 74%)        0 (  0%)     1271 ( 26%)
clock_gating_setup      152        76 ( 50%)        0 (  0%)       76 ( 50%)
clock_gating_hold       152        76 ( 50%)        0 (  0%)       76 ( 50%)
out_setup               134       110 ( 82%)        0 (  0%)       24 ( 18%)
out_hold                134       110 ( 82%)        0 (  0%)       24 ( 18%)
---------------------------------------------------------------------------
All Checks            12521      8508 ( 68%)        0 (  0%)     4013 ( 32%)

1
```

Figure 4.136: Analysis Coverage Report at tt0p78v25c case in Hierarchical Flow



```
****************************************
Report : global_timing
Design : riscv_core
Version: G-2012.06-SP2
Date   : Sat Jun 15 16:22:42 2019
****************************************

No setup violations found.


No hold violations found.

1
```

Figure 4.137: Global Timing Report at tt0p78v25c case in Hierarchical Flow

Chapter 5

Conclusion

In this thesis a complete RTL-to-GDSII flow for the OpenPULP Core known as RI5CY was performed, with three different synthesis styles which yield to three different flow results, objective of this thesis is to help ASIC designers choose the most suitable flow which satisfy their requirements. Next, we shall determine the key benefits of each flow compared to other flows and its weakenesses against other flows. Below we will conclude our results in both post-synthesis stage of the implementation flows and post-layout stage of them, also we will compare our post-layout results with ARM Cortex-M4 and RI5CY cores from reference [6], taking into consideration several factors that may affect validity of the comparison. Future ideas are then discussed that would improve design performance in all implementation flows.

5.1 Post-Synthesis Results

In post-synthesis stage as shown in TABLE 5.1, we can notice that both Hierarchical flow and Topographical flow have nearly the same results in cell area, while all flows have nearly the same total power. Hierarchical flow has achieved the minimum synthesis runtime and next to it comes Topographical flow. However, we will distinguish key benefits of each flow later on after actual layout is made for the core. Below charts that demonstrate the results between flows in post-synthesis stage.

	Flat Flow	Topographical Flow (second pass)	Hierarchical Flow (second pass)
Critical Path Delay (ns)	23.04	26.45	17.66
Cell Area (um^2)	65878.953995	59444.027710	58692.269822
Total Power (mW)	1.134	1.125	1.182
Synthesis Runtime (min)	65	7	2

Table 5.1: Post-synthesis Results Table

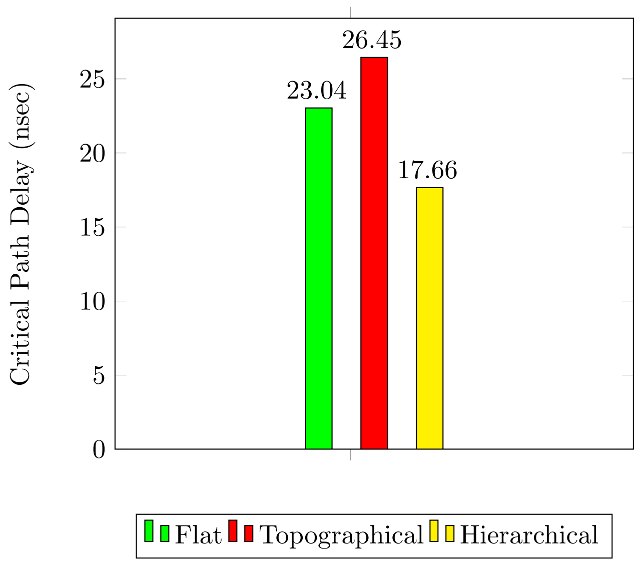


Figure 5.1: Post-synthesis Critical Path Delay

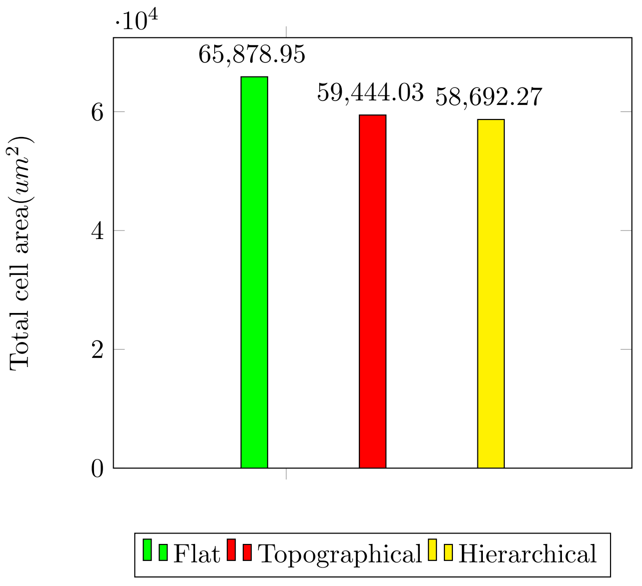


Figure 5.2: Post-synthesis Cell Area

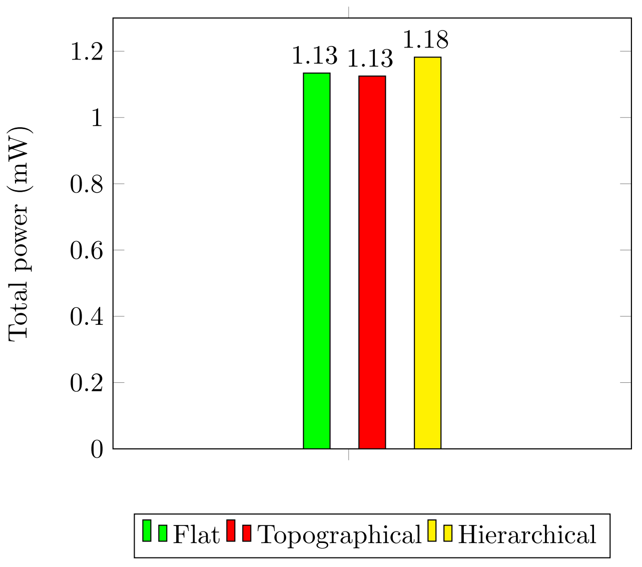


Figure 5.3: Post-synthesis Total Power

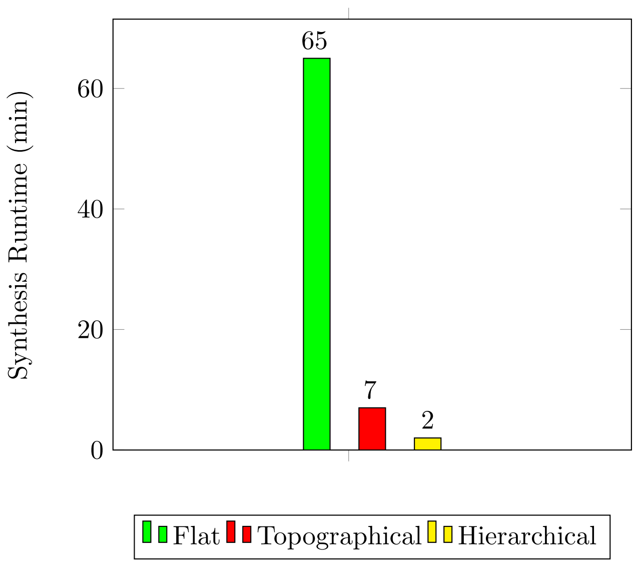


Figure 5.4: Synthesis Runtime

5.2 Post-layout Results

Parameter		Flat Flow	Topographical Flow	Hierarchical Flow
Critical Path Delay (nsec)		21.27	25.44	18.75
Area	Cell area (um^2)	73594	80973	79335
	Physical area (mm^2)	0.1397	0.1136	0.1576
Total Power (mW)	ss0p7v125c	1.5	1.782	1.789
	ss0p75v125c	1.709	1.919	1.99
	ss0p95v25c	1.164	1.514	1.646
	ss0p95v125c	3.629	3.969	3.968
	ss0p95vn40c	0.9375	1.286	1.339
	tt0p78v25c	1.633	1.863	1.978
	tt0p78v125c	6.564	6.564	6.934
	tt0p78vn40c	0.992	1.358	1.421
	tt0p85v25c	2.249	2.592	2.621
	tt0p85v125c	9.036	9.641	9.7
	tt0p85vn40c	1.13	1.31	1.5
	tt1p05v25c	6.067	6.751	6.527
	tt1p05v125c	25	25.8	25.9
	tt1p05vn40c	2.185	2.493	2.617
	ff0p85v25c	29	30.4	30.9
	ff0p85v125c	88.6	90.6	91.5
	ff0p85vn40c	11.1	11.5	12.1
	ff0p95v25c	54.2	56	57.7
	ff0p95v125c	145.2	149.1	150
	ff0p95vn40c	22.8	23.4	24.7
	ff1p16v25c	170.1	175.8	179.9
	ff1p16vn40c	84	86.6	90.8

Table 5.2: Post-layout Results Table

TABLE 5.2 shows post-layout results in addition to the total power calculated at each PDK corner of the verified corners, including the corner which is used for actual implementation of the core in all flows which is (Slow-Slow / 0.7 V / 125 ˚C). Below are charts that help visualize these results.

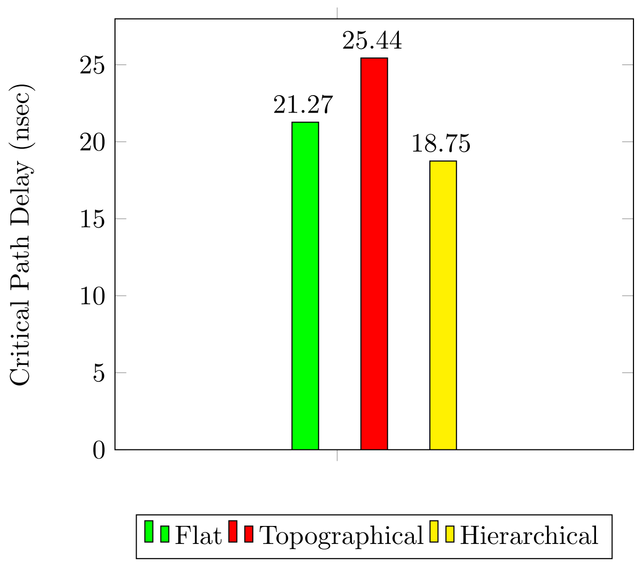


Figure 5.5: Post-layout Critical Path Delay

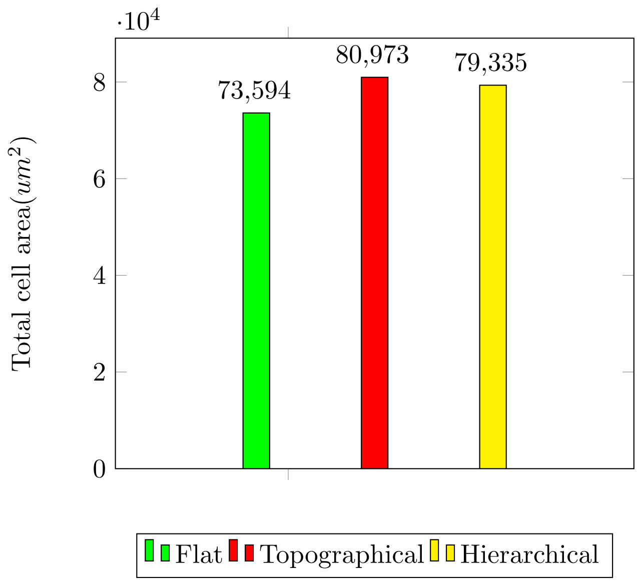


Figure 5.6: Post-layout Cell Area

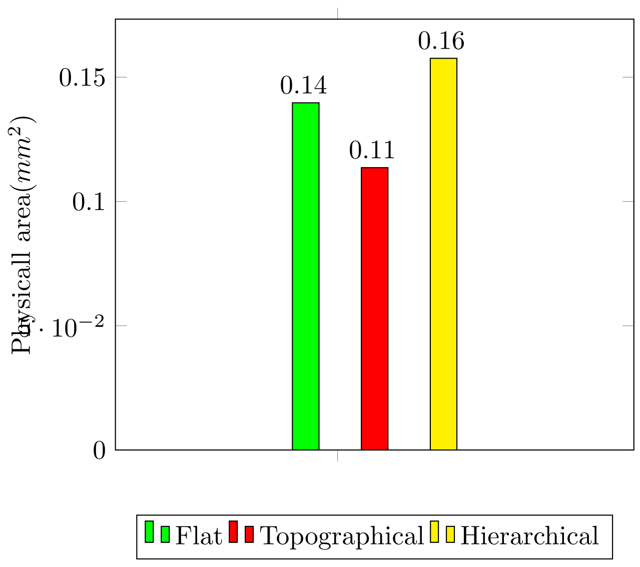


Figure 5.7: Post-layout Physical Area

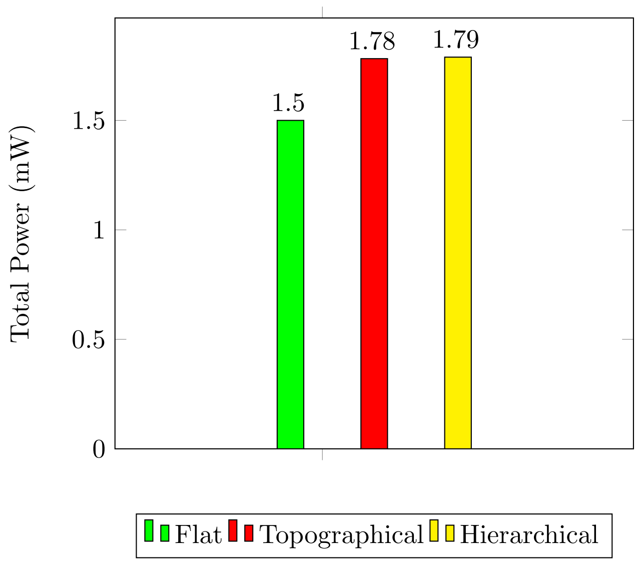


Figure 5.8: Post-layout total Power at the worst case corner

Parameter	Flat	Topographical	Hierarchical
Synthesis runtime(min)	65	7	2
Physical Area(mm^2)	0.1397	0.1136	0.1576
Power(mW)	1.5	1.782	1.789

Table 5.3: Flows Comparison Table

TABLE 5.3 summarizes the key results that distinguish each flow. We can see that Hierarchical flow has the minimum synthesis runtime, this comes from the concept of "Divide and Conquer" as we manage to divide the hierarchy into several blocks. Then we perform parallel block runs to accelerate logic synthesis, then we integrate these blocks and perform another iteration of top-level synthesis. Therefore, we can consider the actual runtime is the sum of the longest block runtime in addition to top-level synthesis runtime, which is found to be two minutes in our case. This may not be an important and critical factor with our proposed design, but it could be crucial to minimize such a factor when working with large designs such as working with the OpenPULP itself or a larger design to meet the required TTR.

An advantage of working with Topographical flow is the highest possible chip utilization that is achieved with 70% , unlike other flows such as Flat flow that worked with a utilization of 60% and Hierarchical flow with a utilization of 50%. This is achieved with the help of congestion-aware routing estimates that highly eases the routing task with highest possible chip utilization, although Hierarchical flow is based on topographical mode runs also, plan groups that are created between blocks result in an increase in the routing congestion which makes routing task a quite hard with higher chip utilization percentages. Therefore, we can notice that Topographical flow achieved the minimum physical area as found in TABLE 5.3.

From the perspective of total power consumption, Flat flow has minimized total power consumption compared to other flows at the worst case corner. Flattening all levels of hierarchy except only the first level which is directly below the top-level has led us to the best possible logical optimization, which highly reduces the total power consumed by the standard cells due to the reduction in the total cell area in the design.

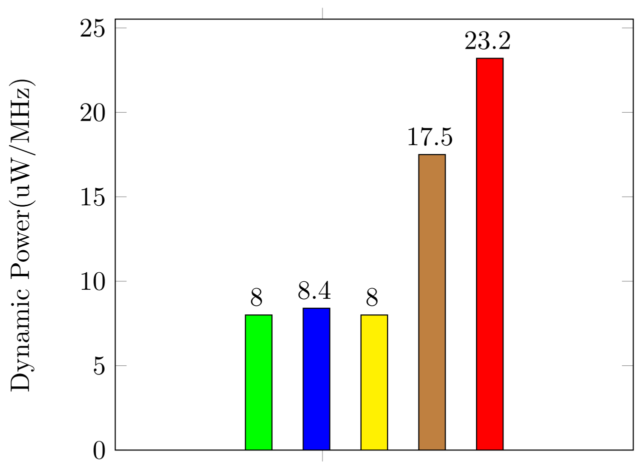
Therefore, if an ASIC designer might want to achieve minimum possible TTR, we strongly advice him/her with working with Hierarchical flow. If the goal is to minimize occupied area as much as possible, Topographical flow is the best choice in that case. And if the goal is to reduce the total power consumed by the design, then it is preferred to perform the implementation with Flat flow to achieve highest cell area optimizations.

5.3 RI5CY vs. ARM Cortex-M4

TABLE 5.4 shows the comparison between ARM Cortex-M4 and RI5CY cores in reference [6] with RI5CY core implemented with our three different implementation flows. Several factors would affect this comparison such as voltage supply and technology. We can see that our results of RI5CY core mentioned in TABLE 5.4 are under different technology and operating conditions compared to the cores in reference [6]. Here we chose the most similar corner to the conditions of the cores in reference [6] under comparison, which is (Typical-Typical / 1.05 V / 25°C).

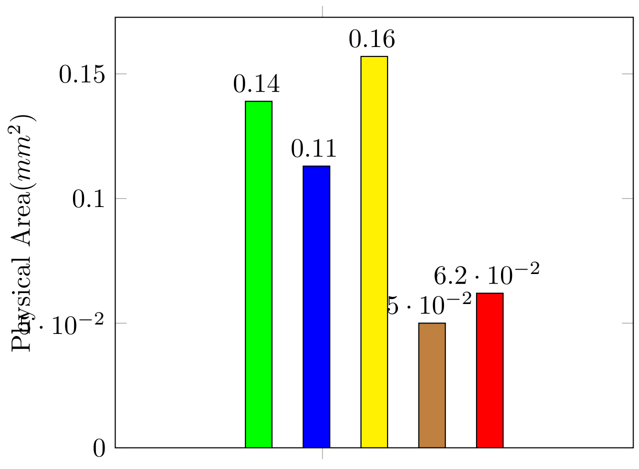
	RI5CY(Flat)	RI5CY(Topo)	RI5CY(Hier)	RI5CY [6]	ARM Cortex-M4 [6]
Technology	32	32	32	65	65
Conditions	1.05 V, 25°C	1.05 V, 25°C	1.05 V, 25°C	1.2 V, 25°C	1.2 V, 25°C
Dynamic Power(uW/MHz)	8	8.4	8	17.5	23.2
Physical Area(mm^2)	0.139	0.113	0.157	0.05	0.062

Table 5.4: RI5CY Core vs. ARM Cortex-M4 Table



🟩 RI5CY(Flat) 🟦 RI5CY(Topographical) 🟨 RI5CY(Hierarchical) 🟫 RI5CY [6] 🟥 ARM Cortex-M4 [6]

Figure 5.9: RI5CY vs. ARM Cortex-M4 in Normalized Dynamic Power



🟩 RI5CY(Flat) 🟦 RI5CY(Topographical) 🟨 RI5CY(Hierarchical) 🟫 RI5CY [6] 🟥 ARM Cortex-M4 [6]

Figure 5.10: RI5CY vs. ARM Cortex-M4 in Physical Area

The dynamic power by definition is proportional to the switched load capacitance, the core frequency, and the square of the voltage supply. Therefore, a scaling to the voltage supply from 1.2 Volt to 1.05 Volt will reduce the normalized dynamic power of ARM Cortex-M4 to be 17.762 uW/MHz and RI5CY in reference [6] by 13.39 uW/MHz at 65nm technology, technology node scaling from 65nm to 32nm also plays an important role in reducing normalized dynamic power by reducing the load capacitance.

Physical area is mainly affected by technology. Therefore, technology scaling is needed to demonstrate the physical area difference between the cores in reference [6] and RI5CY core implemented by all flows. Also switching from SAED 32nm to another manufacturer at the same node would highly reduces the physical area for RI5CY core by the three flows, that is mentioned in TABLE 5.4.

5.4 Future Work

In this section, we propose some ideas that could enhance design performance and speed under all implementation flows. Below we shall describe these ideas in more details.

5.4.1 Place and Route design using MCMM flow

A solution to the problem that not all PDK corners are verified and not passing STA cleanly without DRC/timing violations in our three implementation flows is to perform Place&Route flow using MCMM flow, synopsys IC Compiler offers this flow so that ASIC designers could optimize their design over several corners within both function and test modes, this flow results in a more optimized design which is aware of potential timing/DRC violations that may appear on different corners while running post-layout STA.

5.4.2 Switch from RVT Cells to HVT Cells

HVT standard cells have less leakage power than RVT cells, but this comes with the penalty with an increase in cells propagation delay. As we can see the part that mostly contribute to the total power in all flows is the cell leakage power, reducing this part will significantly affect the total power by reducing it.

Appendix A

RTL-to-GDSII Flow

A.1 RTL Logic synthesis

A.1.1 Introduction to Logic Synthesis

What is RTL?

A brief Introduction In the beginning, the digital designs were performed by simple schematic diagrams, describing the logic functionality, and the hierarchical levels of the design. Then, these designs were being implemented by using the printed circuit board (PCB) technology. By the time, the digital designs were rapidly increasing in complexity, and the implementation of designs using this methodology became more difficult, and non-reliable. Moreover, the performance of the digital circuits, including timing, area and power, degraded with a remarkable rate. Therefore, the need for a new technology was a must.

HDLs The invention of FPGA and ASIC-proven technologies was the double-handed aid, and the rise of a new era for digital world that we now are witnessing. The key entrance for these two technologies was the HDL which is the acronym for Hardware Description Languages.

Hardware description languages are simply describing the behavior of the digital circuits through a software code. Each basic element is the digital circuit (e.g. and, or, not) can be individually implemented with that code. At the level of circuits, this code can describe the circuit either with element-by-element, or functional blocks in a hierarchical diagram manner.

There are two widely used HDLs in the world of digital design, and they are VHDL and Verilog. Each of them has its advantages and disadvantages over the other. Anyway, we can take an example to demonstrate the implementation of a basic logic circuit with Verilog HDL.

```
module SIMPLE_LOGIC(A, B, C, D, F);
input A, B, C, D;
output F;
assign F = ((A & B) — (C & D));
endmodule
```

Regardless to the language syntax, this code logically describes the basic operation of a digital circuit. By tracking and tracing the code, its function is to construct a logic circuit with four inputs and single output. The relation between the inputs and the output is shown in the fourth line. Each symbol in this relation annotates an explicit logical operation (e.g. ‘&’ symbol maps ANDing operation, and ‘—’ maps ORing, and ‘ ’ maps NOT). The braces are used for separating the logical operations as any programming language.
By the same manner, any digital circuit, involving combinational and sequential circuits, could be built using these HDLs. After that, that code can ride over any flow either FPGA or ASIC.

RTL By using the key concept of HDLs and its functional utility, we can build our RTL with close eyes. But firstly, what is the RTL?

In digital design, the design can be split into top-down-based or bottom-up-based layers, and that depends on how it would be looked at.

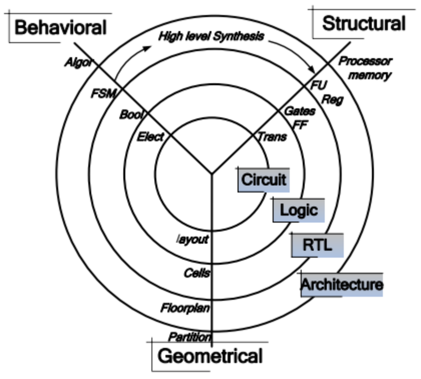


Figure A.1: Gajski Y chart

The register transfer level, or shortly RTL, defined as the level at which the data, control signals transfer could be seen and handled by the designer using the hardware description language in the level of registers and functional blocks.

The "Gajski Y chart", shown in Figure A.1, illustrates the layering process of any digital circuit design from three angles of view. In our case, we need to know the location of RTL among the different layers and views. As illustrated, the RTL is located at the third place from the bottom. That means it covers the circuit levels in the registers and functional units.

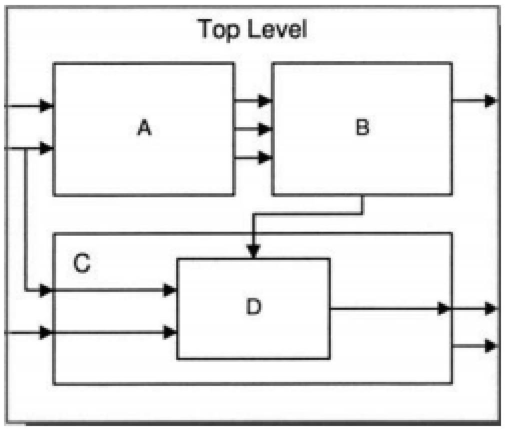


Figure A.2: Top-level view of RTL

As illustrated in FigureA.2, the figure shows the top-level view of the RTL, containing the hierarchical blocks as known from definition.

Code-to-logic synthesis

Introduction After writing the HDL code and verifying it from syntax and logical errors, the next step is to synthesizing the code, so what is the meaning of code-to-logic synthesis?
In the past time, there was a one-step design to reveal the actual implementation into the light. But now, the designs are written in a software code, thus they need to be realized as done in the past. In other words, we can extract the definition of logic synthesis from the previous words. Therefore, the logic synthesis is simply the process of converting the coded RTL representation into a real hardware digital logic.

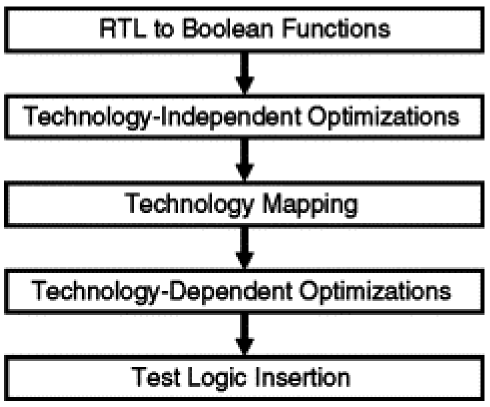

RTL to Boolean Functions
Technology-Independent Optimizations
Technology Mapping
Technology-Dependent Optimizations
Test Logic Insertion

Figure A.3: Top-level view of RTL

Synthesis process flow Logic sythesis is not limited to translate only the HDL code into the real logic, but it involves five main sub-processes as illustrated in Figure A.3. The first one is to directly translate the HDL code into its equivalent boolean functions (e.g. F = AB+C) to be prepared for technology-independent optimiztions. Before stepping forward to the next sub-process, we need to take a break and understand the of meaning of the technology of the digital logic. The technology, as a general view, means simply a multi-level ladder at which we can attain a higher level of performace in a multi-aspect view. In our case of study, the techology of digital logic implys a triangle with three ribs which are area, power and delay. Therefore, the technology-indendent optimizations in the second step means to perform logical optimizations such as minimizing the the number of logic gates and registers using the common methods like karnough map and boolean algebra, and all these optimization techniques are preformed indpendently on the technology metrics which are area, power and timing.

After that, we can apply the technology to the optimized logic so that we can measure the real metrics for the design in terms of area, timing and power. We will talk in detail about technology libraries in "Standard EDA-based synthesis flow" section. However, we can briefly say that the technology package includes the timing information based on the input transition and output capacitance, and the area calculated based on the actual area of the cell in that certain technology. Moreover, the leakage power is typically measured for that cell and so the power needed for proper operation, in addition to wire-load models delays.

After mapping the target technology to our functionally optimized logic, the design may need more or less further optimizations to meet the given standard technical specifications based on a certain technology. These specifications are about our three major components that were mentioned before.

We are now possessing the information of the design about timing, area and power for our specified technology. Therefore, we can optimize the design to achieve clean setup and hold timing. Setup timing optimization can be concisely done by reducing the delays of the timing paths, and that can be achieved through upsizing the logic gates and replacing combination of gates with another has the same function but has less delay. Although up-sizing the gate can enhance the setup timing, but it's not all the way. As we have to consider that the up-sized gate increases the capacitance of it's input, thereby the delay of the previous stage can can be increased and thus the whole path delay. Hold timing optimization can be done by increasing the delay of the timing paths, and that could be achieved by injecting highly buffers with different sizes dependently on the delay status of the timing path. Area is directly propotional to power, thus decreasing the design area will decrease the overall power consumption of the system. The large-sized cells have larger area, and repectively the small-sized cell has smaller area. Consequently, there are a compromise among area and timing. So, we have to balance between them to acquire the best possible results, and to meet the given specifications.

After performing the required optimizations, and the design is ready for being placed on a chip, there is an implicit step to consider, and it has a great significance to perform. The last step is to insert a testing logic device distuributed over the entire network of the design. When the chip have already been manufactured, there is no simulation for verifacation where it's now a hardware piece, and we do not know if the timing paths is logically and physically connected or not. Therefore, we need to insert a scan chain that is built-in in each flip-flop to make a large scale shift register from each input to each output. This topic will also be discussed in detail in its dedicated section.

Need for EDA tools

By the time, the digital designs were growing in scale and complexity, and using the traditional methods for making the entire process of design became competely difficult to achieve. Therefore, the designers thought to find a way to give the life to the digital world again, and that achieved by inventing the electronic design automation (EDA) tools.

The EDA tools is simply performing the same tasks that was perfomed in the past such as performing synthesis, and placing the design into the chip before completely routing it automaticaly, but dut to the complexity of the large scale designs, the manual design became more exhaustive, and time-consuming. Due to these features, the EDA tools were rapidly spreading among the community of designers, and lastly became the main reliable and handful tools in the world of digital design.

There are three main types of EDA tools each has its functions and features, and they are classified as follows: synthesis, place-and-route, and static timing analysis tools. The synthesis tool is taking your RTL design and setting timing, area, and logical constraints on it, and then performing the actual synthesis on it to meet the timing and area requirements. The synthesis process is done by applying the software algorithms to translate, map, optimize the design, and then printing out the final, well-optimizated gate-level netlist.

The second tool is typically taking the logical netlist, and contructing the real physical chip with real dimensions step-by-step using the appropriate algorithms for planing and placing the design into the chip, and then routing it with clean physical design rules.

The last last tool we have is the static timing analysis tool, and it has one main task is to exploit its strong software engine to perform setup and hold timing analysis, in addtion to checking timing violations, and then attempting to fix them.

A.1.2 Standard EDA-based synthesis flow

As we discussed before, the EDA tools have made a revolutionary transition in the world of digital design. Now, we are going to give the synthesis tool its full rights of explanation and detailed discussion due to its favor to the digital designers.

First of all, the synthesis tool can be treated as a multi-input-single-output system, and Figure A.4 shows the information of these inputs and outputs. The inputs comprise of RTL design HDL code, timing and logical constraints, environmental attributes, and at last our technology cell library as an input. The single ouput represents the logical gate-level netlist. Each of them will have its opportunity for discussing. As a starting point, our synthesis flow will begin with technology library.

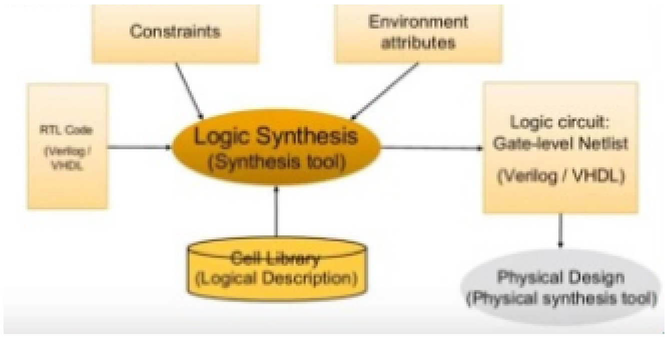

Constraints
Environment attributes
RTL Code (Verilog / VHDL
Logic Synthesis (Synthesis tool)
Logic circuit: Gate-level Netlist (Verilog / VHDL)
Cell Library (Logical Description)
Physical Design (Physical synthesis tool)

Figure A.4: Logic synthesis tool inputs and outputs

Technology database library

One of the dominant reasons for success of logic sythesis process is the existance of logic libraries that is available for various semi-conductor foundries. These libraries facilitates attaining of database information about each logic cell known in the digital world in terms of area, timing, power, and operating conditions. There are two popular commercial formats for library description, and they are Liberty from Synopsys and Advanced Library Format (ALF). The library format can be categorized into two distinct sections. We are going to expose them and their elements in detail.

The first one is technology data. This involves information such as operating conditions (in terms of power supply and temperature) and wire-load models. The wire load model is a statistical estimate for capacitances which is implicity existed in the typical nets as a function of number of pins and size of the design in terms of area. It's used as net capacitances requited for estimating the net delay between two logic cells in the timing path.

The second category in library characterization is the library cell data. Each cell in the library is accurately characterized for timing, area and power. The delay can be estimated from the cell invironmental attributes (i.e. input transition and output load), process technology attributes such as threshold voltage and critical dimensions, in addition to operating conditions such as local power, ground levels and temperature. The timing inforamation also includes constraints such as setup and hold constraints. The power data contains internal switching power and leakage power. It also have the functionality information of each cell.

Timing Modeling The cell timing models are intended to achieve accurate timing models for each cell in the design. They are obtained from detailed circuit simulations of the cell operation. Timing models are specified for each timing arc of the cell.

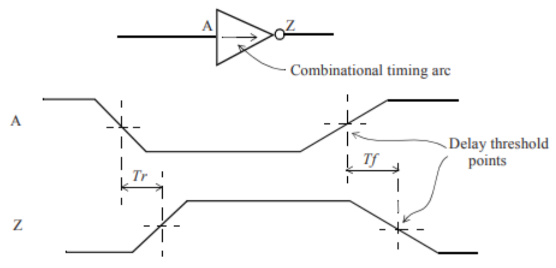


Figure A.5: Timing arc delays for an inverter cell

According to the inverter timing information illustrated in Figure A.5, a rising transition at the input results in a falling transition at the output and vice versa. As shown in the figure, there are two kinds of delays, and they are output rise delay and output fall delay which are annotated by T_r and T_f respectively. These delays are defined based on a threshold points typically measured at 50% Vdd. The delay for the timing arc through the inverter is dependent upon two main factors which are previously mentioned as input transition and output capacitance.

Linear timing modeling The simple delay model, but not accurate one, is considered as a linear function of our two parameters: input transition and output capacitance.

$$D = D0 + D1 * S + D2 * C$$

As shown from the previous relation, the linear delay model is represented in three terms a parameter-independent delay factor D0, and the input transition value multiplied by a constant value D1, and the last term consists of the capacitance value multiplied by a constant value D2. This is a simple delay model to characterize the delay of the cell but the actual used model is a more complex one called non-linear delay model (NLDM).

Non-linear delay modeling The non-linear delay model uses a circuit simulator to characterize the cell's transistors with different of input slew rates and output load capacitances. The results form a two-dimensional table. The cell delay can be addressed from the two previously mentioned factors.

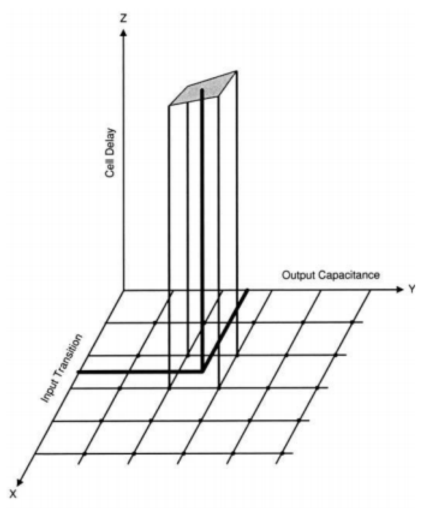


Figure A.6: NLDM Table

The Figure A.6 shows the resulting delays that are interpolated from a certain values of input transition and output capacitance. The modeling accuracy of the delay depends on the precision and range of the simulated values of the input slew and output load. If the delay value is located within a square, as shown in the figure, then it can be calculated using interpolation techniques. On the other hand, if the delay value is located outside the table, then the synthesis tool uses extrapolation techniques to estimate this value.

Power dissipation modeling

Active power The active power is associated with the activity of the input and output pin of the cell. This activity can be defined as the charging and discharging the output capacitance through the drawn current in the CMOS resistances.

There are two types of active power: output switching power and internal switching power. The output switching power is independent on the type of the cell, and it only depends on the output capacitive load and frequency of switching. The internal switching power is dependent upon the type of cell and the activity of the input and output pins. Therefore, the values of internal power can only be found in cell library.

Leakage power The active power, as discussed before, is due the to cell activity. But, if there is a power dissipation without any activity of the cell, it will be only due to non-zero leakage current. The leakage can be due to the sub-threshold leakage current for MOS devices or current tunneling through the gate oxide.

By using high threshold voltage, the phenomenon of subthreshold leakage current can be significantly eliminated. There is a trade-off between the speed of cell and leakage power. When using high 'Vt' cells, the delay of the cell increases, and hence the speed degrades. The gate-oxide tunneling does not change significantly when using high 'Vt' cells. As a result, the leakage power can be specified in the cell library as a default value, and for input-activity conditions.

Wire-load Modeling Wire load models contains information that synthesis tool uses to estimate the values of interconnect net delays during pre-layout phase of the design. There are different models of the wire-loads to be appropriate to different sizes of logic cells. These models define capacitance and resistance and area per unit length of the interconnect wire.

Other attributes in cell library In addition to timing, power information in the cell library, the cell description in the library specified area, functionality.

Constraining the design

In the previous section, we exposed to the concept of logic cell library, and the content of the cell library such as timing and power models and other cell attributes. The synthesis tool needs this technology library as its first step to setup the data for the given design. After establishing the database of the technology, the synthesis tool will ask for explicitly reading the RTL design files. The RTL design that was read has no information about the required timing, area and power where it's technology-independent language and constraints-free design files till now.

The next step is to simply constrain the design in timing, area and power aspects to meet the required design constraints. In this section, we are going to discuss in detail the constraining process of the design from concept and methodology perspective.

Idea behind constraints The mean of 'constraint' word in language is to set a free-space for something to move or a some-time for something to be done within the specified time, thus there are boundaries for each attribute to be constrain with. In the world of digital design, to constrain a design is to give this design the liberty for moving in dimensions of timing, area and power with limiting boundaries, or to get certain constraints that cannot be able to move in that dimension such as constraining the clock with a certain period of time for design to operate at.

The aim of this operation is that the client would like to have a design with specific requirements. In order to achieve that, the design must be constrained in each dimension, and then be optimized.

Timing constraints

Clock attributes and constraints

First of all, design must know its operating frequency as the clock. Therefore, the clock attributes can be defined in the synthesis tool in terms of time period and duty cycle.

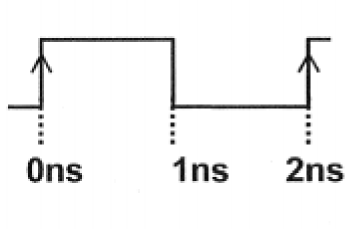


Figure A.7: Defined Clock

Clock definition in most synthesis tools is done in non-realistic form as illustrated in Figure A.7. The clock is defined in ideal form with sharp edges which cannot be achieved in the real designs.

As a result, we need to model the real clock attributes and variations that can be occurred in the real life. Therefore, the need for the actual attributes is a must. The clock attributes that we need to define are transition, latency and uncertainty. Each of them will be discussed in detail.

Clock transition The actual clock is not ideal (i.e. it has no sharp edges) as illustrated in Figure A.8. The reason for that is the circuit of the actual clock source too difficult to be designed to achieve that purpose. As known in the engineering concept, the more quality can be acquired, the more money will be drained. Therefore, the tool can simply define this attribute as a transition time for the clock so that it can work with realistic clock by setting the time of the transition caused by the clock.

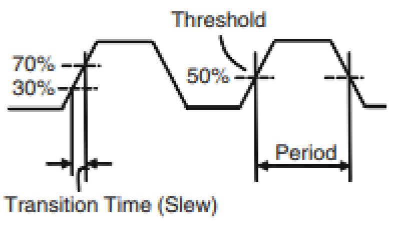


Figure A.8: Clock Transition

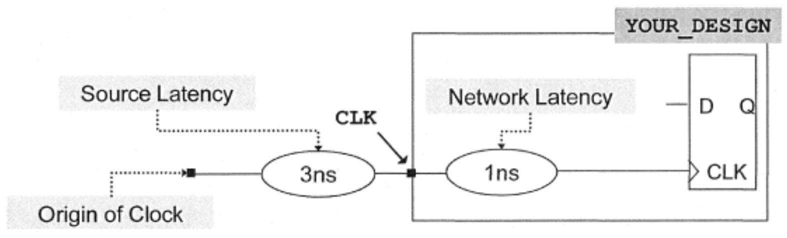


Figure A.9: Clock Latency

Clock latency The Figure A.9 shows the delay in the clock path. The delay between the clock source and the actual pin where the clock is used to trigger the device is defined as clock latency. This delay is due to the capacitive load and the elements existed in the clock tree between the clock source and the clock pin. There are two components in the clock latency which are source latency and network latency. The first one is source latency, and that is the delay between the clock source and the point at which the clock is actually defined. The network latency is the delay from the defined point to the actual device pin that will be triggered. These values are set by the designer in the synthesis tool command line argument, or by setting them in a script file, and then can be sourced from the command line argument.

Clock jitter At the clock generating device, as PLL device, the coming, rising or falling, edge may be not deterministic due to crosstalk or electromagnetic interference or due to PLL characteristics itself. These non-deterministic variations in clock periodicity is noted by clock jitter. As a result, these variations have direct impact on setup and hold timing so that their values will be also changed with respect to the coming non-deterministic edge. As shown in Figure A.10, these variations cause the clock edge to slide in time scale thereby reducing or increasing the required time for meeting the setup and hold requirements. Clock jitter cause lack of predictability to when the exact clock edge will arrive at the point of triggering of the sequential device. These called the clock uncertainty. This phenomenon can be explicitly modeled in the synthesis tool as maximum and minimum predictable values, in unit time, that could be set before and after the expect incoming edge. (e.g. if the edge will arrive at 4, we can define a value of 0.2 for setup to slide the edge backwards to throttle the setup requirement, similarly, we can define a value of 0.1 for hold time to slide the edge forwards to make to throttle the hold requirement).

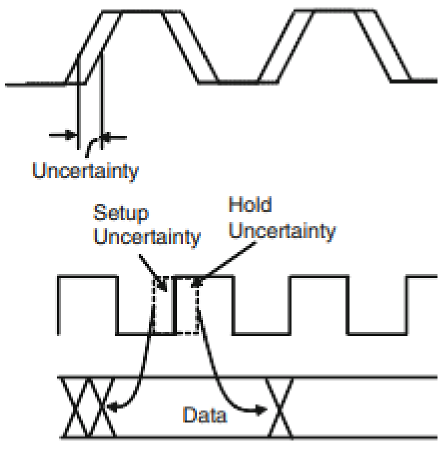


Figure A.10: Clock Uncertainty

Constraining I/O ports

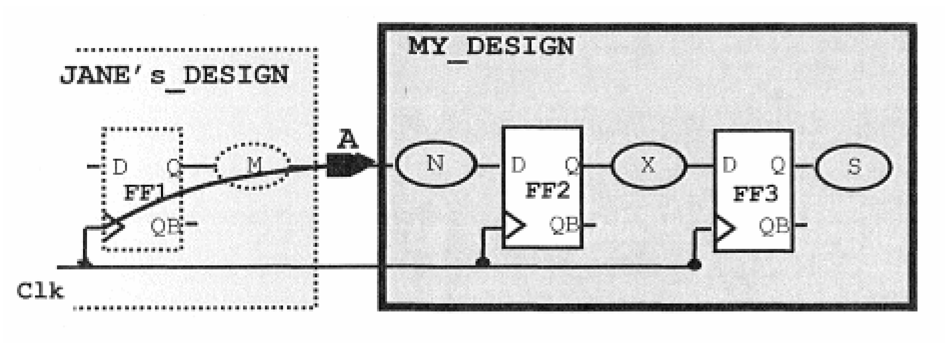


Figure A.11: Input port delay

Input port delays In order to achieve synchronization between different designs, constraining the input and output ports for timing is a must. To make that realizable, the synthesis tool assumes another design attached to the current design with its inputs and outputs.

In our case, as input ports discussion, the JANE's design, as illustrated in Figure A.11, has sequential element and a cloud of logic denoted by 'M', and connected to our design. To achieve synchronization, the two designs must have the same clock. In our design, the input port 'A' attached with a cloud of logic denoted by 'N'.

The time budget of the clock period is then divided between the logic 'M' and logic 'N'. To realize the setup and hold requirements for this virtual path, we need to constrain the input delay coming from the other design. But, unfortunately, we do not have access to the other design, and the only access that we have is our logic delay. Therefore, to achieve the inter-timing requirements between those designs, we need to constrain the outside delay by constraining our inside delay. If we set a delay value for logic 'M', we then impose a value for delay 'N' not to exceed it. Thus, during logic optimization, the synthesis tool then has a certain delay value for 'N' that must do it best to achieve it.

Min/Max input delays As discussed before, the clock time-period budget should be equal to or greater than the sum of internal logic and external logic delays in addition to setup time. Therefore, by setting a maximum value for the input delay, then the internal logic delay is also constrained so as not to exceed the value of clock period subtracted from that maximum input delay value and setup time. So as to decrease the internal logic delay, we have to specify a larger maximum delay value on that input port.

On the other hand, the minimum delay value for the input port is mainly needed for inter-path hold time requirements. We need to specify that value in order to make sure that path has the minimum delay required for hold time. When setting that minimum value to be a certain value, and the delay of the logic 'N' has another certain delay value. If the sum of these two values is less than the time period of the clock cycle, then the synthesis tool will boost the internal logic delay to meet the hold time requirements for this path.

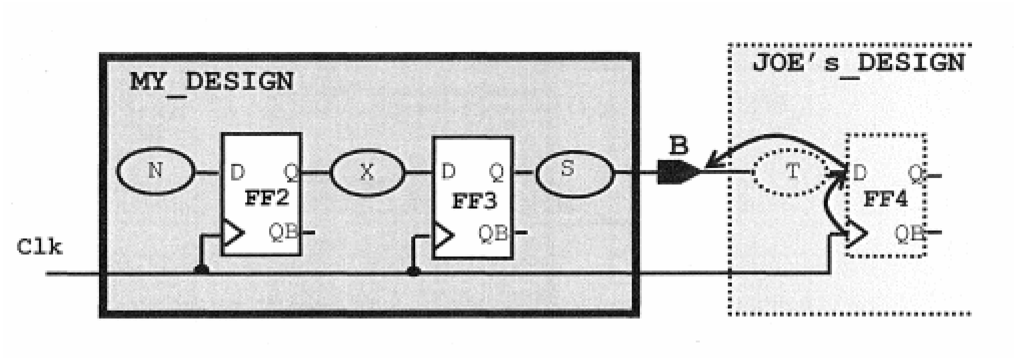


Figure A.12: Output port delay

Output port delays Similarly, the output port delay has the same concept as the input port delay in term of minimum and maximum delays. As illustrated in Figure A.12, the output port 'B' attached to another design input port with a cloud of logic 'T' and input flop 'FF4'. To achieve the maximum output delay, the logic 'S' must be optimized to confine the time into the time period. To achieve minimum output delay, the logic 'S' delay must be boosted to fill the time period.

Environmental Constraints

Output capacitive load

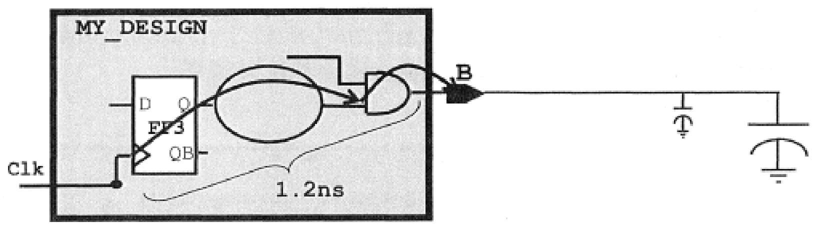


Figure A.13: Output capacitive load

The output load is the actual capacitive load for each output port. That load is existed due to the output parasitic capacitances that is implicitly found in MOS devices. By default, the synthesis tools define the output capacitance to be zero. Therefore, the output transition will have sharp edges which is not realistic at all.

Consequently, one must define an output capacitive value in order to accurately model the actual behavior of the output ports as illustrated in Figure A.13.

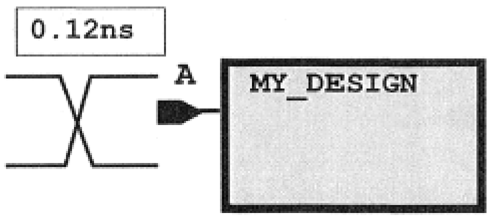


Figure A.14: Input transition to the input port

Input transition Input transition occurs at input ports where the input signals switching from low to high and from high to low. As known, there are no idealities in the life. Therefore, the input signal makes its transition with a finite slope that differs with respect to the type of the driving cell. As illustrated in Figure A.14, the input transition has a finite value of 0.12 ns, and it can be directly modeled in the synthesis tool.

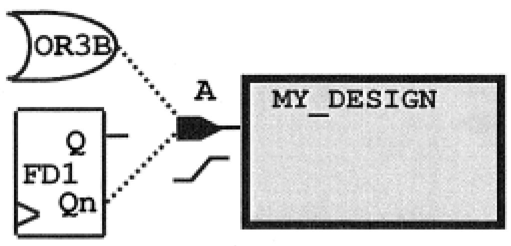


Figure A.15: Driving cells to the input port

Driving cells As discussed before, the input transition can be modeled as a constant value given for a certain input port. But, if a specific transition time value is not known at a block-level input port, one cannot model the transition time with an arbitrary value. Therefore, it must be modeled with an actual driving cell at that input port. As shown in Figure A.15, the input port has been driven by an OR gate or a sequential element. Each has a value to give for the input port to model the transition time. This driving cell can be specified due to the nature of the input port itself, and the driving cell itself that expected to be attached at that port in reality.

Logic optimization

Introduction After the design is well-constrained, it's then ready for being optimized. The timing and logical constraints that applied to the design may be, more or less, not fit to the design in timing, area and power aspects. Therefore, the violations in these three dimensions commence to appear at once as single block of errors that needs to be handled and reduced till being disappeared, and finally one can has a clean timing and a well-confined area and low power-consuming design. Therefore, the only way to achieve that is through logic optimization techniques.

Technology-independent optimization As we discussed before in the basic synthesis flow section, there is a step that is called technology-independent logic optimization. From its name, the logic optimization based on this stage is about optimizing the design without going into any technological aspects. That means no area, no power and no timing information will be set during that optimization. It only be based on the basic logic minimization techniques that satisfy the same logic functionality for the optimized circuit.

Two-level logic minimization

There are different types of two-level logic implementations. The most common one is the SOP implementation where the first level of logic corresponds to AND gates, and the second level refers to OR gates. There are many structures for SOP implementations such as NOR-NOR structures, NAND-NAND structures, OR-AND and AND-XOR structures.

The SOP minimization can be done through the EDA tools through many algorithmic methods such as "Quine-McCluskey" algorithm that it uses the Boolean algebra to minimize the logic in the two-level implementation. The conceptual form for that algorithm can be found in the references section that will be left in this thesis as we do not have to dive into the detailed SOP minimization.

Multi-level logic minimization

The two-level logic implementation used for minimization is limited due to not all logic functions can be implemented in two-level representation. In addition, the multi-level logic is often faster and smaller than the two-level-based logic. Therefore, multi-level logic is preferred in very large-scale designs. There are two basic types of the multi-level logic minimization approaches, rule-based local transformations and algorithmic transformations.

A rule-based local transformation method transforms a pattern for a set of local gates and interconnections into another equivalent pattern when this certain pattern is recognized the logic netlist. These transformations are somehow limited in optimization capability since they are local in nature, and do not have a global perspective of the design.

The algorithmic transformations are evolved in parallel with the activity of two-level-based logic and influenced by it, and it consists of two phases: technology-independent step based on algorithms for manipulating the Boolean logic functions, and a technology mapping step where the design is implemented using generic Boolean functions that can be implemented in design method of choice such as (FPGAs, standard cells or macro cells).

Sequential logic minimization

The basis of the sequential logic is strongly dependent on states so that the logic circuit can transit from one state to another. Therefore, the main minimization purpose should be applied on the number of states in that circuit, hence the number of registers in the circuit can be minimized to realize the same function of that circuit. We can make further logic transformations in the combinational logic that is embedded in the sequential elements. State minimization, state encoding, and logic transformations are using unreachable states or state equivalence as don't cares. These approaches are dedicated to state-based minimization in case that we have to know some information about the given sequential circuit. In contrast, there are structure-based transformations which are carried out according to the circuit structure and do not rely on state information. Retiming and re-synthesis, for example, are practical transformation methods for sequential logic minimization.

Technology-dependent mapping and optimization As discussed before, the technology-dependent mapping and optimization is simply based on a given technology cell library. For a given technology library, the problem of technology mapping is to find a multi-level circuit equivalent to the given Boolean network such that it is comprised of gates in the library and has minimum cost, which can be area, delay, testability or power consumption.

Graph covering algorithm A systematic approach to apply technology mapping is through graph covering algorithm. With this mechanism, the technology mapping problem can be viewed as optimization problem of finding the minimum cost covering of the graph by choosing from the collection of pattern graphs for all gates in the library. A cover is a collection of pattern graphs such that each node in the graph is contributing in one or more patterns in the graph. Furthermore, one restriction in this theory should be taken into consideration is that the inputs of one pattern must be the outputs of some other patterns in the covering. Otherwise, the inputs of one pattern could be the outputs of the internal nodes of some other pattern.

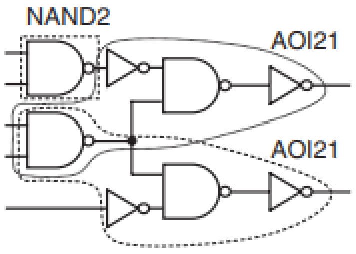


Figure A.16: Graph covering

As illustrated in Figure **??**, the subject graph splits into three pattern graphs. Each of them has outputs, which is directly connected to the inputs of other pattern graph as discussed in the graph covering theory.

One-hit magical logic optimization As known, most of the EDA synthesis tools uses command line arguments to perform any required function or somewhat called one-hit task accomplishment. After constraining the design and setting the environmental attributes, we then compiling the design. So, what is the mean of design compilation?

To compile something in software is to debug it versus errors and then to execute what is written in command line argument. There are a variety of famous synthesis compilers such as Synopsys design compiler and Cadence RTL compiler. Furthermore, any compiler can perform a single task is to simply perform all the steps of the synthesis process in only one single step. These steps involve technology-independent optimization, technology mapping, technology-dependent optimization and timing analysis, in addition to DFT scan-chain insertion. The cost of this process is time. The compilation process is taking much time to perform and terminate. Therefore, the compilation command in any synthesis tool deserves the title of “one-hit magical command”.

A.1.3 Flat Design flow

A quick definition

What is the meaning of word "Flat"? Flat object is an object with no terrains in its surface. In addition to that it is considered as a one solid piece with no partitioning, structuring, or hierarchical levels in its nature. For example, a football pitch can be considered as a flat ground where there are no terrains or multi-leveling in the ground. It just a coherent piece of earth.

Concepts and methodologies

Concepts of flat design in digital VLSI systems

Flat vs hierarchical RTL coding

We can apply the concept of the flat and hierarchical objects in our VLSI system designs so that we can determine the difference between each of them and reason behind preferring one over the other.

Writing a high-quality RTL code in digital designs necessarily required writing each functional block in a single HDL, Verilog or VHDL, file so as to be able to easily verify each sub-module for language syntax errors and test it in functionality manner to make sure that it performs its function properly. Therefore, writing the RTL code in hierarchical manner is a must. But, how do we can write the RTL code in the hierarchical form? It's a common method among all digital designers to write the code of each sub-block, and then test each of them against functionality and syntax errors.

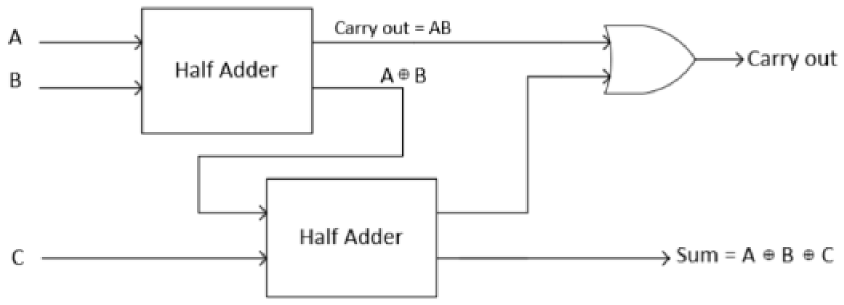


Figure A.17: Hierarchical Design of full adder

In the Figure A.17, we can see hierarchical design of the full adder which consists of two half adder sub-blocks and an OR gate somehow interconnected with each other. After designing the half adder, we know its functionality, therefore, we can test it and ensure that it works properly. After that, we can instantiate this sub-block modules in the top-level HDL file and then somehow interconnect them, and at last we can test the whole system.

On the contrast, the flat RTL coding style is considered to be only used for writing HDL codes for small-scale VLSI designs where the design is simple and easy to code. Flat-based design in RTL coding has an alternative name which is dataflow design style where the data can flow among the logic gates without encountering obstacles or blockages.

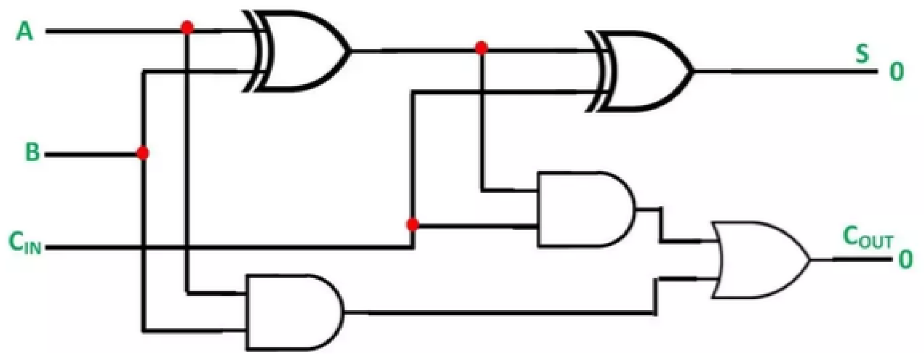


Figure A.18: Flat-based Design of full adder

As illustrated in Figure A.18, the figure shows the flat-based, or dataflow design style, of the full adder. The RTL code of such designs is written by using the Boolean expressions that describe the behavior of the logic circuit. The problem arises when the design is growing in scale. As a result, these Boolean expressions is also growing in complexity, and become more difficult to implement. Another result for writing in flat-based design for large scale designs is that the probability of making syntax and logical errors will increase in exponential-wise manner. In addition to difficulty of debugging and fixing these errors. Furthermore, we can find that the testing and verification of such design is also difficult and time-consuming.

Concept of flat-based design in synthesis process

As we discussed above, the RTL code takes the advantage in writing of the HDL code. Therefore, can expect that the synthesis tool will have an input of RTL in hierarchical form, thus we need to know the idea behind using flat-based methodology in synthesis process.

Flattening process of the design is useful for unstructured designs such as random logic or control logic, since it removes intermediate variables and uses Boolean distributive laws to remove all parenthesis. It's not suited for designs consisting of structured logic such as a multiplier or a look-ahead-adder.

Flattening the design results in a two-level, sum of products from, and resulting in vertical logic (i.e. few logic levels between inputs and outputs). That does not imply removing the levels of hierarchy by the way. Removing hierarchy is used to enhance the performance of logic optimization as it will be discussed later. This design style may have a significant enhancement of logic timing, area and power.

Methodologies of flat-based design flow

Removing the whole design hierarchy

By default, the synthesis tool maintains the design hierarchy during all the synthesis phases. This hierarchy has a great impact on logic optimization process, since the synthesis tool considers each functional sub-block in the hierarchy as a stand-alone design and performs the logic optimization techniques on it without crossing the logical boundaries between all sub-blocks. This has a large effect on combinational logic optimization process, because there may be a combination of logic in some module needs a certain logic element in another some sub-block to make a great minimization in the logic cloud. But due to the logical boundaries, the logic optimization process is confined inside each sub-block.

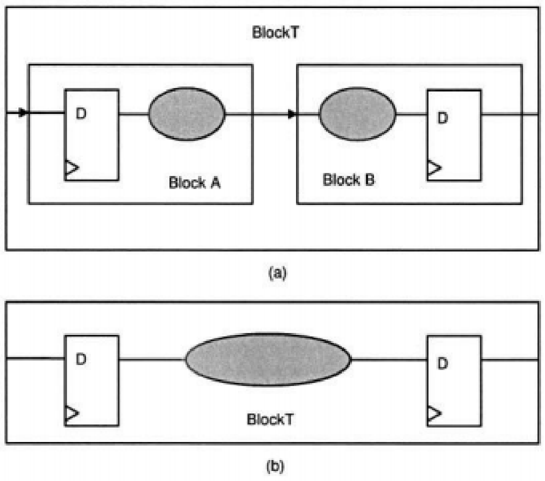


Figure A.19: Removing hierarchy of the design

As illustrated in Figure A.19, the top-level "BlockT" has two sub-blocks 'A' and 'B'. They have a peer-to-peer combinational logic as shown in Figure A.19(a), and it's desirable to remove the logical boundaries between them to enhance the optimization between them. By removing the hierarchy in Figure A.19(b), the synthesis tool became able to optimize the logic without any blockages or boundaries.

Impacts of removing the whole design hierarchy As we explained before, removing the design hierarchy will facilitate the work of the synthesis tool on logic handling and optimization, but the problem arises when the design is growing in scale. Due to memory needs of the synthesis tool, each removed hierarchy level or any ungrouping operation on the design has to be saved in tool memory. Therefore, removing a larger hierarchy level that contains large sub-blocks implies increasing in the memory sized that needed for saving the new changes in these new sub-blocks. As a result, the synthesis tool will see a larger design that needs to be handled in timing, area and power aspects, and the processing time of the design will be larger, hence the optimization will need much time to be realized. On the other hand, the small designs do not have a large-scale area, thus they may be strongly benefited from the advantages of the flat-based design based on removing hierarchy technique.

To overcome the above problem, we can use a mixture of flat-based and hierarchical-based design. We have two methods to solve this issue which will be exposed in this discussion.

Minimizing the levels of design hierarchy

The design hierarchy is considered as a multi-level tree with branching for every level. Each branch has smaller branches, and each sub-branch has also smaller sub-branches.

As we discussed before, the main problem of removing the entire hierarchy located in the tool is the memory needs. Therefore, we can remove only some levels of this hierarchy. Beginning from the leaf blocks hierarchy level, the removal of this level may have a small impact on the memory, thus improving optimization, and also no much time needed. The second step is to step up to the next higher level and remove it, and then performing a compilation check for verifying on the compilation time impact of the removed level, and so on. We can continue to iterate this cycle till the compilation time suffers from a large step in time. We have to use this method after performing the compilation of the standard hierarchical method and almost the timing requirements are about to be realized as we do not want to increase the timing violations due to using this method.

Structuring and flattening

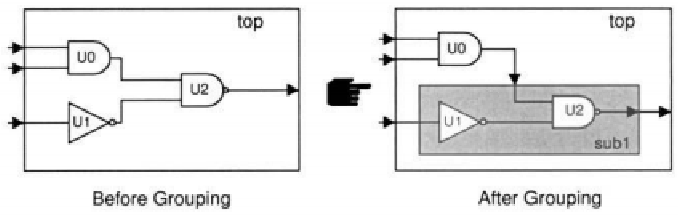


Figure A.20: Grouping the design sub-blocks

Structuring the design implies grouping some of the sub-blocks existed in a certain level of hierarchy under the condition that these sub-blocks is adjacent to each other. As illustrated in Figure A.20, the sub-blocks ”U1” and “U2” in the top level, which are here considered as only logic gates due to smallness of the design scale, are grouped together in a one new sub-block. The sub-blocks, inside the new sub-block, can then be flattened and optimized for timing and area with small effect on the memory needs. After successfully optimizing the sub-blocks, the grouping mask that was inserted, can be removed at last as illustrated in Figure A.21.

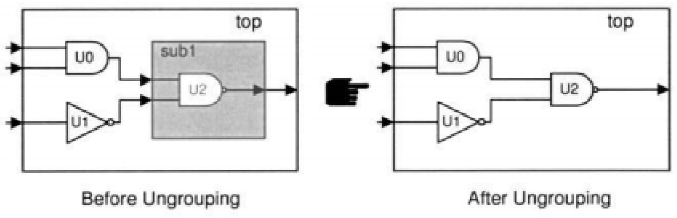


Figure A.21: Unrouping the design sub-blocks

Overall system performance based on flat design flow Due to using the flat design flow process, the process of removing of any level of hierarchy improves the overall system performance where is no constraints on logic optimization process. Therefore, a successful flat-based design process should strongly have advantages over a standard hierarchical-based design process in timing, area and power. The flat-based design process, as we discussed above, helps the tools to make further logic optimizations which can not be done in the standard mode. As a result, the delay of the timing paths will extremely decrease, therefore, the design can operate on a faster clock speed. Another advantage of using this method is that the area of the design is also can be reduced due to that further optimization that is minimizing the logic, hence reducing its corresponding area. The final and the most important factor is power. The power is directly proportional to the design area. Therefore, the power consumption will also be lower than before.

But one of disadvantages of this method is typically time-consumption. To flatten to design and to achieve higher performance of the system, it logically needs much compilation time to realize. In addition to time-consumption, the main and limiting factor is the memory needs of the synthesis tool. This factor makes us resort to intermediate solutions to maintain almost the same performance that we desire to have.

A.1.4 Design Compiler Topographical flow

In 2005 Synopsys incorporated topographical synthesis technology, Topographical technology enables you to accurately predict post-layout timing, area, and power during RTL synthesis without the need for wireload model-based timing approximations. It uses Synopsys' placement and optimization technologies to drive accurate timing prediction within synthesis, ensuring better correlation to the final physical design. This new technology is built in as part of the DC Ultra feature set and is available only by using the compile_ultra command in topographical mode.

How Topographical Technology work:-

In ultra deep submicron designs, interconnect parasitics have a major effect on path delays; accurate estimates of resistance and capacitance are necessary to calculate path delays. In topographical mode, Design Compiler leverages the Synopsys physical implementation solution to derive the "virtual layout" of the design so that the tool can accurately predict and use real net capacitances instead of wireload model-based statistical net approximations. If wireload models are present, they are ignored. In addition, the tool updates capacitances as synthesis progresses. That is, it considers the variation of net capacitances in the design by adjusting placement-derived net delays based on an updated "virtual layout" at multiple points during synthesis. This approach eliminates the need for overconstraining the design or using optimistic wireload models in synthesis. The accurate prediction of net capacitances drives Design Compiler to generate a netlist that is optimized for all design goals including area, timing, test, and power. It also results in a better starting point for physical implementation.

steps running Topographical mode:-

1. Set up the libraries, Design Compiler requires both logic libraries and physical libraries. Design Compiler topographical mode uses the same logic libraries as the Design Compiler wire load mode; it uses the Milkyway format for physical libraries

2. Read in the design (Verilog, VHDL, netlist, or .ddc).

3. Specify timing, area, power, and test constraints, set up the design environment by specifying the external operating conditions (manufacturing process, temperature, and voltage), loads, drives, fanouts.

4. Provide floorplan information, The principal reason for using floorplan constraints in topographical mode is to accurately represent the placement area and to improve timing correlation with the post-place-and-route design. You can provide high-level physical constraints that determine core area and shape, port location, macro location and orientation, voltage areas, placement blockages, and placement bounds.

5. Visually verify the floorplan, Use the Design Vision layout window to visually verify that your pre-synthesis floorplan is laid out according to your expectations. The layout view automatically displays floorplan constraints read in with the extract_physica_constraints command.

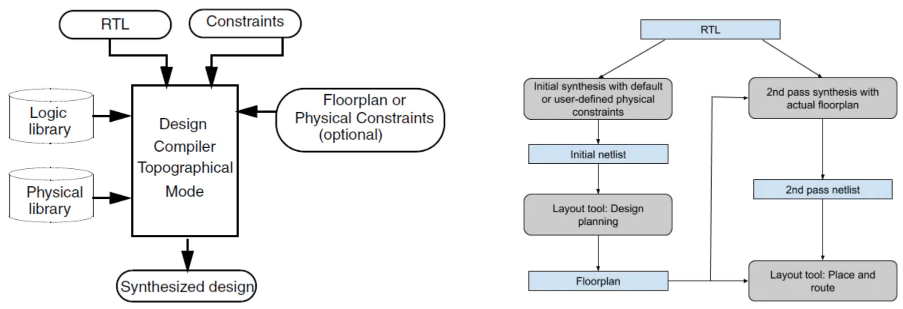

RTL
Constraints
Logic library
Physical library
Design Compiler Topographical Mode
Floorplan or Physical Constraints (optional)
Synthesized design
RTL
Initial synthesis with default or user-defined physical constraints
Initial netlist
Layout tool: Design planning
Floorplan
2nd pass synthesis with actual floorplan
2nd pass netlist
Layout tool: Place and route

Figure A.22: topographical flow

Topographical mode target:-

1. Accurate predicts, timing, area and power.

2. Ensures synthesis output correlate to actual layout.

3. Reduce the number of iterations required to close design goals eliminating the need of load wire models.

4. Early prediction of routing congestion and visualization of congestion hot spots and timing issues.

5. Allows RTL designers to fix design issues early, cutting time and improving scaling predictability.
6. Provide visualization and analysis of congested circuit regions.
7. Perform synthesis optimization to minimize congestion in these areas.
8. Provides significant improvement in design time.
9. Delivers best Quality of Results (QOR) in terms of area, timing, power and test correlated to physical implementation.
10. Designers fix real design issues while still in synthesis and generate a better start point for physical design, eliminating costly iterations.
11. Remove timing bottlenecks by creating fast critical paths.
12. Offers more flexibility for users to control optimization on specific areas of designs.
13. Distributed synthesis with automated chip synthesis.
14. Enables higher efficiency with integrated static timing analysis, test synthesis and power synthesis.
15. Designed for RTL designers and requires no physical design expertise or change to the synthesis use model.
16. Delivers accurate correlation to post-layout timing, area and power without the need of WLM.

A.1.5 Hierarchical Design flow

Whenever we start any big work, we try to break down that work into the small-small work. In software terminology – A big program is divided into the sub-program and sub-programs into the modules (example - OOPs is based on similar approach -People those aren't aware about this approach- skip this line). Now, similar type of approach, when you are going to implement in the VLSI design – That Design is known as Hierarchical Design. As the size increases, complexity increases and then it become very difficult for a single person to do the routing/timing closure/ optimization etc of such a big-complex design. Apart of this there are limitation with respect to the Memory of the computer device and runtime of the EDA tools. So it is good to break/divide such a complex task into small-small task. For example, Let us suppose, you have to replace 10 4-input-AND Gates with 20 2- input-AND gates, so for that you have to load the whole design (which has almost millions of gate). So for such a small task, you have to waste a big chunk of memory. To resolve these issues, hierarchical design methods come into picture.

The basic flow of hierarchical design is simple...

- Dividing a design into multiple blocks (sometimes referred to as sub-chips, sub-blocks, modules, hierarchical blocks, etc.).
- Designers can work on the blocks separately and in parallel from RTL through physical implementation.
- Working with smaller blocks keeps tool run-time short.
- Block-level timing closure should be relatively easy to achieve compared to the timing closure for the entire chip.
- Once all blocks are finished, they are integrated to create the final chip. Here these blocks are treated as Black-box (only few specific information available at the top level).
- Close the timing of the final chip or you can say that close the timing between the blocks. If proper work has done in the starting, it should

be close in first iteration (Ideally). Pictorial view of the Hierarchy based Design Flow is shown in figure A.23 .

In figure A.23 we have mentioned 2 types of hierarchy.

- Physical Hierarchy: Physical hierarchy is based on back-end considerations such as cell placement, I/O placement,macro placement, interconnect routing and associated timing issues.
- Logical Hierarchy: Based on the function of the design modules/blocks, which is usually determined by the designers and their HDL coding methods.

These 2 are different but still there should be a correlation between these 2 so that we can reduce the time needed to achieve the timing of the chip. Hierarchical design Flow benefits:

- Improved Productivity when designing complex chips.
- Run time is fast because you can work over individual block and those will be small in size in comparison to the full design.
- In case of any timing issue, you can fix individual block.
- Incremental functional and timing fixes is possible after timing closure.

In the traditional flat ASIC flow :

- If there is any problem in the timing after routing, then there are equal chances that you have to go back to the architecture level design for correcting that.
- Memory limitation can also create problem
- Run time in the case of multi-million gate design is huge.

One thing keep in mind, I am not saying that Full-flat design is useless or Hierarchical design has replaced that approach completely. But Hierarchical design offloads the burden of Full-flat flow (traditional flow) during the implementation phase. Even now, at the signoff stage, most of the companies (even I can say

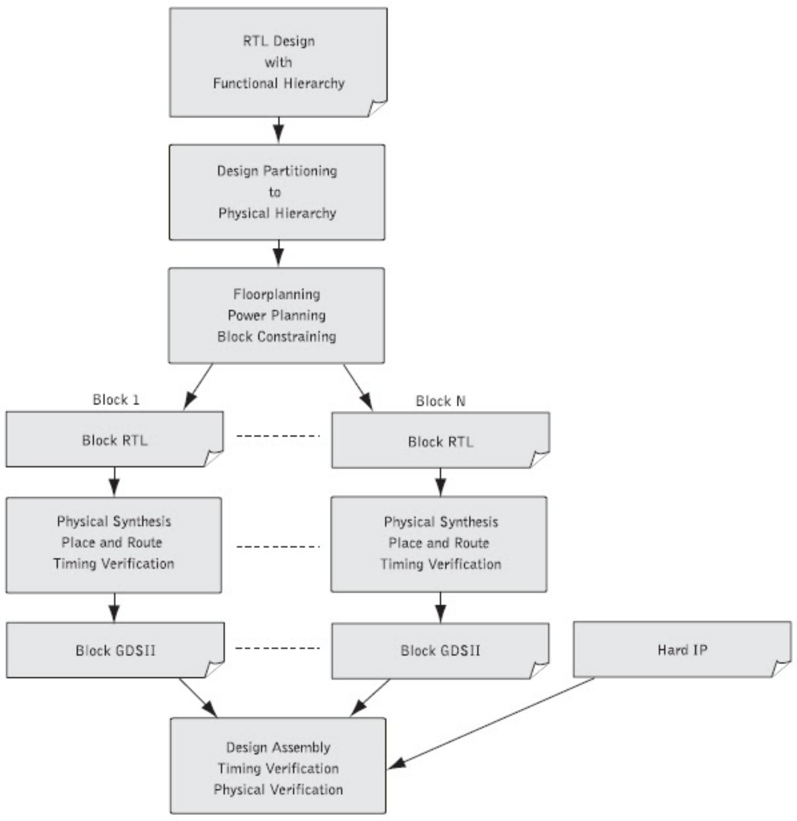

RTL Design with Functional Hierarchy
Design Partitioning to Physical Hierarchy
Floorplanning Power Planning Block Constraining
Block 1
Block N
Block RTL
Block RTL
Physical Synthesis Place and Route Timing Verification
Physical Synthesis Place and Route Timing Verification
Block GDSII
Block GDSII
Hard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

Figure A.23: Hierarchical Flow

99% companies) are using Full-flat-flow for rechecking everything and to make sure nothing is messed up in between. But after using hierarchical approach, even in implementation phase, designers have saved a significant time.
Now if you pay attention, you come to know that as per the pictorial diagram, there is only 1 important step in the Hierarchical flow and that is the "Setting block level constraints".
Block Level Constraints are of 2 types:

- Physical Constraint: These constraints depend on the floor-plan of the top level. Means where exactly this block will be placed on the top level.
  - Size and shape of the block
  - Pin placement with in the block
- Timing Constraint: Block timing Budget allocation or say timing budget-

ing. Like the delay at the input port / output port and all.

Always remember that these constraints can't be decided in a single iteration. For setting these constraints we have to use both the top-down and bottom-up approaches. Like position of the pin in the block depends on the position of the pins required in the final chip. In the similar way, if there is any hard macro (you can't change the position of the pin in that), so you have to place that block in such a way that it should be closer to the pin position of the final chip. So in the complex design with large number of blocks, you have to do few iteration and you have to use both the approaches in parallel. During deciding these constraints, in most of the cases we add enough margins so that we can cover any inaccuracy in estimating (which we have done in early phase of the design) at the end of design cycle. A uniquely identifiable element.
Physical Constraint List of physical constraints are: (it contain all the top level and block level constraints)

- Die area
- Core Placement area
- Utilization
- Cell location
- Pin Location
- Placement Blockage
- Wiring keep-out
- Voltage Area
- Site Row

If you have notice, all physical constraints are related to the location. It's like when you are designing a layout of your house, then you are applying a lot of constraints like window should be in left, door should be in North, some corner is fixed for Kitchen. So similarly in chip designing, you have to place a lot of constraints as per the requirement/specification of the chip or sometime as per the specification of the IP blocks.

Timing Constraint for Block is equivalent to the “Timing Budgeting” or say “Block timing Budget allocation” for example the delay at the input port / output port and all.

Timing Budgeting: Timing budgeting is an important step in achieving timing closure in a physically hierarchical design. The timing budgeting determines the corresponding timing boundary constraints for each block in a design. If the timing boundary constraints for each block are met when they are implemented, the top-level timing constraints are satisfied. To understand it clearly again consider the scenario of your house.

- You want to place your Drawing room close to main entrance (reason being you don't want that any new person travel all through the bedroom-kitchen and then finally sit in drawing room),
- Travel time between Kitchen and Dining room should be as less as possible.
- Bathroom should be as close to your bed room (so that you should not spend much time before going to office just in to and fro between bedroom and bathroom.

So, all these are timing constraints during the finalizing of layout of your house. Think what will happen if Kitchen is in 3 rd floor and Dining room is in ground floor.

Similarly, in the chip designing, while you divide the design into small blocks, you have to take care about timing between block's I/O to other block's I/O, block's I/O to chip I/O. If a data is required by a block A for doing some processing and this data is generated by block B, so Block A should know when it will receive the data from the Block B. Since at the top level these blocks are Black Box, so during timing budgeting we have to define the constraint at input of Block A that it will receive the data after X time (this X we have to estimate correctly on the basic of experience and knowledge of the block, usually we constraint with X+x amount where x is the margin we are keeping in case of wrong estimation).

The block-level timing constraints are in the form of one or more “logical timing constraint points" at the input and output ports of block-level circuits. Each logical timing constraint points Using the logical timing constraint point, the circuit design system performs independent timing analysis and optimization of each block-level circuit.

Let me explain this timing budgeting concept in other way also. Let us suppose that you have a design and as per the specification, date should reach from port x1 to x2 with in 3sec. Now in the flat design you can meet this timing very easily because you are aware about the no of cells, types of cells and wire length between the x1 and x2 also. So it's very easy to estimate the delay between x1 and x2.

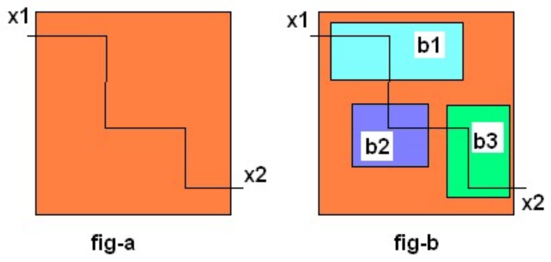


Figure A.24: illustration

Now compare it with the hierarchical design. You have figured out that there are 3 blocks in between x1 and x2. Now the block owners are different, so you want to make sure that everyone gets proper information before designing their own block. Like how much maximum delay should be there for b1/b2/b3 (like we get the specification from x1 to x2 at the top level). So it's our job to provide the proper specification to the block owner. If I will miss this, it will be a near-to impossible task to integrate it at the top level without any iteration. So, dividing this available 3 sec time into these 3 blocks is known as Time Budgeting in this example (On the big design also the definition is similar – Distribute the top level timing constraint effectively to the block level is known as Time Budgeting.)

Now, randomly you have assigned the 1sec delay to each block. But the cell delay (cells present inside the blockb1) of block b1 itself is exceeding 1sec (if you will add the net delay after layout – it will be much more than 1sec), so such type of timing budgeting is known as Under-Budgeted Timing. Similarly, if the cell delay of block b2 is very less (let's assume 0.2sec), then estimating the 0.8sec for the net delay may be too much. And in such cases this type of timing budgeting is known as Over-budgeted Timing. So it's very important to estimate the timing very accurately.

Now let's assume that you have estimated the delay (timing budgeting) correctly for each block (e.g 1.5sec, 0.5 sec, 1 sec). There is a processing in block b2 on the data which is coming from the block b1, so for that, block b2 should know that when it is going to receive the data (after how much time with reference to system clock it will receive the data). Since both the blocks are handling by different person and they are closing their blocks independently, so we have to define few constraints at the input and output ports of each block in terms of timing and all. These constraints can be with respect to clock or data or both.

For example, we can define one constraint at the input of block b2 that for all calculation within the block, it can assume that data has a delay of 1.5sec (in PT it will be like – set_input_delay). So this will be the constraint for this block with respect to this particular data and at this particular port.

Similar logic is applied to all the hierarchical design. And similar type of timing constraint you have to define for all the sub-blocks. Mostly these constraints are in the form of SDC (Synopsys delay Constraint). At a bare minimum, a design will have clock constraints, and input and output delay constraints. As a design gets more complicated, you may tend to add exception constraints. However, the fewer the exceptions the better in terms of tool run time. Anyways the details about the timing constraints are in the different post.

What should be required to do an efficient timing Budgeting:

- All Cells and blocks should be floor-planned. So that at least you can figure out the location of the I/O pins. If you can get a globally routed design, then it is wonderful.
- Timing Constraint for top level should be available.
- If you are using any hard block, the timing constraints/specification should be properly known.

Challenges in the timing budgeting:

- Chip level constraints must be mapped correctly to block level constraints.
- For allocating the timing budgeting to all the sub-blocks, we have to predict reasonable delay for global interconnects. Which is difficult before the design is physically constructed.
- Because of not able to predict accurate delay of global nest, lots of iterations are required to close the timing in the design.
- If you over budgeted any block, you are wasting the timing slack.
- If you under budgeted, you will get negative slack means timing violation. It may be possible that under budget of one block is because of over-budgeting the other block. Because at the end timing should be as per specification of chip.
- Unfortunately, the delay budgeting problem will only become more difficult as more wires become global wires whose pin-to-pin delays are strongly dependent on their actual implementation by detailed routing tools.

Once "timing budgeting" is done, each and every block act like a small chip/design. Once every block is done, these are integrated at the top level and then the timing of top level is verified. For timing verification at the top level we only need the timing information corresponding to each block, so we can ignore other details (which can create a problem in terms of huge memory and runtime). For this purpose, we use the timing models corresponding to each sub-blocks. A timing model contains information about the timing characteristics, but not the logical functionality, of sub-module/blocks. After generating a timing model of a block, we use that model in place of the original netlist for timing analysis at higher levels of hierarchy. This technique makes whole-chip analysis run much faster.

Now in the last, as you know that there is nothing 100% perfect, Hierarchical Flow also have some limitations.

Limitation in Hierarchical Design:

- When you place the block in top level, they can act as routing obstruction and due to which it may be that you have to route the wire more than your estimation and it will create timing violation.
- Nets that interconnect the blocks have to be routed through channels between blocks. These routes tend to be long and can cause timing and signal integrity problems.
- Routing Congestion in some place and some place is not utilized properly. So it may happen that hierarchical design requires more chip area that flat design.
- Timing of block I/O can't be changed during top-level optimization. so blocks must be optimized with a good I/O budget.
- Prediction of I/O budget should be very accurate, else there may be a lot of iterations between I/O budgeting and closing the timing of block.

A.2 Formal Equivalence Check

A.2.1 RTL vs Gate-level netlist

As discussed before, the RTL a high-level description of hardware in the level of data and control signals transfers among registers and functional blocks. The RTL code is basically written in common HDL languages such as Verilog HDL and VHDL. The RTL code is a raw and constraints-free hardware description where no clock period, timing, logical or physical constraints are defined. On the contrast, the gate-level netlist basically comprises of logic gates that connected by wires called nets as illustrated in Figure A.25. The net is just a wire that has a name so that the synthesis tool can distinguish between the different wires in the logic circuit. The gate-level netlist is also written by the hardware description languages in the form of basic logic gates (i.e. AND, OR, XOR, . . . , etc.). After constraining the design with timing and logical constraints, the tool performs logic optimizations on the design in timing, area and power aspects. After that, the result is our gate-level netlist. After synthesis tool finishes its work, it writes the design in HDL language, Verilog or VHDL, based on user's desire. The final synthesized gate-level netlist is optimized in timing, area and power and satisfies the required technical specifications.

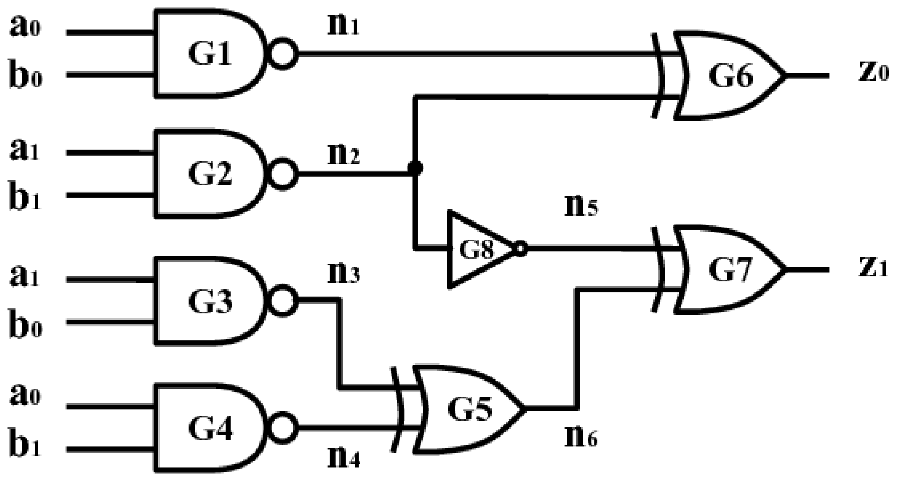


Figure A.25: Gate-level netlist

A.2.2 RTL-to-netlist logic equivalence checking

Definition

Logic equivalence checking, or LEC, as stands from its name, is the process of checking and verifying the logical functionality of the entire design. It's a piece-by-piece checking which means that this checking involves all test cases presented in the design.

Why do we use LEC?

Before synthesis process, the RTL code must be completely tested and verified against syntax errors and logical functionalities. So, why do we make another step of checking? Is this step redundant? The absolute answer is no. One of the explicit reasons to perform this operation is the synthesis tool itself. Due to growing complexity of the designs, the synthesis is, at the end, a software-developed program which may be make some logical mistakes during logic optimization process. In addition to that reason, sometimes manual editing of the gate-level netlist may also generate unexpected errors, and also the insertion of DFT scan chain can make logical errors in the functionality of the circuit.

Working mechanism

The major tools which do formal checking are Synopsys Formality and Cadence LEC. These tools divide the design into compare points. That tool traces the circuit back for each of the compare points in both reference and implementation designs, till it reaches primary inputs or flip-flop outputs. This is called a logic cone of this specific compare point as illustrated in Figure A.26. The tool then generates a stimulus values or vectors for those primary inputs or flipflop outputs, compute the values achieved at that compare point and then checks whether they are the same or not in both designs. Compare points can be primary outputs or flipflops/Latches or inputs of black boxes.

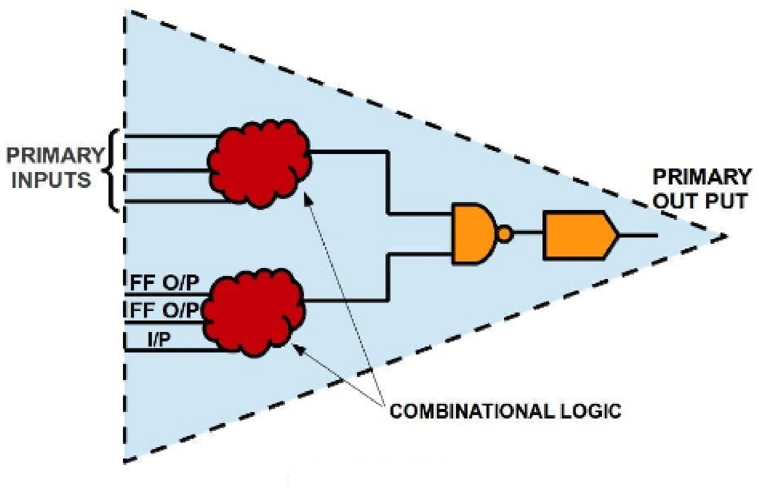


Figure A.26: Logic cone

A.2.3 LEC flow

The synthesized gate-level netlist in most cases is written by synthesis tool in a random form and in a single. Therefore, we can difficultly perform dynamic simulations to check the functionality of entire design as a single block testing, thus most of famous silicon foundries, such as Synopsys and Cadence, developed their own formal logic equivalence checking tool. We are going to expose the main steps that is performed in this process.

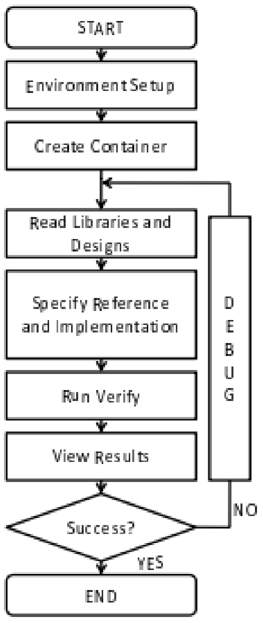


Figure A.27: Logic Equivalence checking process flow

The Figure A.27 illustrates the basic steps of the logic equivalence checking process flow. We are going to discuss each of them as concisely as possible.

Setup

The first step is to simply reading the RTL design files and its corresponding standard cell libraries that needed to link the design. After that, we can read the implementation design which is the synthesized gate-level netlist in our case and also its corresponding libraries. We can add a guidance file generated from synthesis tool to ease the checking operation. This guidance file contains details about the optimized registers, and registers which got merged during synthesis process. If we inserted clock gating structure in the netlist, therefore, we have to identify it to the tool to know that it is an excess structure which is needless to be compared.

Once the setup stage is complete, we just have to verify that the list of black boxes in reference and implementation design is the same in both designs.

Matching

In matching stage, the LEC tool checks the one-to-one correspondence of the compare points in both RTL and synthesized netlist designs. Ideally the number of points should be equal in both designs, but in some cases, the number of compare points differ. For example, the netlist has extra output ports or either one of the designs has extra registers. These unmatched points can be ignored if the next step is completed successfully.

Verification

The final step in LEC process is verification. In this station, the tool checks for logical equivalence of the design by comparing the results of all specified compare points. For successful verification, all compare points must be equivalent.

If the verification is failing, we have to check to matching results whether some expected unmatched points is there or not. If the matching results are clean, we can manually debug the verification failing points using applied patterns. Verification can fail due to some missing constant declaration. For example, if the formal verification is done between non-scanned netlist and scanned netlist, we have to set the test enable constant as zero. Otherwise, the verification will fail with a huge number of failing points.

A.3 Place and Route

A.3.1 Floorplanning

Floorplanning is the art of any physical design. A well-thought-out floorplan leads to an ASIC design with higher performance and optimum area. the input to the floorplanning step is the output of system partitioning and design entry netlist. At the start of floorplanning we have a netlist describing circuit blocks, the logic cells within the blocks, and their connections. We can think of the standard cells as a hod of bricks to be made into a wall. What we have to do now is to set aside spaces (we call these spaces the channels) for interconnect, the mortar, and arrange the cells. We still have not completed any routing at this point that comes later all we have done is placed the logic cells in a fashion that we hope will minimize the total interconnect length.

The floorplanning steps are: Design planning, pad placement, Power planning, and Macro placement.

Design planning

Efficient design implementation of any ASIC requires an appropriate style or planning approach that enhances the implementation cycle time and allows the design goals such as area and performance to be met.

There are two style small to medium ASIC's, flattening the design is most suited; for very large alternatives for design implementation of an ASIC–flat or hierarchical. For and/or concurrent ASIC designs, partitioning the design into sub designs, or Hierarchical Flow, is preferred. The flat implementation style provides better area usage and requires effort during physical design and timing closure compared to the Hierarchical Flow. The area advantage is mainly due to there being no need to reserve extra space around each sub design partition for power, ground, and resources for the routing. Timing analysis efficiencies arise from the fact that the entire design can be analyzed at once rather than analyzing each sub circuit separately and then analyzing the assembled design later. The disadvantage of this method is that it requires a large memory space for data and run time increases rapidly with design size. The hierarchical implementation style is mostly used for very large and/or concurrent ASIC designs where there is a need for a substantial amount of performance degradation is mainly because the components forming the critical path may reside in different partitions within the design thereby partitioned logically or physically. Logical partitioning takes place in the early stages of ASIC design (i.e. RTL coding). The design is partitioned according to its logical functions, as well as physical constraints, such as interconnectivity to other partitions or sub circuits within the design. In logical partitioning, each partition is place-and-routed separately and is placed as a macro, or block, at the ASIC top level. Physical partitioning is performed during the physical design activity. Once the entire ASIC design is imported into physical design tools, partitions can be created which combine several sub circuits, or a large circuit can be partitioned into several sub circuits. Most often, these partitions are formed sub circuits are designed individually. However, hierarchical design implementation may degrade the performance of the final ASIC. This extending the length of the critical path. Therefore, when using a hierarchical In the hierarchical design implementation style, an ASIC design can be design implementation style one needs to assign the critical components to the same partition or generate proper timing constraints in order to keep the critical timing components close to each other and thus minimize the length of the critical path within the ASIC. computing capability for data processing. In addition, it is used when by recursively partitioning a rectangular area containing the design using vertical or horizontal cut lines. Physical partitioning is used for minimizing delay (subject to the constraints applied to the cluster or managing circuit complexity) and satisfying timing and other design requirements in a small number of sub circuits. Initially, these partitions have undefined dimensions and fixed area (i.e. the total area of cells or instance added to the partition) with their associated ports, or terminals, assigned to their boundaries such that the connectivity among them is minimized. In order to place these partitions, or blocks, at the chip level, their dimensions as well as their port placement must be defined.

In this phase we define the core height, core width, core utilization ratio, core aspect ratio, and space between I/O pads and core cells.
core utilization is the ratio between cell area divided by interconnect area for example at core utilization equal to 0.8 that is mean that cell area is 80% and interconnect area is 20%.
core aspect ratio is the ratio between height of the core to its width for example at core aspect ratio equal to 2 that is mean that the length of core height is double of core width.

the first step is to determine ASIC device and core width and height. In addition, standard cell rows and I/O pad sites are created. The rows and I/O pad sites are for standard cell and I/O pad placement. Figure 2-3 shows an initial ASIC design floorplan. The height of a row is equal to the height of the standard cells in the library. If there are any multiple-height standard cells in the library, they will occupy multiple rows. Most of the time, standard rows are created by abutment. The standard rows are oriented in alternating 180-degree rotation or are flipped along the X-axis so that the standard cells can share power and ground busses. If the ASIC core has routing congestion owing to the limited number of routing layers, one solution is to create routing channels between rows. These all can be separated individually or as pairs.

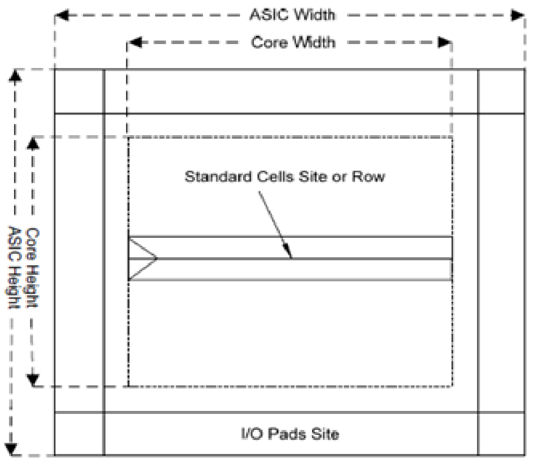


Figure 2-3 ASIC Design Initial Floorplan

pad placement

Correct I/O pad placement and selection is important for the correct function of any ASIC design. for a given ASIC design there are three types of I/O pads. These pads are power, ground, and signal. It is critical to functional operation of an ASIC design to ensure that the pads have adequate power and ground connections and are placed properly in order to eliminate electromigration and current-switching noise related problems. Electromigration (EM) is the movement or molecular transfer of metal from one area to another area that is caused by an excessive electrical current in the direction of electron flow (electron "wind"). Electromigration currents exceeding recommended guidelines can result in premature ASIC device failure. Exceeding electromigration current density limits can create voids or hillocks, resulting in increased metal resistance, or shorts between wires, and can impair ASIC performance.

Switching noise is generated when ASIC outputs make transitions between states. An inadequate number of power and ground pads will lead to system data errors due to these switching noise transients. There are two types of mechanisms that can cause noise:

- dv / dt caused by a capacitive coupling effect
- di / dt caused by an inductive switching effect

The capacitive coupling effect is the disturbance on the adjacent package pin caused when switching transients inject pulses via parasitic coupling capacitance.
The maximum C (dv / dt) noise occurs when the ASIC output current nears the maximum current for a given capacitive output load of C. This noise problem can be resolved by proper pad placement, package pin selection, ASIC output pad type and drive current, and input pad type.

To reduce or eliminate capacitive coupling effects during I/O pad placement and selection, one may consider the following guidelines:

- Isolate sensitive asynchronous inputs such as clock or bidirectional pins from other switching signal pads with power or ground pads.
- Group bidirectional pads together such that all are in the input or output mode.
- Group slow input pads together positioning them on higher inductance package pins.
- Use input pads with hysteresis as much as possible.

The inductive switching effect is related to simultaneous switching ASIC outputs that induce rapid current changes in the power and ground busses. The inductance in the power and ground pins cause voltage fluctuations in the ASIC internal power and ground level relative to the external system. These rapid changes in current can change the ASIC input pads' threshold and induce logic errors or may cause noise spikes on non-switching output pads that affect signals connected to other systems. The maximum L (di / dt) occurs when ASIC output starts to make the transition to another voltage level and its absolute current increases from zero through a wire with inductance of L. Factors such as process, ambient temperature, voltage, location of output pads, and number of simultaneous switching output pads determine the magnitude of inductive switching noise. To control inductive switching noise, enough power and ground pads must be assigned and placed correctly. This way the noise magnitude will be limited. This noise reduction will prevent inputs of ASIC design from interpreting the noise as valid logic level.

Successful reduction of inductive switching noise can be accomplished by the following:

- Reduce the number of outputs that switch simultaneously by dividing them into groups with each group having a number of delay buffers inserted into their data paths.
- Use the lowest rated sink current or low-noise output pads as long as speed is not an issue.
- Place the simultaneously switching output or bidirectional pads together and distribute power and ground pads among them according to their relative noise rating.
- Assign static and low frequency input pads to higher inductance package pins.
- Reduce the effective power and ground pin inductance by assigning as many power and ground pads as possible.

power planning

The next step is to plan and create power and ground structures for both I/O pads and core logic. The I/O pads' power and ground busses are built into the pad itself and will be connected by abutment. For core logic, there is a core ring enclosing the core with one or more sets of power and ground rings. A horizontal metal layer is used to define the top and bottom sides, or any other horizontal segment, while the vertical metal layer is utilized for left, right, and any other vertical segment. These vertical and horizontal segments are connected through an appropriate via cut. The next consideration is to construct the standard cell power and ground that is internal to the core logic. These internal core power and ground busses consist of one or two sets of wires or strips that repeat at regular intervals across the core logic, or specified region, within the design. Each of these power and ground strips run vertically, horizontally, or in both directions. Figure 2-5 illustrates these types of power and ground connections.

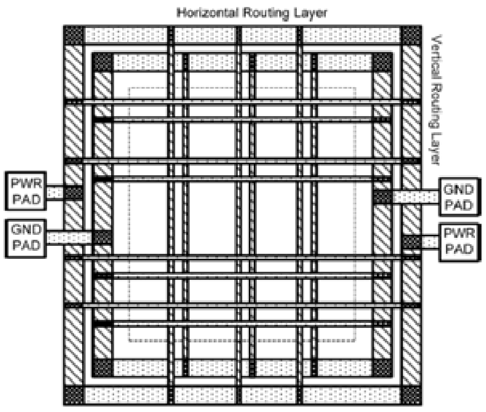


Figure 2-5 Ring and Core Power and Ground

If these strips run both vertically and horizontally at regular intervals, then the style is known as power mesh. The total number of strips and interval distance is solely dependent on the ASIC core power consumption. As the ASIC core power consumption (dynamic and static) increases, the distance of power and ground strip intervals increases. This increase in the power and ground strip intervals is used mainly to reduce overall ASIC voltage drop, thereby improving ASIC design performance.

In addition to the core power and ground ring, macro power and ground rings need to be created using proper vertical and horizontal metal layers. A macro ring encloses one or more macros, completely or partially, with one or more sets of power and ground rings. Another important consideration is that when both analog and digital blocks are present in an ASIC design, there is a need for special care to ensure that there is no noise injection from digital blocks or core into the sensitive circuits of analog blocks through power and ground supply connections. Much of this interference can be minimized by carefully planning the power and ground connections for both digital core and analog blocks. There are several methods to improve the noise immunity and reduce interference.

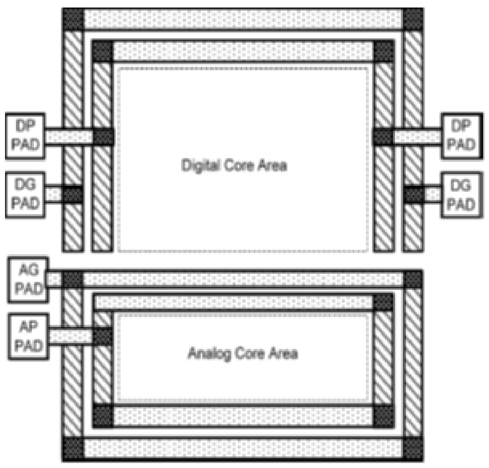


Figure 2-6 Decoupled Analog and Digital Core Power Supply

The most effective method is to decouple the digital and analog power and ground by routing the digital power/ground (DP and DG) and analog power/ground (AP and AG) supply connections separately as shown in Figure 2-6. However, this decoupling will not be complete if there is ground connectivity from the die substrate through standard cells' ground (i.e. source). In order to make sure that analog circuits are completely decoupled from digital circuits, one needs to separate the substrate from the ground in the standard cells (e.g. NWELL Process). This is not mandatory but depends on how sensitive the analog circuit is with respect to noise injection from the digital core area. It is strongly recommended to check for power and ground connectivity and/or any physical design rule violations after construction of the entire power and ground network.

macro placement

Typically, the placement of macros takes place after I/O placement, and after power and ground bus structure has been defined. Macros may be memories, analog blocks, or in the case of Hierarchical Flow, an individually placed and routed subcircuit. Proper placement of these macros has a great impact on the quality of the final ASIC design. Macro placement can be manual or automatic.

Manual macro placement is more efficient when there are few macros to be placed and their relationship with the rest of the ASIC design is known. Automatic macro placement is more appropriate if there is not enough information on which to base the initial macro placement and/or the number of macros is large.

During the macro placement step, one needs to make sure that there is enough area between blocks for interconnections. This process (commonly known as channel allocation or channel definition) can be manual or can be accomplished by floorplan tools. The slicing tree is used by the floorplan algorithm for slicing floorplan during macro placement and to define routing channels between the blocks.

Most of today's physical design tools use a global placer to perform the automatic initial macro placement based on connectivity and wire length reduction. Wire length optimization is the most prevalent approach in automatic macro placement. With increases in the number of embedded blocks such as memories, placing macros of varying sizes and shapes without a good optimization algorithm can result in fragmentation of placement and routing space that can prevent a physical design from being able to complete the final route. One of the basic algorithms used for automatic macro placement considers that macros are connected to each other by nets and are supposed to exert attractive forces on each other by means of wire length proportional to the distance between these macros. Automatic macro placement is an iterative process. During the macro placement process, macros are free to move until the equilibrium or optimum wire length is achieved. It is interesting to note that in this algorithm, if there is no relationship between the macros, they tend to repel each other and their placement result may not be optimum. To improve the placement quality of macros that are not related to each other, one may consider simultaneous standard cell and macro placement provided the physical design tool can deal with both macro and standard cell placement at once. In terms of algorithms, while commercial physical design tools have considerably improved in the past few years, automatic macro placement is still in the early stages of development compared to standard cell placement.

The implementation challenge associated with macro placement is conceptually a time and space problem that needs to be solved simultaneously.

A well-developed macro placement algorithm must be able to handle widely differing shapes and sizes, macro orientation, congestion, and timing-driven placement. Although many improvements have been incorporated into the macro placement algorithms to ensure the quality of their placement, one might need to modify the resulting location and/or orientation in order to achieve an optimum floorplan based on the physical quality of measurement. When it is not an easy task to measure the macro placement quality of an ASIC design containing a large number of blocks, there are some basic physical measurements that one can adopt.
Given a placement solution, the physical measurements could be wire length, data flow direction (e.g. the macro placement relative to each other as well as to standard cell placement), or macro, port accessibility and related timing.

The total wire length for a given placement is a good indicator when comparing different placements of the same physical design. In order to decrease overall wire length, ensure that the chip area is not segmented by the macro's placement.

To avoid area segmentation, macros should be positioned such that the standard cell area is continuous. An area with close to 1:1 aspect ratio is recommended as it allows standard cell placers to utilize the area more efficiently and thereby reduce total wire length.

The segmented floorplan leads to an excess of wire length interconnections from the standard cells located at the bottom of the die to those at the top of the die. Thus, it is necessary that the macros be kept along the ASIC core area in order to avoid a floorplan segmentation problem. Figure 2-7 shows a problematic segmented floorplan that may lead to long interconnections between the bottom and top of the die.

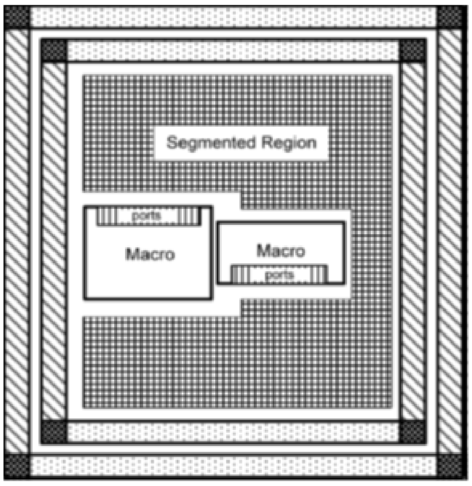


Figure 2-7 Segmented Floorplan

Another aspect of increased wire length is related to macro placement with respect to their orientation and pin placements. Depending on the macro orientation and their actual pin location, the length of the nets connecting to the macros can be different, and can have significant impact on the routing optimization process. With respect to wire length reduction, macros should be oriented such that their ports are facing the standard cells, or core area, and their orientation should match the available routing layers. Thus, any macro placement algorithm requires computing the macro's interconnect distance by including proper orientation and pin positions. Figure 2-8 shows a floorplan with macro ports facing the standard cell region, thereby minimizing localized increase in wire length.

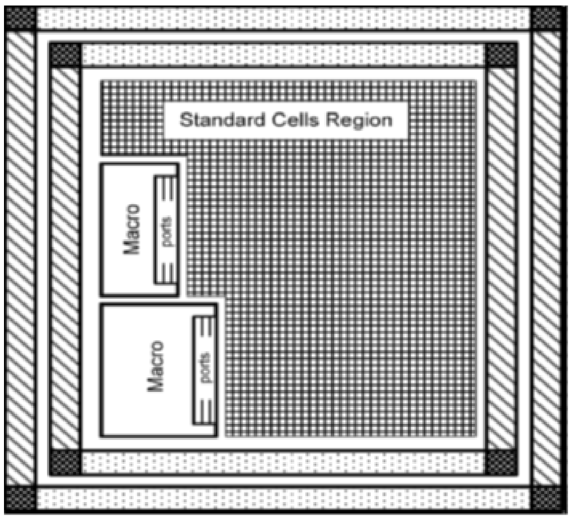


Figure 2-8 Floorplan with Macros Facing Standard Cells Region

Another aspect of physical measure is that of macro placements relative to each other, as well as to the standard cell placement, and the macro port accessibility has a direct impact on the chip's final routing. The macro placement, and thereby the quality measure, can be determined by the analysis of routing congestion produced by a global router.

Most global routers are capable of producing both graphical and text statistical reports. The graphical report, also referred to as the congestion map, provides a visual aid to see where routing congestion exists (e.g. hot spots). The statistical report is a good indication of how much a physical design is congested.

The most common scenarios that cause physical design routing congestion are: there may not be enough space between the macros to provide routing channels–especially for I/O connections and macros (if these macros are placed at the ASIC periphery and over the macro); routing is prohibited; or standard cell trap pockets may be around the edges of macros or within the corners of the floorplan. Standard cell trap pockets are long, thin channels between macros. If many cells are placed in these channels, routing congestion can result. Therefore, these channels need to be kept free for most standard cells and should be available for repeater or buffer insertion (if this type of insertion is supported by the physical design tool). Figure 2-9 shows a floorplan with a standard cell trap pocket.

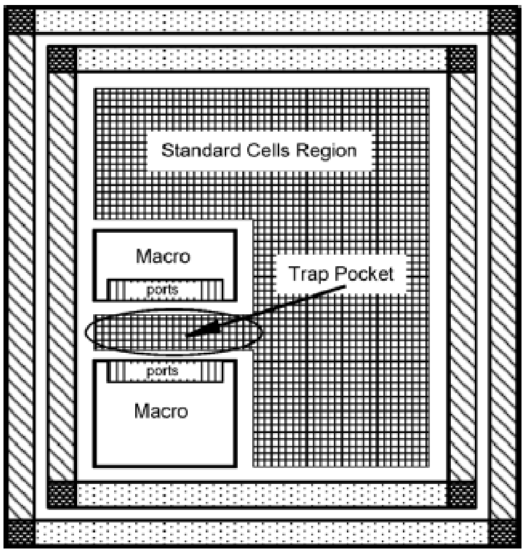


Figure 2-9 Floorplan with Standard Cells Trap Pocket

After macro placement and before performing global routing, most physical designs require keep-out or buffer-only regions to be defined by drawing a blockage layer over an area containing macros to prevent the placer from moving any standard cells into those regions. Naturally, the wires that are used in keep-out regions have a tendency to be long. By allowing buffer insertion in those areas by using a buffer-only region (or blockage), the placer will taper these long nets and thus avoid the long transition times associated with them. These blockage layers are created over pre-placed macros such that their power and ground rings are covered.

Blockage layers are also used to relieve routing congestion around the macro's corners. When a macro is blocked on many routing layers, wires have a tendency to detour around corners and connect to nearby standard cells thereby creating routing congestion at the corner of the macros.

To reserve more resources for the router, one can draw a blockage layer at these corners. These blockage regions can be simple or gradual as illustrated in

Figure 2-10.

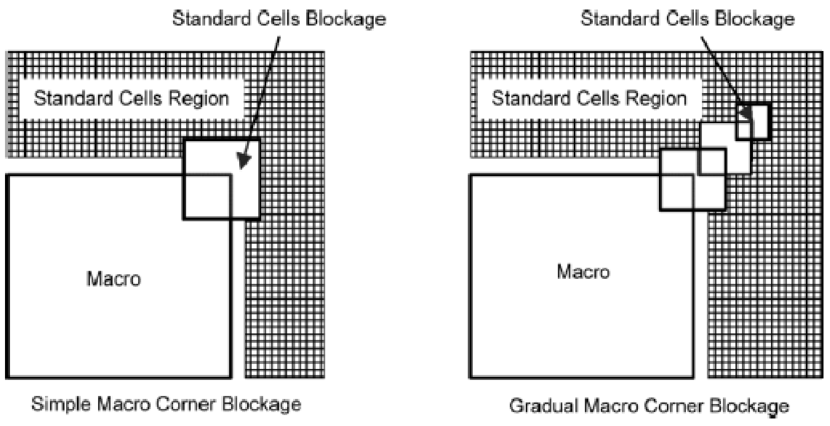


Figure 2-10 Macro Corner Standard Cells Blockage

After refinement of floorplan and macro placement, standard cells are placed and connectivity analysis is performed. Connectivity analysis is the process of studying the connections between macro pairs, macro, I/O pads, and related standard cell instances. Connectivity analysis is also used to identify macros that have substantial direct connectivity and to refine their locations accordingly.

This analysis is conducted by using what is known as fly lines. When fly lines are activated through physical design or place-and-route tools, Graphic User Interface (GUI) displays the lines that mark the connections between currently selected instances (e.g. standard cells, macros, or I/O pads). Using fly lines, one can analyze and identify situations where moving or rotating macros will yield shorter wire lengths that improve the overall ASIC routability during the floorplanning stage of the physical design cycle.

A.3.2 placement

Standard cell placement is the most important and challenging phase in ASIC physical design.

The goal of standard cell placement is to map ASIC components, or cells, onto positions of the ASIC core area, or standard cell placement region, which is defined by rows. The standard cells must be placed in the assigned region (i.e. rows) such that the ASIC can be routed efficiently and the overall timing requirements can be satisfied. Standard cell placement of an ASIC physical design has always been a key factor for achieving physical designs with optimized area usage, routing congestion, and timing behavior. Almost all of today's physical design tools use various algorithms to place standard cells automatically. Although these placement algorithms are very complex and are being improved frequently, the basic idea has remained the same.

In the early days of physical design, the total area for placing standard cells consisted of the area required for the standard cell rows and the area required for channel routing. With advancement in place-and-route tools, standard cell channel routing has almost vanished because all place-and-route tools today are capable of routing over the standard cells. Over-standard-cell routing utilizes all empty space above the standard cells. This allows physical designers to create an ASIC that is as compact as possible without creating extra channels for routing purposes.

With the disappearance of routing channels, the routing congestion problem has become more important. During standard cell placement, excessive congestion resulting in a local shortage of routing resources must be avoided. In over-standard-cell routing, the objective of most place-and-route tools has been to utilize all the available core area to prevent routing overflow. This routing overflow accounts for an increase in ASIC device size and results in performance degradation.

Standard cell placement may be thought of as an automatic process that requires less physical designer intervention. However, a number of design constraints that can be applied during standard cell placement to achieve optimal ASIC design with respect to area, performance, and power. These constraints can be congestion, timing, power, or any combination thereof.

Most place-and-route tools use a two-step approach to place standard cell instances. These steps are global and detail placement. The objective of the global placement algorithm is to minimize the interconnect wire lengths, whereas the objective of the detail placement algorithm is to meet design constraints such as timing and/or congestion, and to finalize the standard cell placement.

Global Placement

When the floorplan is first created, standard cells are in a floating state. This means that they are placed arbitrarily in the ASIC core and have not been assigned to a fixed location within the standard cell rows. At this time one can partition the standard cell area and assign a group of cells to these partitions, or simply group a set of standard cells.

Almost every place-and-route tool supports cluster and region options. These two options are used to guide placement algorithms during standard cell placement.

Cluster refers to a group of standard cells that, during placement, are placed near each other. The location of the cluster is undefined until all standard cells have been placed. This option is mainly used to control the closeness of timing-critical components during placement and resembles a module definition in the structural netlist. Since the development of placement algorithms (e.g. interconnect driven), this option has been rarely used – except in very special cases. An example of a standard cell cluster is shown in Figure 3-1.

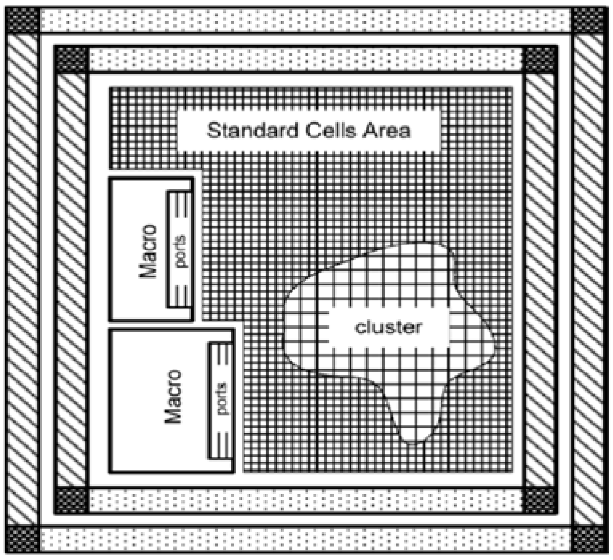


Figure 3-1 Standard Cell Cluster

Region is very similar to the cluster method with the exception that the location of the region is defined prior to standard cell placement. The way this option is implemented is that a cluster or group of standard cells is created and then assigned to a particular area on the core ASIC.

Regions can be soft or hard. Soft region, is physical constraint where logical module is assigned to a location in the core and boundary of the region, it is subject to change during standard cell placement. Hard region, however, is more rigorous than the soft region and defines a physical partition for modular design. It has "hard" boundaries that prevent standard cell crossing during placement. Using the hard region option, one must define the location as well as the shape of the region. This option is used primarily for timing related issues such as grouping clock, voltage, or threshold voltage domains.

In addition, a region can be exclusive or non-exclusive. An exclusive region only allows standard cells assigned to the region to be placed within the region. On the other hand, a non-exclusive region will allow standard cells that do not belong to the region to be placed within it. A hard region (or an exclusive region with a predefined physical boundary) might be used to enforce a floorplan consisting of separate blocks. This approach is useful for dividing the ASIC core area into regions that have different functions or physical aspects.

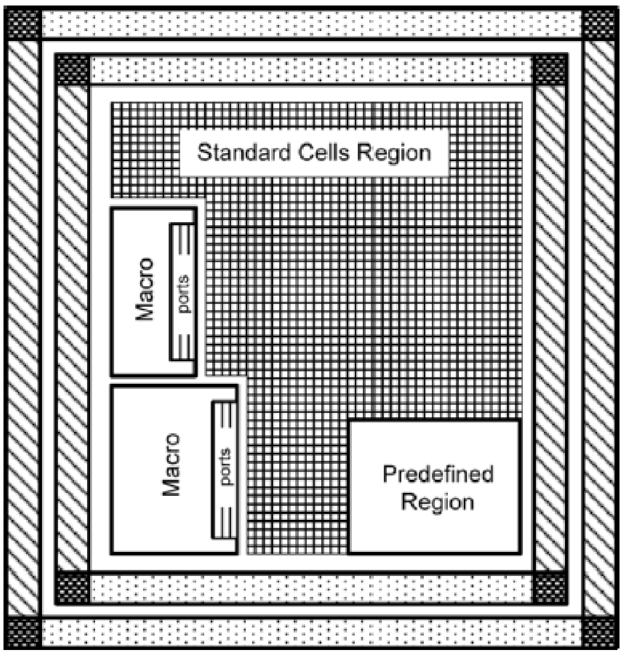


Figure 3-2 Predefined Region

For example, a hard region can be used to partition the ASIC core area so one region has a different voltage from the rest of the design or other regions as shown in Figure 3-2.

After clusters and regions are defined, global placement algorithms begin distributing standard cell instances uniformly across the available ASIC core and use a method of estimation to minimize wire lengths.

During this time, ASIC design is recursively partitioned along alternatively horizontal and vertical cut lines, and standard cell instances are assigned to rectangular bins, or slots, taking partitioning into account. Then, instances in each bin will be moved across each cut line in order to minimize the number of connections between each partition.

The procedure of partitioning and moving standard cell instances across cut lines terminates when certain stop criteria are satisfied (e.g. total number of standard cell instances in each bin). After design partitioning is completed, a legalization step is executed to remove any standard cell overlap and fit current placement into row structures.

These types of global placement algorithms are classified as partition-based. There are two main cost functions associated with partition-based algorithms: to reduce the total wiring or routing length and to distribute the standard cell instances homogeneously in the ASIC core area such that optimal equilibrium among vertical and horizontal routing is achieved.

As mentioned earlier, the global placement algorithm defines the initial, or preliminary, standard cell instance location. During this time, one can use buffering optimization to overcome problems associated with high fan-out nets, long wires, and logic restructuring if the initial timing indicates that the critical timing paths have large timing violation.

In addition, if the ASIC design uses scan testing methodology, the original data output port of a register that connects to the next scan data input port of a register may be reordered, depending upon their locations.

The scan reordering improves the routing congestion if two connected registers are located far from each other. Because this scan-chain reordering is purely connectivity based, it can lead to hold violations at some of the scan data input ports with respect to the input clock. These types of violations are usually resolved after clock tree synthesis.

The problem with nets that have a large number of fan-outs, such as a reset signal, is that one source drives many standard cells across the ASIC core. Although these nets are not critical from a timing perspective, they have a strong impact on the core routing area due to their global nature. Thus, reducing the total number of these fan-outs will improve the overall routing of the ASIC core area. Reducing the number of fan-outs (or connections) to between 40 and 50, to which one standard cell connects, is reasonable.

Long wires are not the same as high fan-out nets as they are not global in nature. They have very small fan-outs, but the driver instance is located far from the receiver. Often this situation is a result of the receiver having very strong connectivity to instances other than the driver.

These types of long wires are highly resistive and can create large input transition times at the input port of the receiver cell. This large input transition time increases the receiver cell's propagation delay. Therefore, it is beneficial to segment these long wires using buffers.

Another timing optimization that is used during global placement is logic restructuring. Logic restructuring is mainly supported by physical synthesis tools that combine several primary logic functions into a few standard cells, decompose functional gates into their equivalent primary logic gates, and/or duplicate combinational logic (cloning).

The logic restructuring algorithm used during global placement mainly focuses on logic reconstruction of critical paths that are not meeting required timing constraints. The objective of the algorithm is to rearrange logic in the critical paths such that the timing constraints are met.

Detail Placement

Once all standard cell instances are placed globally, a detail placement algorithm is executed to refine their placement based on congestion, timing, and/or power requirements.

Congestion refinement or congestion-driven placement is more beneficial to ASIC designs with very high density, and the objective of the detail placer is to distance standard cell instances from each other such that more routing tracks are created among them.

The quality of congestion placement directly relates to how well the global placer partitions the design and could have a negative impact on the device size and performance.

For minimal device size, one may use more routing layers. In determining the total number of routing layers to use, it is imperative to consider the trade-off between increasing the device size and using extra routing layers. In some instances, it may be more economical to increase the device size rather than adding extra routing layers (i.e. extra mask).

Timing-driven placement algorithms have been classified as either net or path based. Net-based schemes try to control the delay on a signal path by imposing an upper-bound delay or by assigning a weight to each net. Path based approaches apply constraints to delay paths of small sub-circuits (the disadvantage of path-based algorithms is the fact that it is impossible to enumerate all paths within a design).

The major challenge in timing-driven placement is to optimize large sets of path delays without enumerating them in the ASIC design. This optimization is accomplished by interleaving weighted connectivity-driven placements with timing analysis that annotates individual instances, nets, and path delays with design constraint information.

To meet these types of design constraint, various placement techniques have been proposed or used. The most well-known detail placement method is simulated annealing. Not only is simulated annealing efficient, it can also handle complex design constraints.

A.3.3 clock tree synthesis

Clock Tree Synthesis (CTS) is a process which make sure that the clock gets distributed evenly to all sequential elements in a design.
CTS is the process of insertion of buffers or inverters along the clock paths of ASIC design in order to achieve minimum skew or balanced skew.
In ICs, clock consumes around half of the total power consumption. Here clock gating technique helps to reduce power consumption by the clocks.
Goals of CTS:

- To meet clock tree design rule constraints such as maximum transition, maximum load capacitance and maximum fanout.
- To meet clock tree targets such as minimum skew and minimum insertion delay.

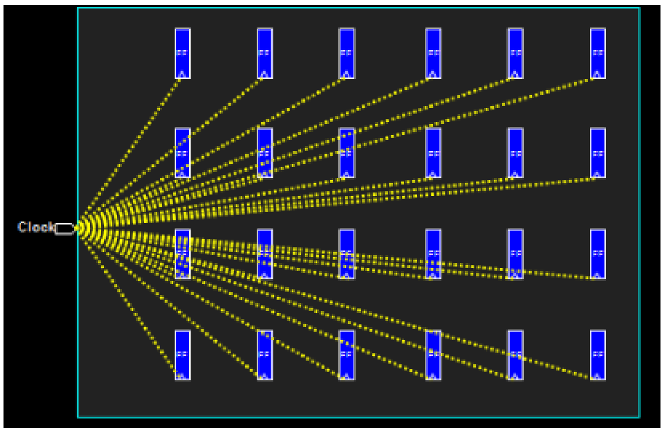
Checklist before CTS:

- Placement is completed and optimized.
- Power & Ground (PG) nets are pre-routed.
- Estimated congestion – Acceptable.
- Estimated Max trans/Cap - No violations.
- High Fan-out Nets are synthesized with buffers (clocks are not buffered still).

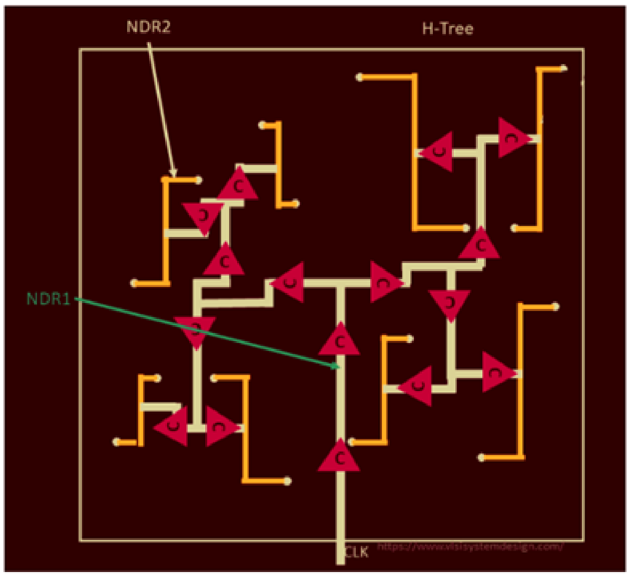
Checklist after CTS:

- Skew report.
- Clock tree report.
- Timing reports for setup and hold.
- Power and area report.

After placement stage, all the cells including macros and standard cells are placed. But the clock is still ideal. We only optimize the data paths at placement stage with buffer insertion and cell sizing, but no change is done in the clock net.



Just look into the above figure. Here the clock port connects all the synchronous elements in the design. The fanout of the particular port driver is too high and also the clock is not reaching all the flops at a time. The clock network delays are different. So, the skew value is very high, which is not recommended in a design. That's why CTS is performed to balance the clock net by adding buffers and minimize the skew as much as possible (ideally the skew value is zero). After the clock tree synthesis. the clock net is buffered and the NDR rule is also applied as shown in the below figure.



Difference between HFNS and CTS?
HENS (High Fanout Net Synthesis) used in placement stage which uses buffers and inverters of relaxed rise and fall times. But in CTS (Clock Tree Synthesis), buffers and inverters of equal rise and fall times are used. NDR rules are also used for clock tree routing.
Note: The reason why the clock is defined as ideal in placement stage is, if we don't define clock as ideal, the HFNS will insert buffers, inverters and other optimizations in clock net also. But the clock nets need buffers and inverters of equal rise and fall times, not the normal buffers used by HENS.
Difference between Clock buffers and Normal buffers:

- Clock buffers have equal rise and fall time.
- Normal buffers have unequal rise and fall time.
- Clock buffers are usually designed such that an input signal with 50% duty cycle produces an output signal with 50% duty cycle.

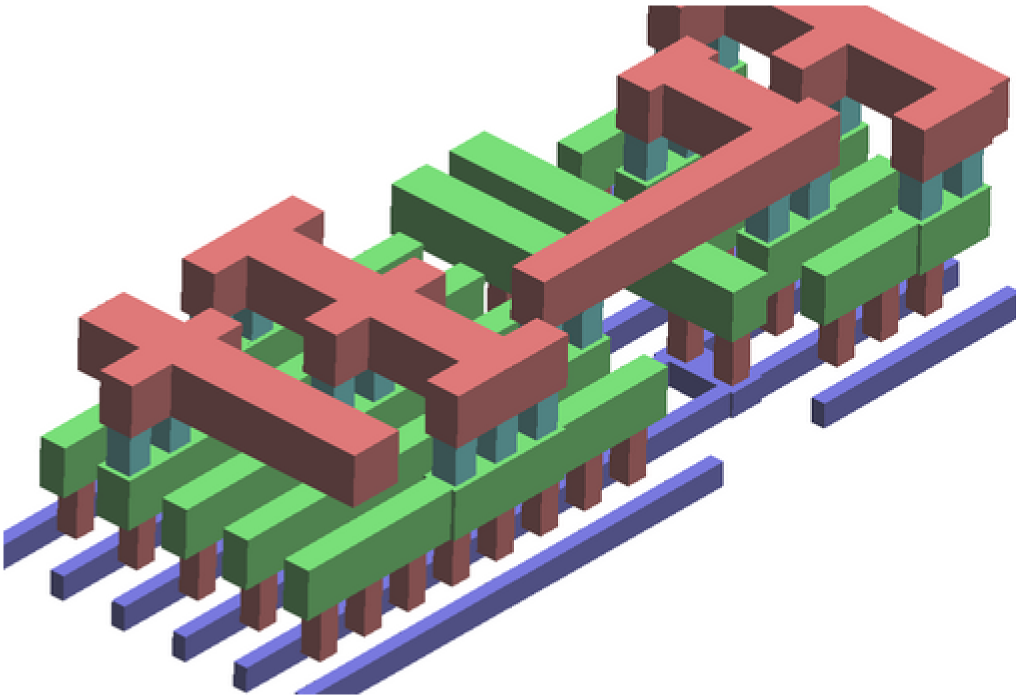
Note: Buffers have unequal rise and fall times is because of the difference in PMOS and NMOS resistances. Normally the resistance of the PMOS is two times more than that of NMOS. So, the time taken for charging the load capacitor (rise time) through PMOS is more than the discharging time through NMOS (fall time). For designing clock buffers, we should make both the resistances of PMOS and NMOS equal. We have to increase the width of PMOS such that its resistance become equal to NMOS resistance. These clock buffers are specially designed for clock path. The main disadvantage of clock buffer is its big size because of increased width of PMOS. So, these buffers will lead to increase the chip area.
Non-Default Clock Routing (NDR)
Non-Default Routing (NDR) rules are double spacing, double width and shielding. These are used to applied on the clock nets to make it less sensitive to crosstalk and electromigration effects.

A.3.4 Routing

Routing is the stage after CTS where the interconnections are made by determining the precise paths for each net. This includes the interconnection of the standard cells, the macro pins, the pins of the block boundary or the pads of the chip boundary. After CTS, the tool will be having the information about the exact locations of the standard cells, the pins, the 10 pons and the pads.

The logical connectivity is defined by the netlist and the design rules are defined in the technology files are available to the tool. In routing stage, metal and vias are used to create the electrical connection in layout so as to complete all connections defined by the netlist.

So, in short, routing can be termed as allocating set of wires in the routing space that connects all the nets in the netlist by using certain design rules for the metals and vias used in doing so.



Goals of Routing:

- Establishing the entire connectivity of the design with minimum number of vias and optimized total wire length. To meet the timing constraints.
- No WS errors (Layout vs Schematic) i.e., the all the connections described in the netlist are completed physically.
- No DRC (Design Rules Check) violations in doing so.
- Complete routing within the area of the design.

Inputs:

Design which is done with placement, CTS and optimization.
output:

Design with completed interconnection and geometric layout of the nets.
Prerequisites and Checks:

- Timing DRC and QoR post CTS must be acceptable.
- Acceptable global routing congestion.
- HFNS should be less than the specified limit.
- Check for overlapping cells, if any.
- Check for any blocked pins, ports or PG connection.

Stages of Routing:

- Global Routing.
- Track Assignment.
- Detailed Routing.
- Search and Repair.

Each stages of routing are described below.

Global Routing

In global routing, the region to be routed are divided into sectors (tiles/rectangles) called global routing cells or gcells.
Then it decides tile to tile path for the nets and simultaneously trying to optimize the length, without actually making any physical connection.
The routing capacity of each gcell depends on the blockages, routing tracks, pin density inside it.
This rough routing is done on the basis of available tracks in the region.
If the required routing resources are greater than the available routing resources, then it will lead to congestion.
So, it is called coarse grain routing assignment.

Objectives of global routing:

- Minimize total overflow.
- Minimize total wire length.
- Minimize total run time for carrying out routing process.

Track Assignment

After the gcell estimation, tracks are assigned to each global route.
The tracks are assigned in vertical and horizontal direction for each partition.
The direction of routing is dependent on the metal used, which has preferred routing direction. For e.g. If Metal 1 has routing direction Horizontal, then Metal 2 has direction in this stage, the global routes are replaced with metal layers, which has many DRC violations, Signal Integrity (SI) and timing violations.

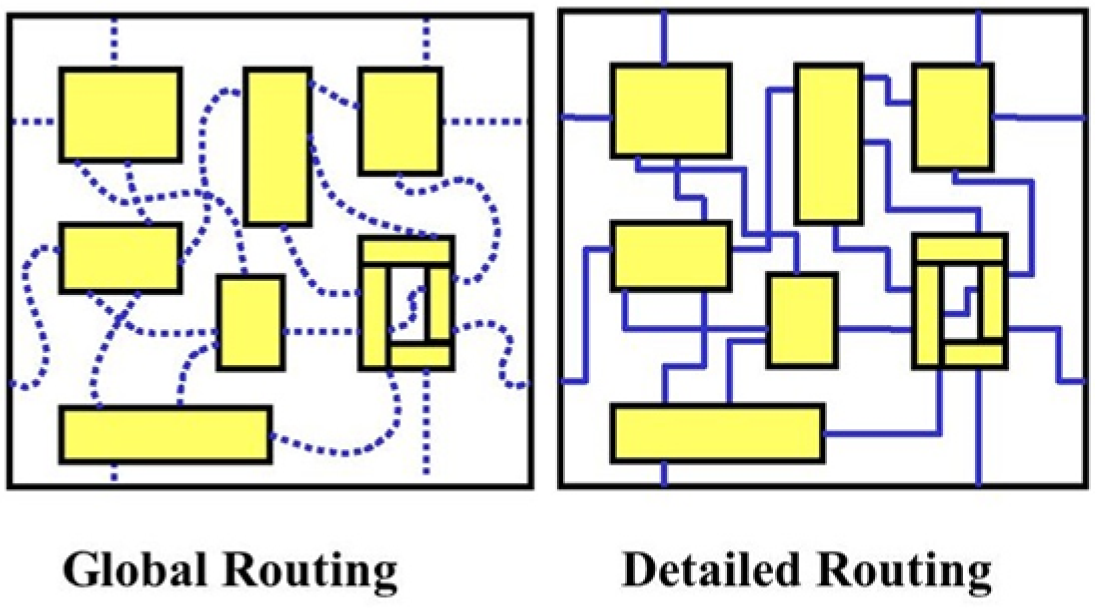
Detailed Routing

In detailed routing, the router uses the scheme made in the global routing and track assignment phases to lay metals to connect the nets to the pins.
The violations that were created in the previous stage, will be fixed by multiple

iterations of routing, so that no connections will be left short, open or spacing violations.

First, the block is divided into specific areas called the Sboxes (switch boxes) which comprises of multiple gcells.

These boxes are in alignment with the gcell boundary.



Search and Repair

It is done along with detailed routing, specifically after the primary iteration.

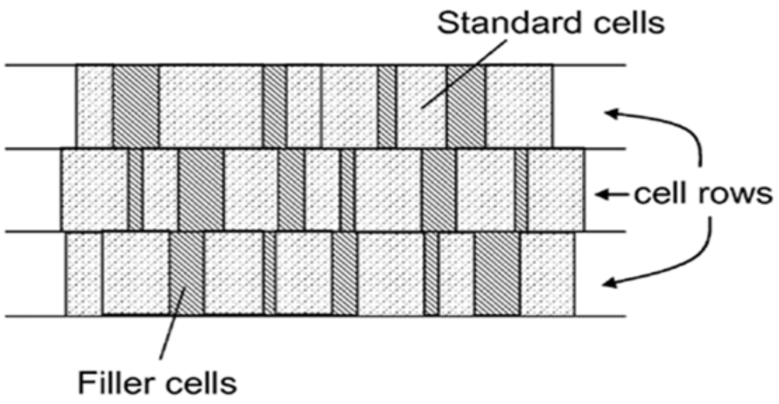
The shorts and spacing violations are sorted and is fixed.

A.3.5 Chip Finish

chip finish stage comes after routing optimization, where filler cells and metal fills are added and also wire spreading to meet the DRC rules. three steps are mainly performed in this stage.
1. Adding Filler Cells.
2. Adding Metal Fills.
3. wire spreading.

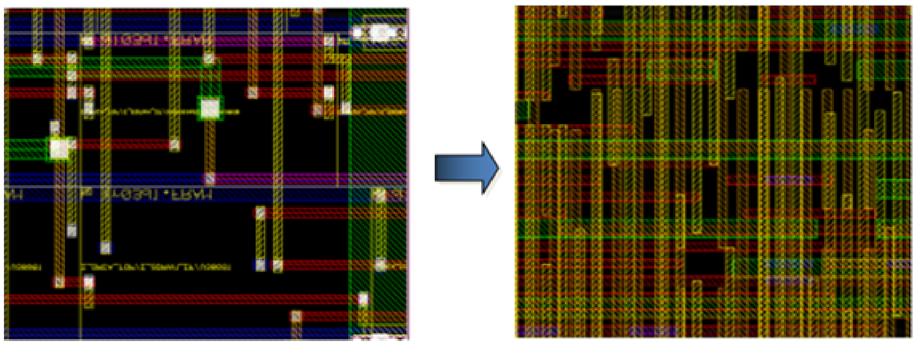
Adding Filler Cells

Filler cells are used for rail continuity and to fill up gaps between standard cells in the rows, and thereby reducing the DRC violations created by the base layers. Filler cells are physical-only cells designed in such a way that they contain only n-well, p-well & power rails.
It is also possible to reduce the IR drop by inserting de-cap filler cells, but this comes at a cost of higher leakage currents.



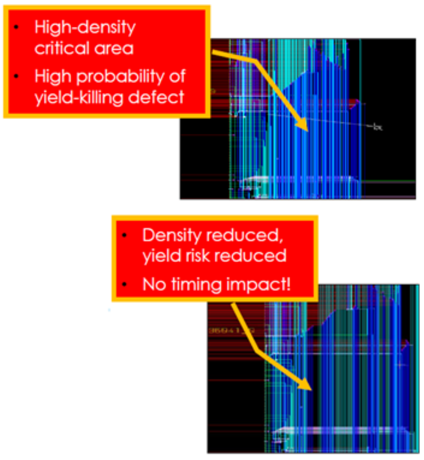
Adding Metal Fills

The metal fills also known as dummy metal layers, are small, floating metal nets, inserted in empty spaces in the design after post-route optimization in order to maintain uniformity in metal layer density.
These are added to meet the metal density DRC rules (density violations) which are mandatory by most manufacturing processes.



wire spreading

Random particle defects during manufacturing may cause shorts or opens during the fabrication process.
Wires at minimum spacing are most susceptible to shorts.
Minimum-width wires are most susceptible to opens.
Wire spreading is one of the most effective solutions to reduce the previous problems, hence reduce yield loss. It relates to a new physical design that is able to increase the spacing between metal wires in the layout effectively and efficiently without violating any design rules.


High-density critical area
High probability of yield-killing defect
Density reduced, yield risk reduced
No timing impact!

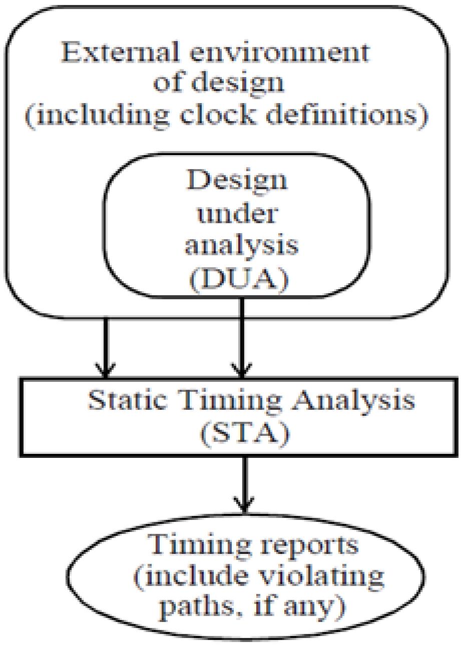
A.4 Static Timing Analysis verification

What is Static Timing Analysis?

Static Timing Analysis (also referred as STA) is one of the many techniques available to verify the timing of a digital design. An alternate approach used to verify the timing is the timing simulation which can verify the functionality as well as the timing of the design. The term timing analysis is used to refer to either of these two methods static timing analysis, or the timing simulation. Thus, timing analysis simply refers to the analysis of the design for timing issues.

The STA is static since the analysis of the design is carried out statically and does not depend upon the data values being applied at the input pins. This is in contrast to simulate based timing analysis where a stimulus is applied on input signals, resulting behavior is observed and verified, then time is advanced with new input stimulus applied, and the new behavior is observed and verified and so on.

Given a design along with a set of input clock definitions and the definition of the external environment of the design, the purpose of static timing analysis is to validate if the design can operate at the rated speed. That is, the design can operate safely at the specified frequency of the clocks without any timing violations. The following flow chart shows the basic functionality of static timing analysis. The DUA is the design under analysis. Some examples of timing checks are setup and hold checks. A setup check ensures that the data can arrive at a flip-flop within the given clock period. A hold check ensures that the data is held for at least a minimum time so that there is no unexpected pass-through of data through a flip-flop: that is, it ensures that a flip-flop captures the intended data correctly. These checks ensure that the proper data is ready and available for capture and latched in for the new state.


External environment of design (including clock definitions)
Design under analysis (DUA)
Static Timing Analysis (STA)
Timing reports (include violating paths, if any)

The more important aspect of static timing analysis is that the entire design is analyzed once and the required timing checks are performed for all possible paths and scenarios of the design. Thus, STA is a complete and exhaustive method for verifying the timing of a design.

The design under analysis is typically specified using a hardware description language such as VHDL or Verilog. The external environment, including the clock definitions, are specified typically using SDC or an equivalent format. SDC is a timing constraint specification language. The timing reports are in ASCII form, typically with multiple columns, with each column showing one attribute of the path delay.

Why Static Timing Analysis?

Static timing analysis is a complete and exhaustive verification of all timing checks of a design. Other timing analysis methods such as simulation can only verify the portions of the design that get exercised by stimulus. Verification through timing simulation is only as exhaustive as the test vectors used. To simulate and verify all timing conditions of a design with 10-100 million gates are very slow and the timing cannot be verified completely. Thus, it is very difficult to do exhaustive verification through simulation.

Static timing analysis on the other hand provides a faster and simpler way of checking and analyzing all the timing paths in a design for any timing violations. Given the complexity of present-day ASICs, which may contain 10 to 100 million gates, the static timing analysis has become a necessity to exhaustively verify the timing of a design.

The design functionality and its performance can be limited by noise. The noise occurs due to crosstalk with other signals or due to noise on primary inputs or the power supply. The noise impact can limit the frequency of operation of the design and it can also cause functional failures. Thus, a design implementation must be verified to be robust which means that it can withstand the noise without affecting the rated performance of the design. Verification based upon logic simulation cannot handle the effects of crosstalk, noise and on-chip variations.

Design flow

STA is rarely done at the RTL level as, at this point, it is more important to verify the functionality of the design as opposed to timing. Also, not all timing information is available since the descriptions of the blocks are at the behavioral level. Once a design at the RTL level has been synthesized to the gate level, the STA is used to verify the timing of the design. STA can also be run prior to performing logic optimization - the goal is to identify the worst or critical timing paths. STA can be rerun after logic optimization to see whether there are failing paths still remaining that need to be optimized, or to identify the critical paths.

At the start of the physical design, clock trees are considered as ideal, that is, they have zero delay. Once the physical design starts and after clock trees are built, STA can be performed to check the timing again. In fact, during physical design, STA can be performed at each and every step to identify the worst paths.

In physical implementation, the logic cells are connected by interconnect metal traces. The parasitics RC (Resistance and Capacitance) of the metal traces impact the signal path delay through these traces. In a typical nanometer design, the parasitic of the interconnect can account for the majority of the delay and power dissipation in the design. Thus, any analysis of the design should evaluate the impact of the interconnect on the performance characteristics (speed, power, etc.). As mentioned previously, coupling between signal traces contributes to noise, and the design verification must include the impact of the noise on the performance.

At the logical design phase, ideal interconnect may be assumed since there is no physical information related to the placement; there may be more interest in viewing the logic that contributes to the worst paths. Another technique used at this stage is to estimate the length of the interconnect using a wireload model. The wireload model provides estimated RC based on the fanouts of a cell.

Before the routing of traces are finalized, the implementation tools use an estimate of the routing distance to obtain RC parasitics for the route. Since the routing is not finalized, this phase is called the global route phase to distinguish it from the final route phase. In the global route phase of the physical design, simplified routes are used to estimate routing lengths, and the routing estimates are used to determine resistance and capacitance that are needed to compute wire delays. During this phase, one cannot include the effect of coupling. After the detailed routing is complete, actual RC values obtained from extraction are used and the effect of coupling can be analyzed. However, a physical design tool may still use approximations to help improve run times in computing RC values.

Figure A.28 shows how to use static timing analysis during design flow:

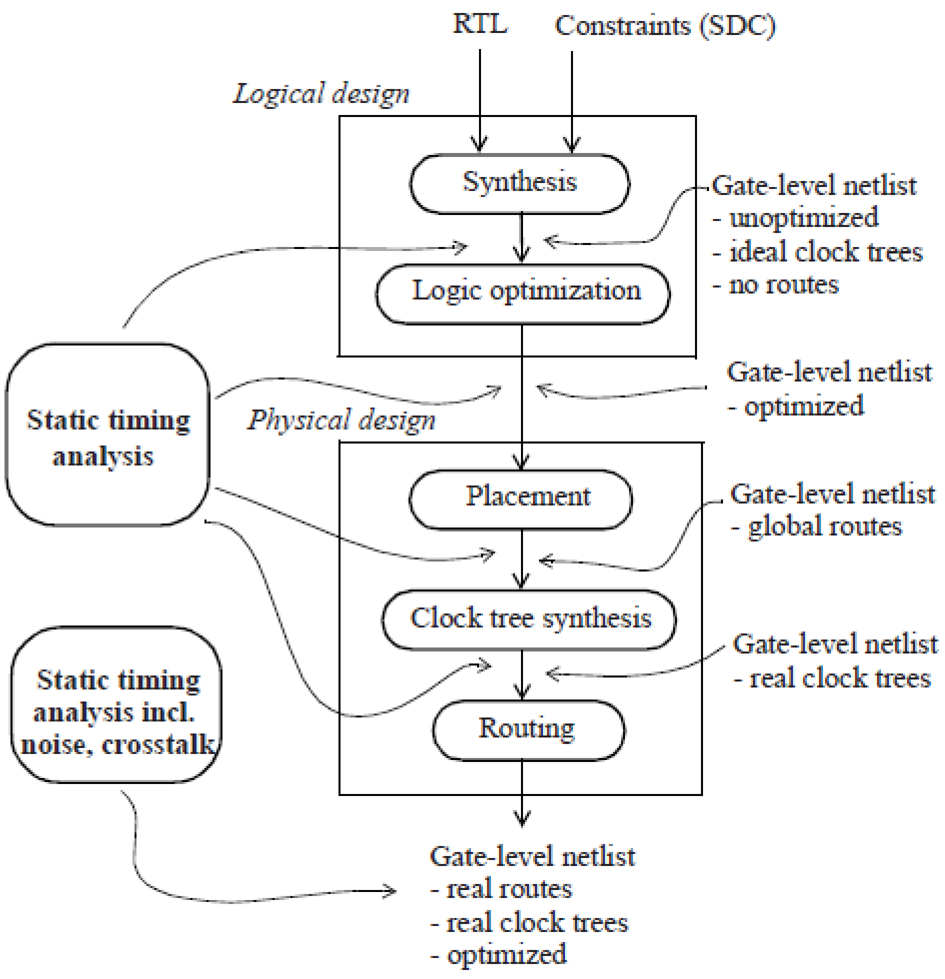


Figure A.28: The design Flow

Limitations of Static Timing Analysis

While the timing and noise analysis do an excellent job of analyzing a design for timing issues under all possible situations, the state-of-the-art still does not allow STA to replace simulation completely. This is because there are some aspects of timing verification that cannot yet be completely captured and verified in STA.

Some of the limitations of STA are:
i. Reset sequence: To check if all flip-flops are reset into their required logical values after an asynchronous or synchronous reset. This is something that cannot be checked using static timing analysis. The chip may not come out of reset. This is because certain declarations such as initial values on signals are not synthesized and are only verified during simulation.

ii. X-handling: The STA techniques only deal with the logical domain of logic-0 and logic-1 (or high and low), rise and fall. An unknown value X in the design causes indeterminate values to propagate through the design, which cannot be checked with STA. Even though the noise analysis within STA can analyze and propagate the glitches through the design, the scope of glitch analysis and propagation is very different than the X handling as part of the simulation-based timing verification for nanometer designs.

iii. PLL settings: PLL configurations may not be loaded or set properly.

iv. Asynchronous clock domain crossings: STA does not check if the correct clock synchronizers are being used. Other tools are needed to ensure that the correct clock synchronizers are present wherever there are asynchronous clock domain crossings.

v. IO interface timing: It may not be possible to specify the IO interface requirements in terms of STA constraints only. For example, the designer may choose detailed circuit level simulation for the DDR1 interface using SDRAM simulation models. The simulation is to ensure that the memories can be read from and written to with adequate margin, and that the DLL2, if any, can be controlled to align the signals if necessary.

vi. Interfaces between analog and digital blocks: Since STA does not deal with analog blocks, the verification methodology needs to ensure that the connectivity between these two kinds of blocks is correct.

vii. False paths: The static timing analysis verifies that timing through the logic path meets all the constraints, and flags violations if the timing through a logic path does not meet the required specifications. In many cases, the STA may flag a logic path as a failing path, even though logic may never be able to propagate through the path. This can happen when the system application never utilizes such a path or if mutually contradictory conditions are used during the sensitization of the failing path. Such timing paths are called false paths in the sense that these can never be realized. The quality of STA results is better when proper timing constraints including false path and multicycle path constraints are specified in the design. In most cases, the designer can utilize the inherent knowledge of the design and specify constraints so that the false paths are eliminated during the STA.

viii. FIFO pointers out of synchronization: STA cannot detect the problem when two finite state machines expected to be synchronous are actually out of synchronization. During functional simulations, it is possible that the two finite state machines are always synchronized and change together in lock-step. However, after delays are considered, it is possible for one of the finite state machines to be out of synchronization with the other, most likely because one finite state machine comes out of reset sooner than the other. Such a situation cannot be detected by STA.

ix. Clock synchronization logic: STA cannot detect the problem of clock generation logic not matching the clock definition. STA assumes that the clock generator will provide the waveform as specified in the clock definition. There could be a bad optimization performed on the clock generator logic that causes, for example, a large delay to be inserted on one of the paths that may not have been constrained properly. Alternately, the added logic may change the duty cycle of the clock. The STA cannot detect either of these potential conditions.

x. Functional behavior across clock cycles: The static timing analysis cannot model or simulate functional behavior that changes across clock cycles.

References

[1] Patterson, D. and Waterman, A., 2017. "The RISC-V Reader: An Open Architecture Atlas". Strawberry Canyon.

[2] "Introduction to RISC-V". Jielun Tan, James Connolly. February, 2019 (https://www.eecs.umich.edu/courses/eecs470/labs/rv.pdf).

[3] Waterman, A. and Asanovic, K., 2019. "The RISC-V instruction set manual, volume I: Unprivileged ISA document", version 20190608-baseratified. RISC-V Foundation, Tech. Rep.

[4] Github, "Pulp Hardware Reference Manual", (https://github.com/pulp-platform/pulp/tree/master/doc), version 1.0.0, February 11, 2019.

[5] Traber, A., Gautschi, M. and Schiavone, P.D., 2017. "RI5CY: User Manual".

[6] Andreas, T. (2016). "PULPino: A small single-core RISC-V SoC" [PowerPoint Slides]. Retrieved from (https://riscv.org/wp-content/uploads/2016/01/Wed1315-PULP-riscv3_noanim.pdf).

[7] Jiang, J.H.R. and Devadas, S., 2009." Logic synthesis in a nutshell". In Electronic Design Automation (pp. 299-404). Morgan Kaufmann.

[8] Sharief, Sh. (2019). "Logic synthesis,floorplan&placement" [PowerPoint slides]. Retrieved from (https://www.slideshare.net/ShariefShaikSharief/logic-synthesisflootplanplacement).

[9] Deiptli, D. (2012). "Topographical Synthesis" [PowerPoint slides]. Retrieved from (https://www.slideshare.net/deeptishankardas/tograhical-synthesis).

[10] Synopsys, "Design Compiler Guide", Synopsys, Inc, September 2011.

[11] Jessiman, G. (2016, September 1). Re: Why do we do Equivalence Check on RTL vs Gate Netlist? [Discussion post]. Mentor Graphics verification

academy (https://verificationacademy.com/forums/systemverilog/why-do-we-do-equivalence-check-rtl-vs-gate-netlist).

[12] Gayathri. (2013, February 7). ”Formal verification basics”. Retrieved from (http://tech.tdzire.com/formal-verification-basics/).

[13] VLSI Guide. (2018, July 4). ”Clock Tree Synthesis (CTS)”. Retrieved from (https://www.vlsiguide.com/2018/07/clock-tree-synthesis-cts.html).

[14] VLSI Guide. (2018, July 4). ”Routing”. Retrieved from (https://www.vlsiguide.com/2018/07/routing.html).

[15] Bhatnagar, H., 2002. ”Advanced ASIC chip synthesis”. Kluwer Academic.

[16] Lavagno, L., Scheffer, L. and Martin, G. eds., 2006. ”EDA for IC implementation, circuit design, and process technology”. CRC press.

[17] Bhasker, J. and Chadha, R., 2009. ”Static timing analysis for nanometer designs: A practical approach”. Springer Science & Business Media.

[18] Gangadharan, S. and Churiwala, S., 2013. ”Constraining Designs for Synthesis and Timing Analysis”. Springer.

[19] Wang, L.T., Chang, Y.W. and Cheng, K.T.T. eds., 2009. ”Electronic design automation: synthesis, verification, and test”. Morgan Kaufmann.

[20] An, A.S.I.C., ”Physical Design Essentials”.

[21] Mohamed Mahmoud Mohamed Farag, ”ASIC DESIGN OF THE OPENSPARC T1 PROCESSOR CORE”, Faculty of Engineering, Cairo University, January 2013.